Российская академия наук
Суперкомпьютерный консорциум университетов России

Сборник трудов

Электронное издание

Международная научная конференция

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ (ПаВТ'2010)

Всероссийская научная конференция молодых ученых

ПАРАЛЛЕЛЬНЫЕ И РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ

г. Уфа, 29 марта – 2 апреля 2010 г.

Челябинск,
Издательский центр ЮУрГУ
2010

УДК 004.75
П 18

Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 29 марта – 2 апреля 2010 г.) [Электронный ресурс] – Челябинск: Издательский центр ЮУрГУ, 2010. – 711 с. – Режим доступа: http://omega.sp.susu.ac.ru/books/conference/PaVT2010

ISBN 978-5-696-03987-9

Данный сборник содержит статьи, включенные в программу Международной научной конференции «Параллельные вычислительные технологии». Конференция проводится с 29 марта по 2 апреля 2010 года. Подробную информацию о конференции можно найти в сети Интернет по адресу `http://agora.guru.ru/pavt`.

Отпечатано с авторских оригиналов.

Одобрено Советом механико-математического факультета ЮУрГУ

Рецензенты:
С.М. Абрамов, член-корреспондент РАН
В.В. Воеводин, член-корреспондент РАН

Ответственные за выпуск:
Л.Б. Соколинский, доктор физ.-мат. наук
К.С. Пан

Конференция проводится при поддержке
Российского фонда фундаментальных исследований

ISBN 978-5-696-03987-9

Содержание

Короткие статьи **378**

Плакаты **651**

Всероссийская конференция молодых ученых «Параллельные и распределенные вычисления»

ОРГАНИЗАТОРЫ

Учредителями конференции являются Российская академия наук и Суперкомпьютерный консорциум университетов России

СПОНСОРЫ

Платиновые спонсоры:
Корпорация Intel
Компания Т-Платформы

Золотые спонсоры:
Компания РСК СКИФ
Корпорация NVIDIA
Корпорация IBM

Серебряные спонсоры:
Корпорация AMD
Компания ТЕСИС

Спонсор секции:
Компания Делкам-Урал

ИНФОРМАЦИОННАЯ ПОДДЕРЖКА

Информационно-аналитический центр Parallel.ru
Газета «Поиск»
Информационно-аналитический журнал «Rational Enterprise Management»
Международный информационно-аналитический журнал «CAD/CAM/CAE Observer»

ПРОГРАММНЫЙ КОМИТЕТ

Председатель программного комитета:
Воеводин В.В., чл.-корр. РАН, НИВЦ МГУ, г. Москва

Сопредседатели программного комитета:
Абрамов С.М., чл.-корр. РАН, ИПС РАН, г. Переславль-Залесский
Четверушкин Б.Н., чл.-корр. РАН, ИММ РАН, г. Москва

Ученый секретарь программного комитета:
Соколинский Л.Б., д.ф.-м.н., ЮУрГУ, г. Челябинск

Заместитель ученого секретаря программного комитета:
Цымблер М.Л., к.ф.-м.н., ЮУрГУ, г. Челябинск

Члены программного комитета:
Абламейко С.В., чл.-корр. НАН РБ, ОИПИ НАН РБ, г. Минск
Афанасьев А.П., д.ф.-м.н., ИСА РАН, г. Москва
Бердышев В.И., чл.-корр. РАН, ИММ УрО РАН, г. Екатеринбург
Болдырев Ю.Я., д.т.н., СПбГПУ, г. Санкт-Петербург
Бухановский А.В., д.т.н, СПбГУ ИТМО, г. Санкт-Петербург
Газизов Р.К., д.ф.-м.н., УГАТУ, г. Уфа
Гергель В.П., д.т.н., ННГУ, г. Нижний Новгород
Горячев В.Д., д.т.н., ТГТУ, г. Тверь
Гузаиров М.Б, д.т.н., УГАТУ, г. Уфа
Гузев М.А., чл.-корр. РАН, ДВО РАН, г. Владивосток
Донгарра Дж. (J. Dongarra, University of Tennessee), США
Ильин В.П., д.ф.-м.н., ИВМиМГ СО РАН, г. Новосибирск
Лыкосов В.Н., чл.-корр. РАН, ИВМ РАН, г. Москва
Мейер Х. (H. Meuer, ISC General Chair), Германия
Немухин А.В., д.х.н., МГУ, г. Москва
Попов Л.Д., д.ф.-м.н., ИММ УрО РАН, г. Екатеринбург
Ситоле Х. (H. Sithole, Director of CHPC), ЮАР Сулимов В.Б., д.ф.-м.н., НИВЦ МГУ, г. Москва
Шабров Н.Н., д.т.н, СПбГПУ, г. Санкт-Петербург
Якобовский М.В., д.ф.-м.н., ИММ РАН, г. Москва

ОРГАНИЗАЦИОННЫЙ КОМИТЕТ

Председатель организационного комитета:
Гузаиров М.Б., ректор УГАТУ

Заместители председателя:
Бадамшин Р.А., проректор УГАТУ по научной и инновационной деятельности
Газизов Р.К., зав. каф. высокопроизв. вычислительных технологий и систем УГАТУ

Члены организационного комитета:
Антонов А.С., с.н.с. НИВЦ МГУ
Ахмадинуров Б.Б., руководитель предприятия ООО «M2M телематика Уфа»
Байков В.А., заведующий кафедрой математики УГАТУ
Байков Р.А., заместитель директора ИКИ при НИЧ УГАТУ
Водопьянов В.В., декан общенаучного факультета УГАТУ
Кривошеев И.А., декан факультета авиационных двигателей УГАТУ
Лукащук С.Ю., доцент каф. высокопроизв. вычислительных технологий и систем УГАТУ
Мухтаров А.Р., главный инженер ИТЦ КиТ УГАТУ
Пан К.С., программист кафедры системного программирования ЮУрГУ
Рахматуллин Р.Р., директор ИТЦ КиТ УГАТУ
Репина К.В., программист кафедры системного программирования ЮУрГУ
Соболев С.И., н.с. НИВЦ МГУ
Хисамутдинов Р.А., доцент каф. высокопроизв. вычислительных технологий и систем УГАТУ
Цымблер М.Л., доцент кафедры системного программирования ЮУрГУ
Юлдашев А.В., ассистент каф. высокопроизв. вычислительных технологий и систем УГАТУ

Индекс по фамилиям

Распараллеливание алгоритмов численного моделирования процессов перколяции с вытеснением

А.А. Яппарова, С.А. Маякова

В данной работе рассматриваются методы параллельной реализации алгоритмов численного моделирования процессов перколяции с вытеснением. Исследуется влияние граничных условий на фрактальную размерность перколяционного кластера. В зависимости от типа граничных условий выбираются различные подходы к распараллеливанию расчетных программ: распараллеливание по статистике и по пространству. Проводится сравнение ускорений, полученных для каждого из методов распараллеливания.

1. Введение

В различных прикладных задачах нефтяной промышленности производится моделирование течения флюидов (нефти, воды) в пласте. Существует множество моделей, использующих различные подходы к описанию физической сущности процессов, протекающих в нефтяном пласте. Модель перколяции с вытеснением с образованием захваченных областей достаточно хорошо отражает физические свойства процесса фильтрации и позволяет прослеживать движение фронта раздела вытесняющей и вытесняемой жидкостей. Кроме того, возможность образования «захваченных» областей позволяет учитывать несжимаемость нефти и оценивать остаточную нефтенасыщенность.

Рассмотрим вытеснение нефти водой. Если вода нагнетается очень медленно, то капиллярные силы преобладают над вязкими и динамика вытеснения полностью определяется процессами в масштабе отдельных пор. В пределе можно пренебречь всеми скачками давления как в вытесняющей (воде), так и в вытесняемой жидкости (нефти). Однако между этими двумя жидкостями сохраняется разность давлений (капиллярное давление), равное

$$p_{вытесняюще\ е} - p_{вытесняемо\ е} = \frac{2\sigma\ \cos\theta}{r}, \qquad (1)$$

здесь σ – межфазное натяжение на границе жидкостей, θ – краевой угол между поверхностью раздела и стенкой поры, а r – радиус кривизны поры в месте контакта жидкостей.

Капиллярные силы наиболее велики в самых узких местах системы пор. Таким образом, если горловины уже пор, то поверхность раздела воды и нефти быстро движется сквозь горловины и тормозится, попадая в крупные поры. В согласии с подобными простыми теоретическими моделями и экспериментальными результатами это движение можно представить в виде ряда дискретных скачков, так что на каждом временном шаге вода вытесняет нефть из самой малой из имеющихся пор, занятых нефтью.

Вытесняющая жидкость захватывает области вытесняемой жидкости. По мере продвижения вторгающейся жидкости она способна полностью охватить области, заполненные вытесняемой средой, т.е. полностью отрезать кластеры вытесняемой жидкости от узлов на входе в решетку. Это одна из причин возникновения проблемы «остаточной нефти» – жгучей экономической проблемы нефтепромыслов. Поскольку нефть несжимаема, можно сформулировать правило: вода не может вытеснить нефть из захваченных областей.

Для описания течения двух несмешивающихся жидкостей в пористой среде (в нашем случае это процесс вытеснения нефти водой) Уилкинсон и Виллемсен в начале 1980-х годов предложили модель перколяции с вытеснением (invasion percolation) [1–5]. Уилкинсон и Виллемсен предложили моделировать этот процесс в идеализированной среде, в которой пористую среду можно рассматривать как регулярную решетку с узлами и связями, играющими роль пор и связывающих их каналов (горловин). Физическая неоднородность привносится в эту среду путем приписывания узлам и связям чисел, которые характеризуют размеры соответствующих пор и горловин. Моделирование процесса на выбранной реализации решетки состоит, таким образом, в расчете движения поверхности раздела воды и нефти по мере того, как она продвигается через самые малые из имеющихся пор, заполняя их вытесняющей жидкостью. В модели перколяции с вытеснением и возможностью образования захваченных областей (trapping invasion percolation) учитывается практическая несжимаемость нефти, а именно: области, полностью окруженные водой, считаются «захваченными», и нефть не может быть извлечена из таких областей.

Перколяция с вытеснением – динамический перколяционный процесс, требующий пошагового расчета роста кластера. Кластер вытесняющей жидкости, полученный в результате моделирования процесса, имеет фрактальную структуру. Для оценки различных фрактальных свойств перколяционного кластера (например, фрактальной размерности) необходимо абстрагироваться от конкретной реализации решетки (вместо конкретных размеров пор и горловин, узлам и связям решетки присваиваются случайные числа) и провести большое количество испытаний на различных реализациях решетки с последующим усреднением. Среди методов численного решения задачи о перколяции особое место занимают методы Монте-Карло, получившие широкое распространение в связи с увеличивающимися темпами развития вычислительных мощностей.

2. Модель перколяции с вытеснением

Рассматривается процесс перколяции с вытеснением на решетке размером *2Lx3L.* Вытесняющая жидкость впрыскивается на левой границе решетки, процесс прослеживается до тех пор, пока не наступает протекание, т.е. пока вытесняющая жидкость не достигнет правой границы (рис.1).

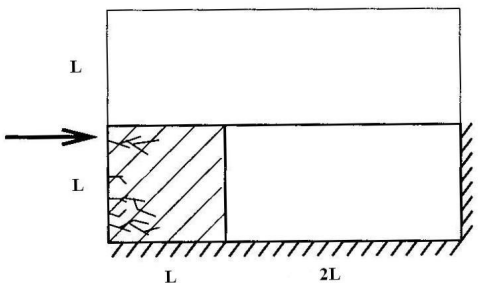


Рис.1 Модель эксперимента

Исследуется влияние непроницаемой нижней границы на поведение растущего кластера. Верхняя и нижняя граница оказывают аналогичное влияние на кластер, поэтому, чтобы исключить влияние верхней границы, вытесняющая жидкость впрыскивается только в нижние *L* ячеек левой границы. Изучаются свойства левого нижнего квадрата размера *LxL*, в котором можно считать, что процесс установился и асимптотические фрактальные особенности перколяционного кластера ярко выражены.

2.1 Последовательный алгоритм

Обратимся к подробному описанию моделирования процесса перколяции с вытеснением. Рассматривается двумерная регулярная прямоугольная решетка. Каждой ячейке решетки ставится в соответствие число r из интервала (0,1). Формируется двумерный массив, значение элемента массива равно нулю, если соответствующая ему ячейка заполнена нефтью. Если в ячейке вода, то значение соответствующего элемента массива равно положительному числу, равному номеру шага, на котором произошло вытеснение нефти водой из соответствующей ячейки. Выберем место впрыскивания вытесняющей жидкости (воды) – *источник* – и место вытекания вытесняемой среды (нефти) – *сток*. Элементам массива, соответствующим местам впрыскивания, присвоим значения, равные единице.

Алгоритм:

1. Найдем *узлы роста* как узлы, занятые вытесняемой жидкостью и соседствующие узлам с вытесняющей средой.
2. Пропустим вытесняющую среду в тот узел роста, в котором случайное число r принимает наименьшее значение. Присвоим этому узлу значение, равное номеру шага.
3. При моделировании перколяции с захватом на каждом шаге необходимо проверять наличие захваченных областей. Области, полностью окруженные вытесняющей жидкостью, помечаются как *«захваченные»* и исключаются из списка узлов роста, для обнаружения таких областей используется алгоритм Хошена-Копельмана.
4. Возвращаемся ко второму шагу алгоритма и повторяем процесс до тех пор, пока вытесняющая жидкость не достигнет стока.

Поясним подробнее пункт 3 алгоритма. Для того, чтобы определить «захваченные» области, необходимо найти на решетке все кластеры нефти, и, исключить кластеры, соединенные с открытой границей. Все оставшиеся кластеры нефти будут со всех сторон окружены водой, и нефть из них вытеснить будет уже никак нельзя. Поиск кластеров нефти проводится по алгоритму Хошена-Копельмана [6].

Важно отметить, что принадлежность узла какому-либо кластеру – глобальное свойство и может быть определена только после просмотра всей решетки, а главное достоинство алгоритма Хошена-Копельмана в том, что он позволяет идентифицировать все кластеры за один проход по решетке. Идея алгоритма состоит в том, что узлам, принадлежащим к одному кластеру, присваиваются метки с одинаковым номером.

2.2 Алгоритм Хошена-Копельмана маркировки кластеров

Последовательно обходим все ячейки решетки по строкам, начиная с первой. Будем называть ячейку свободной, если в ней находится вытесняемая жидкость, и занятой, если в ней находится вытесняющая жидкость. Производится поиск и маркировка свободных ячеек. Необходимо ввести дополнительный массив «правильных» меток, k-й элемент этого массива будет равен k, если k-*я* метка – «правильная», и равен i $(i < k)$, если все ячейки, помеченные номером k, на самом деле принадлежат кластеру с меткой i. Начинаем присваивать ячейкам кластерные метки, начиная с единицы. Для каждой свободной ячейки существует четыре возможных варианта:

1. соседняя ячейка слева занята, соседняя сверху занята;
в качестве рабочей гипотезы принимаем, что данная ячейка входит в новый кластер, присваиваем ячейке текущий номер кластерной метки, увеличиваем номер кластерной метки на единицу; конечно может оказаться, что новый, как нам кажется, кластер является просто ответвлением какого-то другого кластера, мы узнаем об этом, только просмотрев всю решетку

2. соседняя ячейка слева свободна, соседняя сверху занята;
текущая ячейка и ее сосед слева принадлежат к одному кластеру;присваиваем ячейке метку соседа слева

3. соседняя ячейка слева занята, соседняя сверху свободна;
текущая ячейка и ее сосед сверху принадлежат к одному кластеру, однако, у соседа сверху кластерная метка может быть «неправильной», так как в результате проверок могло выясниться, что кластер к которому принадлежит эта ячейка, слился с другим кластером; присваиваем текущей ячейке метку соседа сверху, либо «правильную» метку соседа сверху

4. соседняя ячейка слева свободна, соседняя справа свободна;
если метки соседей справа и слева не совпадают, то считаем большую из них «неправильной» и заносим в соответствующий элемент массива «правильных» меток номер меньшей из меток соседей; присваиваем текущей ячейке метку, наименьшую из «правильных меток» соседних справа слева ячеек

2.3 Фрактальная размерность кластера

На рис. 2 представлена одна из реализаций эксперимента при L=40. Из рисунка видно, что перколяционный кластер заполняет некоторую площадь, но имеет сложную структуру: сильно изрезанный внешний периметр, множество «дырок» во внутренней части.

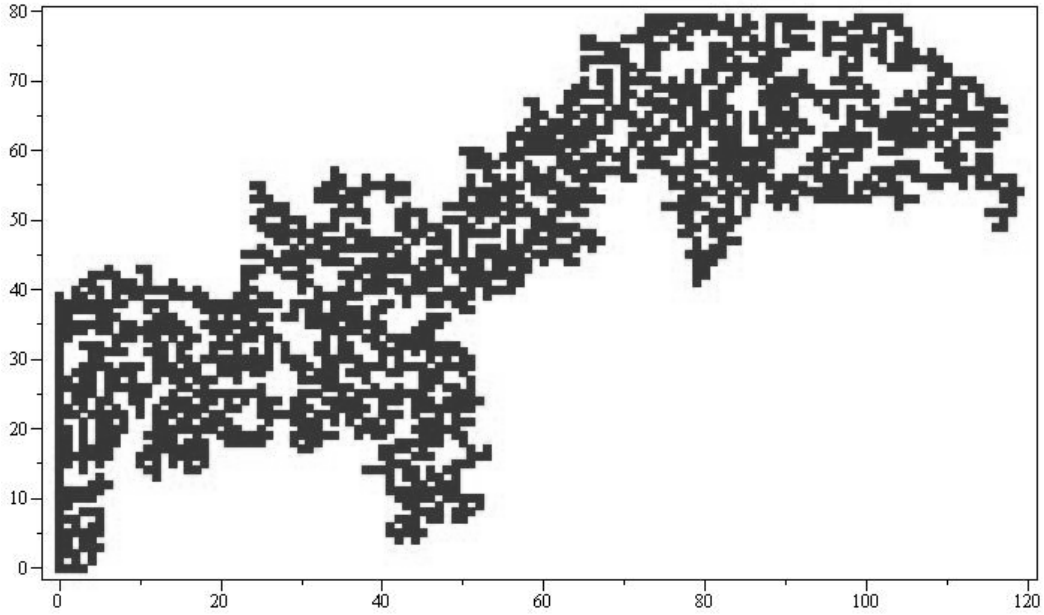


Рис.2 Типичная реализация эксперимента, L=40

Перколяционный кластер обладает фрактальной размерностью. Число занятых узлов решетки в момент наступления протекания растет с размером решетки по закону

$$N(L) = a \cdot L^{d}\ , \qquad (2)$$

где d – фрактальная размерность кластера при перколяции с вытеснением с образованием захваченных областей, путь описания основных свойств кластера. В различных работах, посвященных решению задач о перколяции, приводится следующая оценка значения фрактальной размерности кластера – d=1.82 ± 0.01 [2,3].

Изучая фрактальную структуру кластера, мы ищем легкий путь описать ее основные свойства. Наиболее простой способ – выбрать подмножество меньшей размерности, которое полностью характеризует всю структуру. Для фрактала размерности D, вложенного в двумерную область, – это точки пересечения структуры и прямой. Подмножество пересечения также фрактально, и его размерность $d=D-1$.

В данной работе изучалась фрактальная размерность набора точек, полученных пересечением кластера линией, параллельной нижней границе. Прослеживается изменение фрактальной размерности кластера от границы к средней области.

2.4 Анализ результатов

В ходе численного эксперимента первой задачей было оценить массовую размерность кластера. Для этого велся подсчет ячеек, занятых вытесняющей жидкостью. Причем, статистика собиралась только по левому нижнему квадрату размера *LxL*, т.к. можно считать, что в этой области процесс установившийся.

Однако основной целью работы была проверка гипотезы об уменьшении фрактальной размерности кластера в направлении от средней области решетки к непроницаемой границе [7,8]. Интуитивно понятно, что вероятность роста кластера около границы меньше, так как у граничных ячеек на одного соседа меньше, чем у центральных, то есть около границы меньше *узлов роста*, а раз кластер вероятнее всего будет расти в центре решетки, то размерность его около границы должна быть меньше.

Для повышения достоверности экспериментальные данные осреднялись по статистике. Была проведена серия из 1000 численных экспериментов для значений L =32, 64, 128.

При расчетах было проведено сравнение результатов, полученных с применением стандартного генератора случайных чисел и генератора MT19937-64 на основе алгоритма твистера Мерсенна. Оценки фрактальных размерностей, полученные с помощью генератора MT19937-64, лучше согласуются с теоретическими.

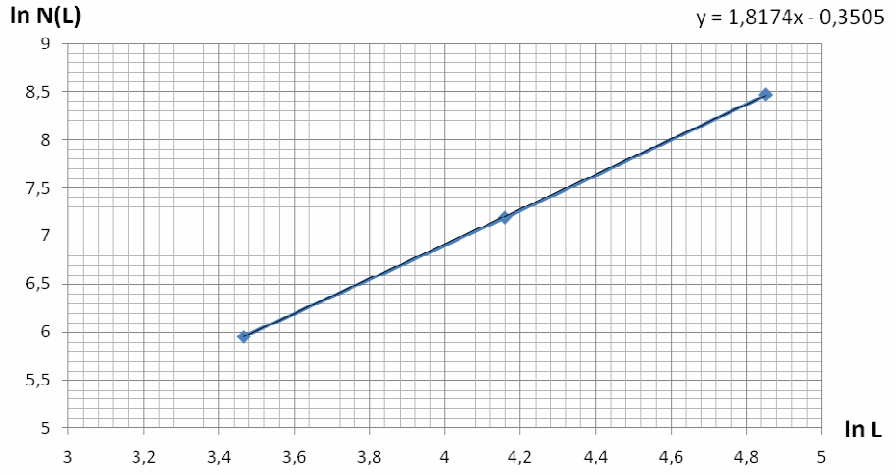


Рис.3 Зависимость числа ячеек перколяционного кластера от размера решетки в двойных логарифмических координатах. Массовая размерность кластера оценивается как угловой коэффициент наклона прямой

Теперь обратимся к анализу численных результатов. Прологарифмировав (2), получим

$$\ln N(L) = d \ln L + \ln a \,, \qquad (3)$$

т.е. фрактальная размерность кластера d – это угловой коэффициент прямой, выражающей зависимость $N(L)$ в двойных логарифмических координатах (см. рис. 3). На рис.3 приведено уравнение прямой, линейно аппроксимирующей искомую зависимость. Была получена оценка фрактальной размерности методом подсчета клеток $d \approx 1.8174$, что вполне согласуется с экспериментально полученным значением $d=1.82$ из различной литературы, посвященной перколяции с вытеснением с образованием захваченных областей.

Одной из важных задач, продиктованных целью работы, было проследить влияние границы на размерность кластера, а именно рост фрактальной размерности пересечения при движении от нижней границы к внутренней области кластера. Это явление было прослежено в ходе эксперимента, результаты наглядно представлены на рис. 4, здесь z изменяется от единицы до L, таким образом, z/L – нормализованное расстояние от границы. Исследовалось поведение величины $\ln N(L)/\ln L$, близкой к значению размерности с точностью до $\ln a/\ln L \to 0$ при $L \to \infty$.

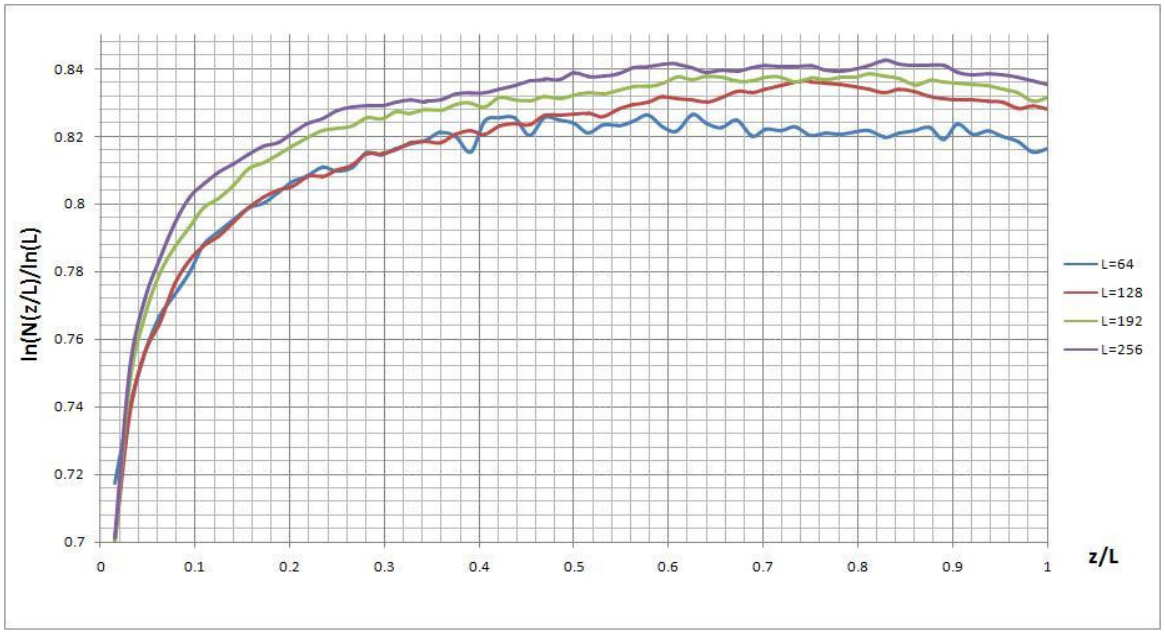


Рис. 4 Изменение фрактальной размерности пересечения в зависимости от расстояния от границы

Важным доказательством адекватности полученных результатов является то, что верхняя граница изменения размерности пересечения согласуется с полученным значением массовой размерности, т.е. фактически экспериментальное значение массовой размерности не зависит от способа измерения.

3 Подходы к распараллеливанию

При моделировании процессов методом Монте-Карло необходимо проведение большого числа статистических испытаний. Естественно, с увеличением числа испытаний, время работы программы линейно растет. Совершенно очевидной была идея произвести распараллеливание нашей программы по испытаниям. Это было сделано средствами OpenMP с минимальными изменениями в коде программы.

Распараллеливание по статистике, как и следовало ожидать, дает очень хорошие результаты по ускорению и эффективности (см. табл. 1). Очевидно, что распараллеливание по статистике дает линейный рост ускорения, что было проверенно на четырехядерной системе и должно выполнятся и для систем с большим количеством ядер.

Хотя для большей достоверности без проведения большого числа испытаний не обойтись, важно отметить, что оценки фрактальных размерностей тем точнее, чем больше размер решетки, так как размерность Хаусдорфа вычисляется в нашем случае по формуле $d = \lim_{L\to\infty} \frac{N(L)}{L}$. Но с увеличением размера решетки, время работы программы растет нелинейно (см. рис. 5), и начиная с некоторых значений L мы столкнулись с тем, что время расчета одного испытания на одном процессоре слишком велико.

Все приведенные ниже результаты были получены на вычислительной платформе со следующими характеристиками: 2 x Intel Xeon 2.00 GHz (4ядра), 4Mb Cache, 4Gb RAM.

Таблица 1

Число ядер	Время работы программы, сек	Ускорение	Эффективность
1	821.78	–	–
2	414.59	1.98	0.99
3	278.99	2.95	0.98
4	211.08	3.89	0.97

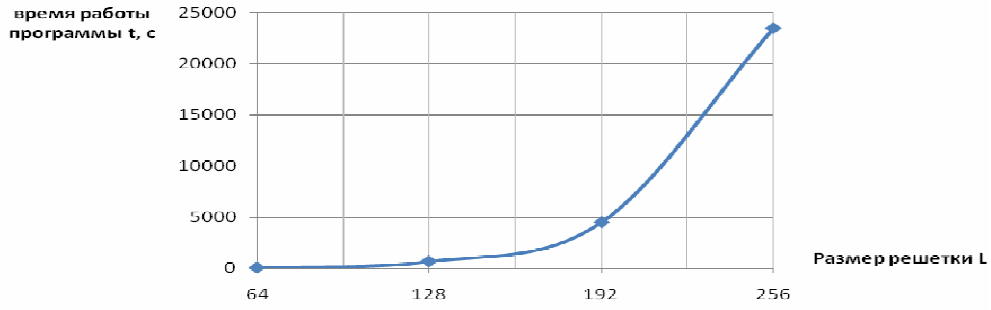


Рис. 5 Динамика роста времени расчетов одного испытания на одноядерном процессоре с увеличением размера решетки

В связи с этим мы стали рассматривать различные варианты пространственного распараллеливания по решетке каждого отдельного испытания. После тщательного анализа работы программы было выяснено, что наибольшее время счета занимает функция поиска «захваченных» областей. Таким образом, проблема распараллеливания отдельного эксперимента была решена использованием на каждом шаге роста кластера не последовательной, а параллельной реализации алгоритма Хошена-Копельмана [9].

3.1 Параллельная версия алгоритма Хошена-Копельмана

Решетка делится на полосы в направлении роста кластера, и каждый процессор исследует только свою полосу. Маркировка кластеров на каждом процессоре выполняется локально и только в конце необходим обмен граничными данными между процессорами.

Кластеры, полностью расположенные внутри одной полосы и не соединяющиеся с границами, естественно называть локальными. Кластеры, расположенные на нескольких полосах и хранящиеся соответственно на нескольких процессорах будем называть глобальными.

В алгоритме Хошена Копельмана каждому кластеру присваивается метка с номером. Различают «правильные» и «неправильные» метки. «Неправильные» метки появляются в процессе подсчета, когда кажется что обнаружен новый кластер, а оказывается что это часть уже известного кластера. Для «неправильных» меток известен номер соответствующей «правильной» метки.

Для параллельной реализации необходимо ввести еще один тип меток – «глобальные». В этих структурах данных должна храниться информация о том, какие локальные кластеры связаны между собой, то есть образуют глобальный кластер.

Таблица 2

Число ядер	Время работы программы, сек	Ускорение	Эффективность
1	5412.73	–	–
2	3689.32	1.47	0.74
3	2700.66	2.00	0.67
4	2183.87	2.49	0.62

В таблице 2 приведены результаты пространственного распараллеливания. Ускорение и эффективность конечно меньше, чем при распараллеливании по статистике, потому что при пространственном распараллеливании необходимо наличие последовательных участков кода. Эффективность падает линейно в связи с линейным ростом количества областей сшивки кластера по закону (2*p*-3), где *p* - число ядер. Получив линейную зависимость для четырехядерной системы, мы можем сделать вывод о линейном характере роста ускорения и для большего количества ядер.

4 Выводы

Оба рассмотренных подхода к распараллеливанию моделирования процесса перколяции с вытеснением доказали на практике свою эффективность и полезность. Анализ ускорения и эффективности показал, что распараллеливание по числу статистических испытаний выгодно осуществлять, когда $N_{MC}/L \geq 10$, где N_{MC} – число экспериментов. Начиная с *L*=256, когда условие $N_{MC}/L \geq 10$ нарушается, распараллеливание только по статистике уже невыгодно, так как расчет одного эксперимента занимает продолжительное время. При моделировании процесса на больших размерах решеток необходимо пространственное распараллеливание. Надо отметить, что возможно успешное комбинирование этих двух подходов при больших размерах решеток и большом количестве статистических испытаний.

Литература

1. Koplik J., Percolation and capillary fluid displacement / J. Koplik, D. Wilkinson, J. Willemsen//Schlumberger-Doll Research preprint, 1983, – 15 p.
2. Sahimi M. Applications of percolation theory./ M. Sahimi. – Taylor&Francis, 1993, – 258p.
3. Stauffer D. Introduction to percolation theory./ D. Stauffer, A. Aharony. – Taylor&Francis, 2003, – 181p.
4. Федер Е. Фракталы: Пер. с англ./ Е. Федер. – М.: Мир, 1991. – 254 с.
5. Шредер М. Фракталы, хаос, степенные законы. Миниатюры из бесконечного рая./ М. Шредер. – Ижевск: НИЦ «Регулярная и хаотическая динамика», 2001, – 528 с.
6. Тарасевич Ю.Ю. Перколяция: теория, приложения, алгоритмы: Учебное пособие./ Ю.Ю. Тарасевич. – М.: Едиториал УРСС, 2002. – 112с., илл.
7. Cafiero R., Surface effects in invasion percolation / R. Cafiero, G. Caldarelli, A. Gabrielli // Phys. Rev. E, 1997, – 4 p.
8. Gabrielli A., Theory of boundary effects in invasion percolation / R. Cafiero, G. Caldarelli, R. Cafiero // arxiv.org, 1998, – 11 p.
9. Tiggemann D., Simulation of percolation on massively-parallel computers/ D. Tiggemann // arxiv.org, 2008,– 9p.

Иерархические методы улучшения масштабируемости и эффективности распределенных расчетов в системе метакомпьютинга X-Com *

С.И. Соболев

Система метакомпьютинга X-Com - инструментарий для организации распределенных неоднородных вычислительных сред и проведения расчетов в таких средах. В статье обсуждаются архитектурные решения, примененные в системе X-Com последнего поколения, направленные на улучшение масштабируемости и повышение эффективности распределенных расчетов.

1. Введение

Применение технологий распределенных вычислений для решения больших вычислительно сложных задач стало привычным фактом. Однако уже сейчас на этом пути встают проблемы, связанные с координацией огромного числа взаимодействующих компьютеров и потоков данных между ними. Технологически несложно объединить для работы над единой задачей, скажем, пять суперкомпьютерных комплексов, находящихся в различных регионах России, однако эффективность такого решения будет под вопросом. Предположим, обработка одной независимой порции задачи занимает в среднем 5 минут, а всего над задачей работают 6 тысяч процессоров распределенной среды. Организуя расчеты по обычной клиент-серверной схеме, получаем, что сервер должен корректно обрабатывать до 40 клиентских запросов в секунду. Такая нагрузка на сервер распределенных вычислений имеет тот же порядок, что и суммарная нагрузка на веб-сайты компании Google, обслуживаемые большими фермами серверов. При этом мы не принимаем в расчет затраты ресурсов сервера на генерацию вычислительных порций и обработку результатов. При использовании шифрования данных нагрузка на сервер увеличится еще на порядок. Кроме того, приведенные в качестве примера 6 тысяч процессоров – это реалии сегодняшнего дня, однако уже сейчас в России строятся суперкомпьютеры с десятками тысяч процессорных ядер. Мы естественным образом приходим к необходимости использования распределенных и иерархических технологий при организации самих распределенных вычислений.

В НИВЦ МГУ имени М.В. Ломоносова разрабатывается система метакомпьютинга X-Com [1, 4] – программный инструментарий, предназначенный для организации распределенных неоднородных вычислительных сред и проведения вычислений в таких средах. Базовыми компонентами системы является сервер задачи и клиент X-Com. Сервер задачи отвечает за разбиение конкретной прикладной задачи на независимые вычислительные порции, распределение их на вычислительные узлы и объединение получаемых результатов. Клиенты X-Com, устанавливаемые на узлах, принимают от сервера данные, запускают вычислительные модули прикладной задачи и отправляют результаты обработки порций обратно на сервер.

В последние два года [6, 7] основная работа над системой была сосредоточена на улучшении ее архитектуры с целью повышения масштабируемости и эффективности проводимых с помощью нее расчетов. Наиболее важные реализованные и планируемые архитектурные решения обсуждаются в настоящей статье.

2. Иерархия "сервер задач – сервисы запросов"

Требование обработки сервером задач крайне интенсивного потока запросов от вычислительных узлов приводит к необходимости распределения его работы на отдельные процессы, способные выполняться на разных физических машинах. Естественным представляется разбиение функциональности сервера задач на процесс-координатор, обладающий полной информа-

* Работа выполняется при поддержке гранта Президента РФ для молодых ученых МК-3040.2009.9.

цией о ходе задачи, и сервисы обработки запросов, взаимодействующие непосредственно с вычислительными узлами. Такая архитектура будет особенно эффективна при включенном шифровании данных, требующем дополнительных затрат процессорного времени. В этом случае шифрование и дешифрование может проводиться на выделенных компьютерах, нагрузка на компьютер с процессом-координатором при этом существенно снижается.

В серверной части X-Com выделяются следующие типы запросов и соответствующих им сервисов (Рис. 1). Сервис TSR отвечает за выдачу первоначальной информации о задаче. Файловый сервис – это фактически файловый сервер, у которого клиент запрашивает файлы прикладной задачи и вспомогательные файлы, если они необходимы (отметим, что в качестве альтернативного файлового сервиса может использоваться любой httpd-сервер). Через сервис REQ клиент запрашивает очередную порцию данных у сервера задачи. Результаты выполнения прикладной задачи над данными очередной порции клиент пересылает сервису ASW. Сервис STAT предназначен для предоставления статистической информации о ходе расчета в форматах HTML и XML.

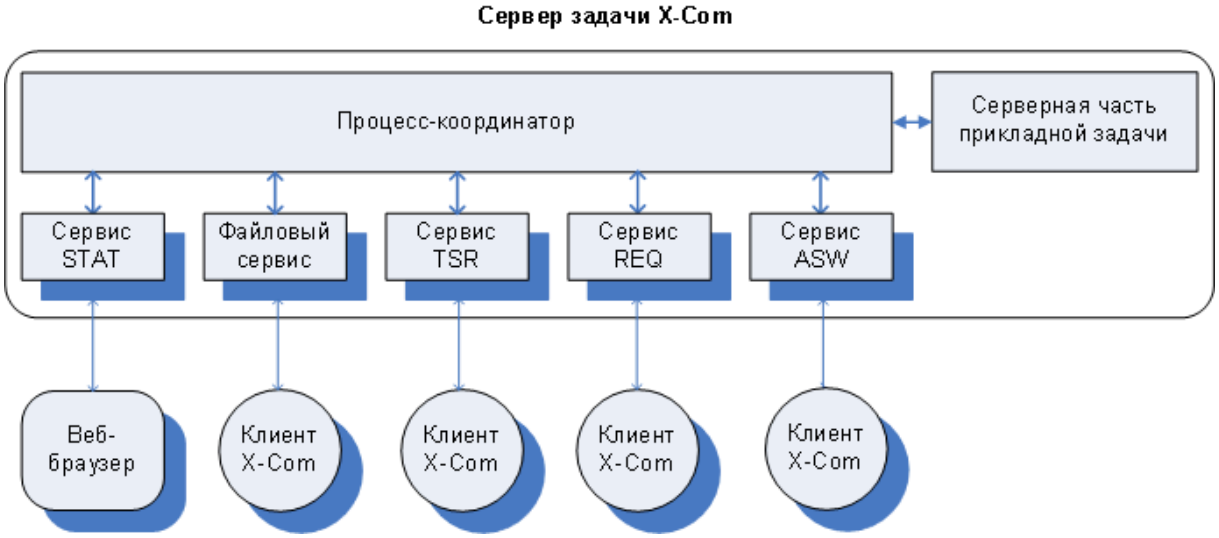


Рис. 1. Структура сервера задачи X-Com

Обмен данными между клиентами X-Com и сервисами запросов производится по специальному протоколу на основе XML поверх стандартного протокола HTTP в сети TCP/IP. Обмен данными между сервисами запросов и процессом-координатором ведется по более простому и экономичному протоколу, причем осуществляться он может как поверх TCP/IP (сервер задач и процессы запросов могут работать на различных компьютерах; поддерживается в любой ОС), так и посредством локальных сокетов UNIX (сервер задач и процессы запросов должны работать на одном компьютере; поддерживается только для ОС семейства UNIX/Linux). Первый способ более универсален, второй обеспечивает более высокую производительность при работе всех серверных процессов на одной физической машине.

Сервер задачи может быть запущен в режиме любого из сервисов вручную (в документации [2] данный режим обозначен как Subserver). При этом в настройках указываются точка доступа (пара хост-порт или имя сокета) процесса-координатора.

3. Иерархия "подсистема управления заданиями – серверы задач"

Работа в масштабной распределенной среде – это, в том числе, огромное число задач и пользователей. Запуск и контроль прохождения заданий в таких средах вручную попросту невозможен – слишком много факторов нужно учесть. Необходимы простые и понятные средства взаимодействия с пользователями, а также механизмы централизованного управления потоками заданий. В системе X-Com эти функции обеспечиваются подсистемой управления заданиям XQSERV. Прототип подсистемы XQSERV был описан в работе [3], настоящая статья описывает актуальное состояние подсистемы.

Подсистема управления заданиями XQSERV состоит из серверной части, отвечающей за распределение задач в вычислительной среде, и клиентской, реализующей пользовательские интерфейсы (Рис. 2). Эти интерфейсы позволяют пользователям работать с вычислительной средой с использованием привычных метафор традиционных высокопроизводительных комплексов – поставить задание или набор заданий в очередь, проконтролировать ход их выполнения, удалить задание из очереди. Базовый метод работы с клиентом подсистемы управления заданиями – вызов клиента из командной строки с необходимыми опциями. Общение между клиентом и сервером XQSERV может осуществляться двумя способами – через сокеты UNIX либо поверх TCP/IP. Первый способ работоспособен только в операционной среде UNIX/Linux и только в том случае, когда клиент и сервер XQSERV запущены на одной физической машине. Этот способ средствами операционной системы обеспечивает получение сервером корректной информации об имени пользователя, вызвавшем клиента. Второй способ обмена данными (TCP/IP) такой информации, вообще говоря, не предоставляет, однако он более универсален и может использоваться при работе клиента на удаленной машине с отличной от серверной операционной системой.

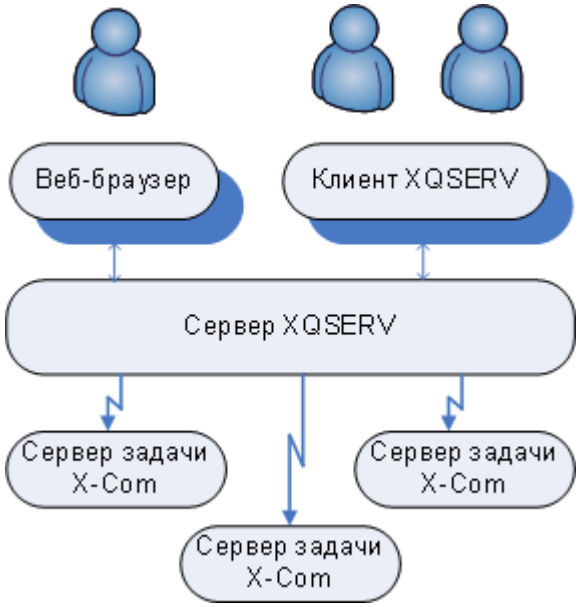


Рис. 2. Подсистема управления заданиями XQSERV

Еще один пользовательский интерфейс, реализуемый подсистемой XQSERV – веб-интерфейс, позволяющий отслеживать общее состояние очереди заданий и ход выполнения каждой задачи из очереди.

Сервер подсистемы управления заданиями XQSERV (управляющий сервер) реализует логику распределения заданий в вычислительной среде. За каждое конкретное задание отвечает свой сервер задачи; с точки зрения управляющего сервера запуск задания означает запуск соответствующего сервера задачи. В простейшем случае управляющий сервер организует линейную очередь, запуская все поступающие от пользователей задания последовательно на всех доступных ресурсах. Такой метод распределения заданий реализует основную идею метакомпьютерных вычислений, а именно использование максимального объема ресурсов для решения задачи, и позволяет достичь максимальной суммарной производительности подключенных ресурсов. Этот метод, однако, не гарантирует максимальной эффективности их использования, в частности, не учитывая требования прикладных задач к тем ресурсам, на которых они будут выполняться. С другой стороны, в ряде случаев может возникнуть необходимость в одновременной работе более одной задачи. Очевидно, необходимы методы динамического разбиения вычислительной среды на сегменты с заданными свойствами, каждый из которых будет выделяться одной задаче, при этом при изменении состава заданий состав сегментов также будет меняться.

4. Иерархическая сегментация среды

Задачи в распределенной среде могут предъявлять различные требования к ресурсам, на которых они должны выполняться. Рассмотрим типичный случай [5]: предположим, что время обработки каждой вычислительной порции достаточно велико, при этом оно существенно зависит от тактовой частоты процессора, а среда объединяет узлы как с высокой, так и низкой частотой CPU. В этом случае вполне возможен вариант, при котором слабые узлы, получив свои порции в самом начале обработки задания, не закончат их обработку до момента завершения всего расчета. Подключение таких узлов для данного расчета окажется нецелесообразным; в то же время, их вполне можно было бы использовать, например, для решения относительно небольших задач либо для тестовых запусков приложений.

При постановке задания в очередь XQSERV можно указать следующие требования к ресурсам: минимальную и максимальную тактовую частоту CPU и/или производительность, минимальный и максимальный объем оперативной памяти, минимальное и максимальное число процессорных ядер, тип операционной системы, процессорную архитектуру. Можно также указать список кластеров, на которых разрешается расчет (принадлежность к кластерам определяется по идентификаторам вычислительных узлов).

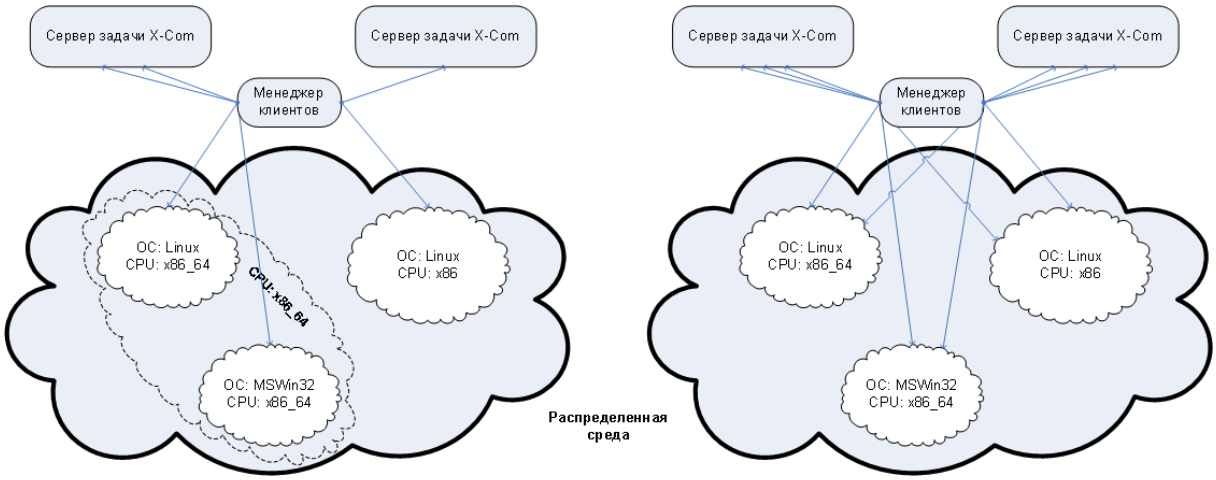


Рис. 3. Примеры сегментации распределенной среды

Чтобы реализовать динамическое перераспределение вычислительных узлов между заданиями, в состав подсистемы управления заданиями XQSERV включен менеджер клиентов. Менеджер клиентов представляет собой процесс-сервер, к которому обращаются при первом запуске все клиенты на вычислительных узлах. Менеджер перенаправляет вычислительные узлы к тем серверам задач, требованиям которых они соответствуют, либо (при отсутствии явно указанных требований) распределяет клиентов поровну на каждую запущенную задачу. Если имеется несколько одновременно работающих задач, и подключающийся узел удовлетворяет требованиям каждой из них, он будет перенаправлен к задаче, запущенной раньше всех. После завершения задачи клиенты на узлах вновь обращаются к менеджеру за новым назначением. Примеры возможной сегментации распределенной среды приведены на Рис. 3.

5. Иерархия промежуточных серверов

Рассмотренные выше механизмы иерархической сегментации среды реализуют логическую группировку вычислительных ресурсов. Однако зачастую ресурсы распределенной среды уже имеют явно выраженную физическую группировку, представляя собой, например, узлы вычислительных кластеров или машины компьютерных классов. Как правило, такие узлы размещены в рамкой закрытой локальной сети, из которой прямой доступ к серверу задач через Интернет может отсутствовать по соображениям безопасности. В этом случае в распределенную среду может быть введен еще один компонент – промежуточный (буферизирующий) сер-

вер. С точки зрения нижележащих узлов промежуточный сервер представляется единственным сервером, доступным данным узлам, с точки же зрения центрального сервера промежуточный сервер сам представляется вычислительным узлом. Помимо организации доступа в закрытую сеть, промежуточный сервер несет еще одну важную функцию – буферизацию обмена данными между "своими" узлами и центральным сервером, снижая тем самым нагрузку на него за счет минимизации числа сетевых соединений (это достигается группировкой нескольких порций в едином запросе) и оптимизации сетевого трафика (клиент на вычислительном узле скачивает необходимые файлы не с центрального, а с ближайшего к нему промежуточного сервера).

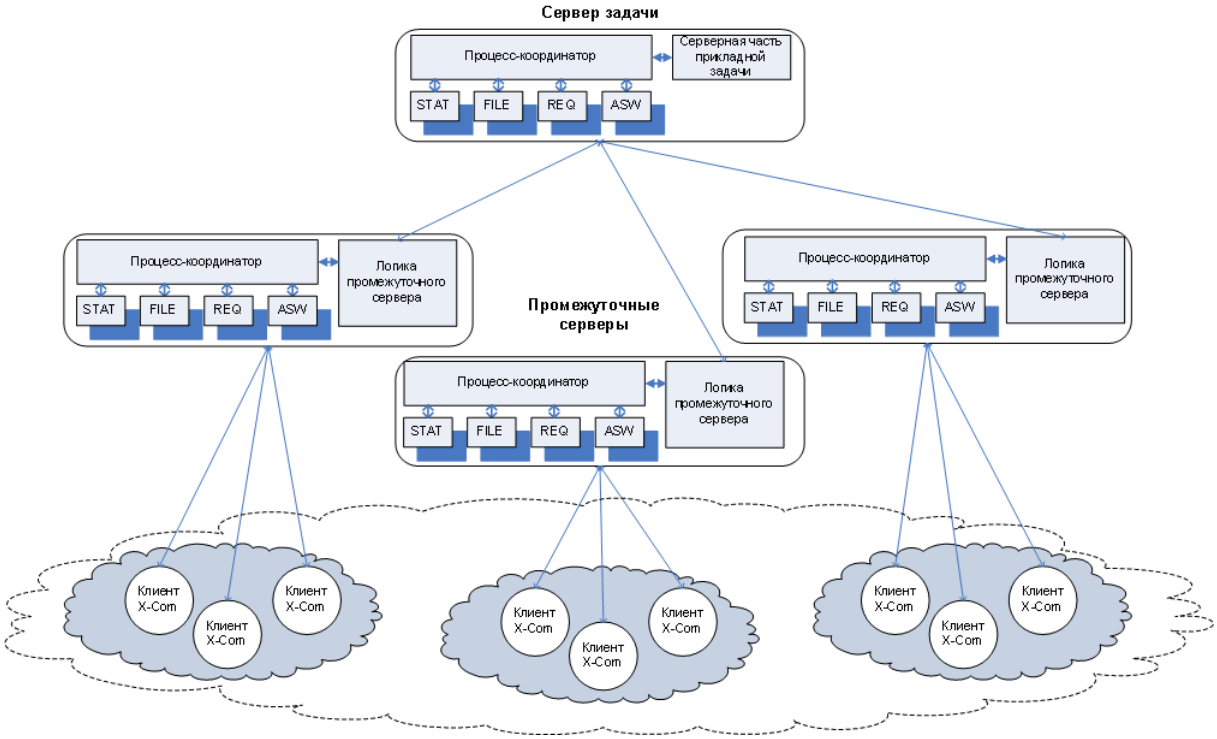


Рис. 4. Организация вычислений с использованием промежуточных серверов

Механизм промежуточных серверов достаточно подробно описан в предыдущих публикациях по системе X-Com, в частности, в [7]. С точки зрения реализации промежуточный сервер – это один из режимов работы сервера задачи (в документации [2] обозначен как Proxy), при котором в качестве "задачи" подключается модуль, реализующий общение с вышестоящим сервером (полностью аналогично клиентской части X-Com). Промежуточные серверы позволяют формировать распределенную среду в виде дерева с произвольным числом ярусов, в корне которого будет центральный сервер X-Com, в узлах – промежуточные серверы, а листьями будут являться клиенты X-Com (Рис. 4).

Хотелось бы отметить, что анализ модели промежуточных серверов и их реализация существенно способствовали внедрению иерархических методов и в другие компоненты системы X-Com.

6. Иерархия "вычислительный клиент – рабочие процессы"

Клиентская часть X-Com (вычислительный клиент), устанавливающаяся на всех узлах распределенной среды и взаимодействующая с сервером задачи, отвечает за запуск вычислительной части прикладной задачи. Архитектура современных вычислительных узлов позволяет запускать сразу несколько вычислительных процессов на одном узле (обычно по числу процессорных ядер). В настоящее время для реализации этой возможности на каждом узле запускается по несколько клиентов X-Com, которые взаимодействуют с сервером задач и запускают вычислительные процессы независимо друг от друга. Логичным представляется переход к версии клиента X-Com, поддерживающей многопроцессорные и многоядерные конфигурации. Такой

клиент будет однократно скачивать все рабочие файлы прикладной задачи на узел (это позволит уменьшить сетевой трафик и нагрузку на сервер задач) и запускать необходимое число параллельных вычислительных процессов (Рис. 5).

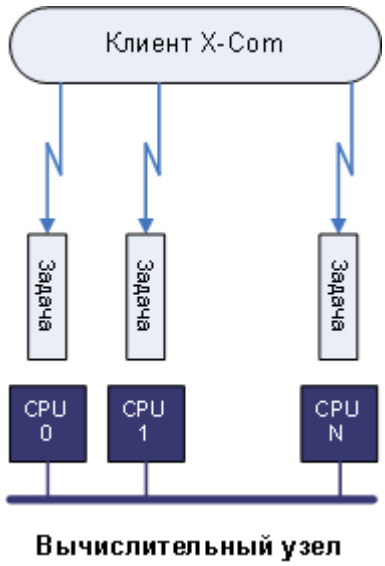


Рис. 5. Перспективная архитектура клиента X-Com

Архитектура такого клиента слегка напоминает архитектуру промежуточного сервера. Автоматически появляется возможность запускать на узлах как чисто последовательные программы, так и программы, написанные с использованием OpenMP. Число запускаемых процессов на узле может указываться при установке клиента, а может быть параметром при запуске каждой задачи.

7. Заключение

Внедрение иерархических и распределенных методов обработки данных на всех уровнях архитектуры системы X-Com способствует снижению накладных расходов системы метакомпьютинга и, как следствие, увеличивает эффективность ее работы. Распределение функциональности сервера задач на независимые процессы, способные работать на отдельных компьютерах, уменьшает загрузку процессора и каналов связи каждой из машин. Подсистема управления заданиями позволяет разбить все множество имеющихся вычислительных ресурсов на подмножества, эффективно решающие имеющиеся прикладные задачи, а кроме того, обеспечивает высокоуровневые пользовательские интерфейсы для работы в распределенной среде. Промежуточные серверы оптимизируют передачу больших объемов данных и снижают загрузку центрального сервера распределенных вычислений. Иерархическая организация клиентской части также позволит экономить сетевой трафик и оптимизировать загрузку вычислительных узлов.

Литература

1. Семейство программ X-Com (официальный сайт) [Электронный ресурс]. -Режим доступа: http://x-com.parallel.ru/ (дата обращения: 5.11.2009).
2. Руководство пользователя системы X-Com^2 [Электронный ресурс]. -Режим доступа: http://x-com.parallel.ru/documentation.html (дата обращения: 5.11.2009).
3. С.И. Соболев. Управление заданиями в Виртуальном метакомпьютерном центре на основе технологий X-Com. Распределенные вычисления и Грид-технологии в науке и образовании. Труды второй международной конференции (Дубна, 26 - 30 июня 2006 г.), Дубна: ОИЯИ, 2007, с. 401-404.

4. Соболев С.И. Использование распределенных компьютерных ресурсов для решения вычислительно сложных задач // Системы управления и информационные технологии. 2007, №1.3 (27). С. 391-395.

5. С.И. Соболев. Эффективная работа в распределенных вычислительных средах // Численные методы, параллельные вычисления и информационные технологии. 2008. 249-258.

6. С.И. Соболев. Архитектура нового поколения системы метакомпьютинга X-Com // Распределенные вычисления и Грид-технологии в науке и образовании. Труды третьей международной конференции. Дубна, 30 июня - 4 июля 2008 г. С. 123-127.

7. Воеводин Вл.В., Жолудев Ю.А., Соболев С.И., Стефанов К.С. Эволюция системы метакомпьютинга X-Com // Вестник Нижегородского государственного университета им.Н.И. Лобачевского. 2009, №4. С. 157-164.

Параллельное построение множества достижимости высокоманевренного летательного аппарата методом «мультифиниша»

Е.М. Воронов, А.П. Карпенко, В.А. Федин

Рассматривается задача построения области достижимости динамической системы. Исследуется подход к решению этой задачи на основе многократного интегрирования модельной системы обыкновенных дифференциальных уравнений при различных управлениях. Предлагается метод балансировки загрузки многопроцессорной вычислительной системы при параллельном решении задачи на основе указанного подхода. Описывается реализация этого метода средствами коммуникационной библиотеки MPI. Приводятся результаты параллельного решения задачи для системы, описывающей динамику высокоманевренного летательного аппарата.

1. Введение

Во многих прикладных областях, использующих модели исследуемых объектов в виде обыкновенных дифференциальных уравнений (ОДУ), возникает задача построения области достижимости соответствующей динамической системы. Например, такая задача возникает при решении проблемы траекторной безопасности летального аппарата [1], а также при решении близкой задачи о посадке вертолета на подвижный носитель [2]. Важной особенностью задачи построения области достижимости является то, что ее часто приходится решать в режиме реального времени.

Аналитическое построение области достижимости удается лишь в простейших случаях, не представляющих практического интереса. Поэтому для построения этой области приходится использовать численные методы.

Дискретную аппроксимацию области достижимости динамической системы можно построить путем многократного интегрирования модельной системы ОДУ при различных управлениях (метод мультифиниша). В практических задачах типичной является ситуация, когда основные вычислительные затраты при построении области достижимости методом мультифиниша обусловлены затратами на интегрирование указанной системы ОДУ. Поэтому ускорить вычисления можно за счет параллельного интегрирования этой системы ОДУ.

К параллельным методам интегрирования ОДУ относятся блочные методы, а также методы на основе геометрической схемы декомпозиции системы ОДУ.

Методы первого класса основаны на использовании одношаговых и многошаговых блочных формул интегрирования ОДУ, например, формулы Адамса-Башфорта [3]. В вычислительной практике обычно используются не более чем 4-х точечные блочные методы. Поэтому потенциальный параллелизм этой группы методов невелик.

Идея методов второго класса состоит в разбиении модельной системы ОДУ на подсистемы, количество которых равно количеству используемых процессоров вычислительной системы, и интегрировании каждой из этих подсистем ОДУ на своем процессоре. В силу, как правило, относительно невысокой размерности вектора фазовых координат исследуемой динамической системы данные методы также принципиально не могут обеспечить значительное ускорение.

В работе полагается, что, используемая многопроцессорная вычислительная система (МВС) представляет собой *однородную* MIMD-систему с общей памятью [4]. Поскольку метод мультифиниша использует интегрирование совокупности систем модельных ОДУ, естественным в данном случае является распараллеливание на основе разделения этой совокупности на наборы, количество которых равно количеству процессоров в используемой МВС. В работе рассматривается проблема балансировки загрузки МВС при распараллеливании метода мультифинаша по указанной схеме, получены оценки эффективности параллельных вычислений. В качестве примера рассмотрена задача построения области достижимости для летального аппарата, описываемого системой ОДУ шестого порядка.

2. Постановка задачи

Рассмотрим динамическую систему

$$\begin{cases} \dot{x}_1 = f_1(t, x_1, ..., x_n, u_1, ..., u_m),\ x_1(0) = x_1^0, \\ ... \\ \dot{x}_n = f_n(t, x_1, ..., x_n, u_1, ..., u_m),\ x_n(0) = x_n^0, \end{cases} \tag{1}$$

где $X = X(t) = (x_1(t), x_2(t), ..., x_n(t))^T$ – n-мерный вектор фазовых переменных системы, $U = U(t) = (u_1(t), u_2(t), ..., u_m(t))^T$ – m-мерный вектор управлений, $X^0 = (x_1^0, x_2^0, ..., x_n^0)^T$ – n-мерный вектор начальных условий, $t \in [0, T]$. На вектор фазовых переменных X и вектор управления U наложены ограничения

$$X \in D_X,\ U \in D_U \subset L_U[0, T], \tag{2}$$

где $L_U[0, T]$ – некоторое пространство m-мерных функций, определенных на интервале $[0,T]$, например, пространство функций, интегрируемых с квадратом на этом интервале.

Для компактности записи используем векторную форму системы (1)

$$\dot{X} = F(t, X, U),\ X(0) = X^0, \tag{3}$$

где $F = F(t, X, U) = (f_1(t, X, U), f_2(t, X, U), ..., f_n(t, X, U))^T$ – n-мерная вектор-функция правой части системы ОДУ (1).

Среди фазовых переменных $x_1, x_2, ..., x_n$ выделим $\nu \le n$ переменных. Не ограничивая общности, положим, что эти переменные образуют ν-мерный вектор $Y = Y(t) = (x_1(t), x_2(t), ..., x_\nu(t))^T$.

Областью достижимости $D_Y = D_Y(T, X^0)$ системы (3) назовем множество всех возможных значений вектора $Y(T)$, которые достигаются на решениях системы (3) при начальных условиях X^0 и ограничениях (2).

Рассмотрим следующую *прямую задачу*: при заданном векторе начальных условий X^0 и конечном времени t_T построить область достижимости D_Y системы (3). Рассмотрим также *обратную задачу*: при заданных векторе начальных условий X^0, конечном времени T и точке $Y(T)$, принадлежащей области достижимости D_Y, найти управление $U \in D_U$, переводящее систему (3) в эту точку.

Для решения прямой задачи, очевидно, достаточно построить границу Γ_Y области достижимости D_Y. В некоторых случаях удается найти множество допустимых управлений $D_U^\Gamma \subseteq D_U$, принадлежащих классу управлений $L_U^\Gamma[0, T] \subseteq L_U[0, T]$, которые приводят систему (3) на эту границу [1]. В таком случае прямая задача сводится к построению границы Γ_Y.

Сделаем следующие допущения. Во всех случаях при интегрировании системы ОДУ (3) используется алгоритм с постоянным шагом интегрирования $\Delta t = \frac{T}{K}$, где $K = K(T)$ – количество шагов интегрирования. Для каждого из шагов интегрирования требуется l вычислений значения функции $F(t, X, U)$, вычислительная сложность которой (количество арифметических операций, необходимых для однократного вычисления значения этой функции) не зависит от аргументов t, X, U и равна C_F. Вычислительная сложность затрат на реализацию алгоритма интегрирования на одном шаге интегрирования равна $C_I = C_I(l)$.

Покроем множество D_U некоторой сеткой с узлами $U_1, U_2, ..., U_M$, где M – общее количество узлов сетки. Поставим в соответствие системе (3) совокупность M систем ОДУ с указанными управлениями:

$$\begin{cases} \dot{X}_1 = F(t, X_1, U_1), X_1(0) = X^0, \\ \quad\quad ... \\ \dot{X}_M = F(t, X_M, U_M), X_M(0) = X^0. \end{cases} \quad (4)$$

Тогда схему приближенного решения прямой задачи методом мультифиниша можно представить в следующем виде:

1) Путем интегрирования совокупности систем ОДУ (4) находим множество точек $\{Y_i(T), i \in [1:M]\}$, представляющее собой дискретную аппроксимацию области D_Y.

2) Запоминаем полученные наборы значений $(U_i, Y_i(T))$, $i \in [1:M]$.

3) Во множестве $\{Y_i(T), i \in [1:M]\}$ находим граничные точки $\{Z_j(T), j \in [1:\varsigma]\}$, представляющие собой дискретную аппроксимацию границы Γ_Y области достижимости.

4) Запоминаем соответствующие наборы значений $(U_j, Z_j(T))$, $j \in [1:\varsigma]$.

На основе полученных точек $\{Z_j(T), j \in [1:\varsigma]\}$ может быть построена непрерывная аппроксимация $\tilde{\Gamma}_Y$ границы Γ_Y.

3. Метод мультифиниша

3.1. Общая схема распараллеливания вычислений

Схема распараллеливания вычислений имеет следующий вид:

1) процессор P_i, $i \in [1:N]$ получает от *host* процессора векторы $U_{(i-1)\mu+1}, U_{(i-1)\mu+2}, ..., U_{i\mu}$;

2) процессор P_i, интегрирует при указанных управлениях систему ОДУ (3) – находит множество точек $\{Y_j(T), j \in [(i-1)\mu+1, i\mu]\}$;

3) процессор P_i передает координаты этих точек *host* -процессору.

Здесь N – количество процессоров в используемой МВС; $\mu = \left\lceil \frac{M}{N} \right\rceil$, где $\lceil v \rceil$ – символ ближайшего целого большего *v*.

При построении границы Γ_Y области достижимости D_Y возникает проблема балансировки загрузки МВС. Рассмотрим для примера случай, когда класс функций $L_U^\Gamma[0,T]$ представляет собой класс релейных функций с не более чем одной точкой переключения. Положим, что одна из границ множества достижимости формируется управлениями этого класса, в которых все компоненты вектора управления, кроме одного, постоянны, а один из компонентов имеет одну точку переключения. Пусть, например,

$$u_1^\Gamma(t, t^S) = \begin{cases} +1, & t \in [0, t^S], \\ -1, & t \in (t^S, T], \end{cases} \quad u_2^\Gamma(t) = u_3^\Gamma(t) = ... = u_m^\Gamma(t) = const,$$

где $t^S \in [0,T]$ – момент времени, когда происходит переключение управления $u_1^\Gamma(t)$ [1]. Далее рассмотрим два способа распределения точек переключения по процессорам для подобного управления.

3.2. Равномерная декомпозиция точек переключения

Покроем интервал $[0,T]$ равномерной сеткой с шагом $\Delta t^S = \frac{T}{M} \le \Delta t$ и узлами $t_i^S, i \in [1:M]$. Положим, что шаг Δt^S кратен шагу Δt, так что $\frac{\Delta t^S}{\Delta t} = \frac{K}{M} = r$, где $r \ge 1$ – целое число. При этом множество управлений $U_1^\Gamma, U_2^\Gamma, ..., U_M^\Gamma$ естественно сформировать в виде $U_i^\Gamma = \left(u_1^\Gamma(t, t_i^S),\ u_2^\Gamma,\ u_3^\Gamma,\ ...,\ u_m^\Gamma\right)^T$, $i \in [1:M]$. Таким образом, в данном случае схема распараллеливания имеет вид, представленный на Рис. 1.

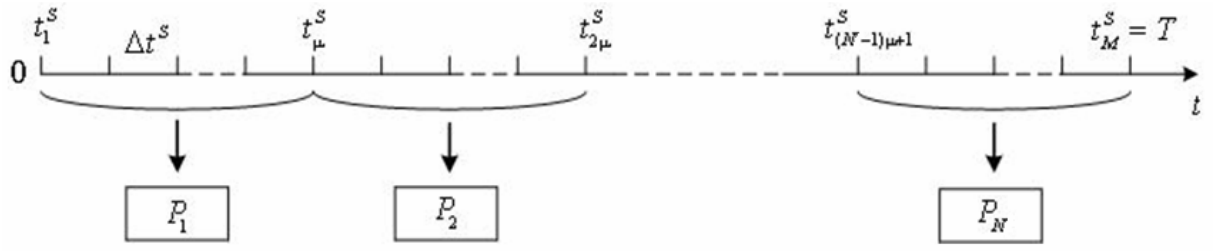


Рис. 1. Схема распараллеливания при равномерной декомпозиции точек переключения

В соответствии с этой схемой на процессоре P_1 выполняется интегрирование системы ОДУ (3) при управлениях $U_1^\Gamma, ..., U_\mu^\Gamma$, на процессоре P_2 – при управлениях $U_{\mu+1}^\Gamma, ..., U_{2\mu}^\Gamma$ и т.д. до процессора P_N, который выполняет интегрирование при управлениях $U_{(N-1)\mu+1}^\Gamma, ..., U_M^\Gamma$.

Вычисления на процессоре P_i организуем следующим образом.

Шаг 0 (этап «разгона»). Исходя из начальных условий X^0, выполняем интегрирование системы (3) при управлении $(1, u_2^\Gamma, u_3^\Gamma, ..., u_m^\Gamma)$ от момента времени 0 до момента времени $t_{i\mu-1}^S$ и запоминает значения компонентов векторов $X(t_{(i-1)\mu+1}^S)$, $X(t_{(i-1)\mu+2}^S)$, …, $X(t_{i\mu-1}^S)$.

Шаг 1. Исходя из начальных условий $X(t_{(i-1)\mu+1}^S)$, выполняем интегрирование системы (3) при управлении $(-1, u_2^\Gamma, u_3^\Gamma, ..., u_m^\Gamma)$ от момента времени $t_{(i-1)\mu+1}^S$ до момента времени T.

Шаг 2. Исходя из начальных условий $X(t_{(i-1)\mu+2}^S)$, выполняем интегрирование системы (3) при управлении $(-1, u_2^\Gamma, u_3^\Gamma, ..., u_m^\Gamma)$ от момента времени $t_{(i-1)\mu+2}^S$ до момента времени T.

…

Шаг μ. Исходя из начальных условий $X(t_{i\mu-1}^S)$, выполняем интегрирование системы (3) при управлении $(-1, u_2^\Gamma, u_3^\Gamma, ..., u_m^\Gamma)$ от момента времени $t_{i\mu-1}^S$ до момента времени T.

В рассмотренной схеме каждый из последующих процессоров повторяет шаг 0 своего предыдущего процессора и только затем выполняет интегрирование системы (3) на «своем» интервале $[t_{(i-1)\mu}^S, t_{i\mu-1}^S]$. Если вычислительные затраты на интегрирование системы (3) велики, то может оказаться целесообразной более тонкая организация вычислений на этапе разгона – однократное интегрирование системы (3) на интервале $[0, T]$ при управлении $(1, u_2^\Gamma, u_3^\Gamma, ..., u_m^\Gamma)$ на одном процессоре и передача значений необходимых компонентов векторов $X(t_1^S)$, $X(t_2^S), \ldots, X(t_M^S)$ остальным процессорам МВС. Однако, если $\mu >> 1$, что можно считать типичной ситуацией, выигрыш от такой организации вычислений не может быть существенным.

Оценим ускорение рассмотренной схемы распараллеливания. В сделанных допущениях время выполнения процессором P_i этапа «разгона» равно

$$T_{i,0} = t_c r(i\mu - 1)(lC_F + C_I); \tag{5}$$

время выполнения *j*-го шага, где $j \in [1:\mu]$, можно оценить величиной

$$T_{i,j} = t_{cal} r(M - (i-1)\mu - j)(lC_F + C_I). \tag{6}$$

Таким образом, общее время решения задачи процессором P_i равно

$$T_i = T_{i,0} + \sum_{j=1}^{\mu} T_{i,j}. \tag{7}$$

Здесь и далее t_{cal} – время выполнения арифметической операции на одном процессоре используемой МВС.

Поскольку загрузка первого процессора P_1 в данном случае, очевидно, максимальна, время параллельного решения задачи определяется выражением

$$\tau_{NN} = T_1 = \sum_{j=1}^{\mu} t_{cal} r(M - j)(lC_F + C_I) \approx t_{cal} r(lC_F + C_I)\frac{M^2}{N},$$

где учтено, что $M >> \mu$. Аналогично, время последовательного решения задачи на одном процессоре оценим величиной

$$\tau_1 = t_{cal} r(lC_F + C_I)\sum_{i=1}^{M}(M - (i-1)) \approx \frac{1}{2} t_{cal} r(lC_F + C_I)M^2.$$

Отсюда имеем

Утверждение 1. Ускорение S параллельного метода на основе равномерной декомпозиции точек переключения управления $u_1^{\Gamma}(t)$ оценивается величиной $S \approx \frac{N}{2}$.

Из утверждения 1 следует, что равномерная декомпозиция точек переключения управления $u_1^{\Gamma}(t)$ не может обеспечить ускорение, превышающее 50% от максимального потенциально возможного ускорения, равного N.

Лучшей балансировки загрузки многопроцессорной вычислительной системы можно добиться за счет неравномерной декомпозиции точек переключения управления $u_1^{\Gamma}(t)$.

3.3. Неравномерная декомпозиция точек переключения

Положим, что процессор P_i выполняется интегрирование системы ОДУ (3) при управлениях $U_a^{\Gamma}, ..., U_b^{\Gamma}$, а процессор P_{i+1} – при управлениях $U_{b+1}^{\Gamma}, ..., U_c^{\Gamma}$, $0 \le a < b < c \le M$. Потребуем, чтобы времена решения задачи процессорами P_i, P_{i+1} были равны, т.е. чтобы выполнялось равенство $T_i = T_{i+1}$, $i \in [1 : N-1]$. Из выражений (5) – (7) следует, что для этого необходимо выполнение равенства $b + \sum_{j=a}^{b}(M - j) = c + \sum_{j=b}^{c}(M - j)$ или, что то же самое, равенства

$$b + \frac{2M - (b+a)}{2}(b-a) = c + \frac{2M - (c+b)}{2}(c-b). \qquad (8)$$

Отсюда вытекает квадратное уравнение для определения величины c при известных значениях величин a, b:

$$c^2 - 2(M+1)c + \left(a^2 - 2b^2 - 2Ma + 2(2M+1)b\right) = 0. \qquad (9)$$

Если для процессора P_1 задаться величинами $a = a_1 = 0$, $b = b_1$, то из выражения (9) легко найти последовательно управления, которые должны обрабатывать процессоры $P_2, P_3, ...$

Рассмотрим для примера случай $M = 1000$, $b_1 = 100$. Из (9) вытекает, что в данном случае можно обеспечить равномерную загрузку 6 процессоров (таблица 1). Отметим, что здесь количество точек переключения Δt_6^S вычислено по остаточному принципу и не удовлетворяет уравнению (9).

Оценим ускорение рассматриваемого метода. Положим, что в последовательном режиме задача решается по схеме, аналогичной рассмотренной выше (с этапом «разгона»). Тогда время последовательного решения задачи можно оценить величиной

$$\tau_1 = t_{cal} r(lC_F + C_I)\left(M - 1 + \sum_{i=1}^{M}(M - i)\right) = t_{cal} r(lC_F + C_I)(M-1)(1 + 0.5M).$$

Поскольку в данном случае вычислительная загрузка всех используемых процессоров (исключая, быть может, последний процессор) примерно одинакова, оценка времени параллельного решения задачи следует из выражения (8) и равна

$$\tau_{NN} = T_1 = t_{cal} r(lC_F + C_I)\left((M+1)b_1 - 0.5b_1^2\right).$$

Таблица 1. Пример неравномерной декомпозиции точек переключения

Процессор P_i	t_a^S	t_b^S	Δt_i^S
P_1	0	100	100
P_2	101	212	111
P_3	213	342	129
P_4	343	504	151
P_5	505	756	251
P_6	757	1000	243

Утверждение 2. Ускорение S параллельного метода на основе неравномерной декомпозиции точек переключения управления $u_1^{\Gamma}(t)$ оценивается величиной

$$S = \frac{(M-1)(1+0.5M)}{(M+1)b_1 - 0.5b_1^2} \approx \frac{M^2}{2Mb_1 - b_1^2}.$$

Из утверждения 2 следует, что для рассмотренного выше примера ускорение равно

$$S \approx \frac{1000^2}{2\times 1000\times 100 - 100^2} \approx 5.26.$$

Заметное отличие ускорения от максимально возможного (равного 6) объясняется тем, что загрузка последнего процессора P_6 далека от оптимальной загрузки.

Рассмотрим другой пример. Пусть $\Delta t_1^S = 88$. Тогда по формуле (9) получим $\Delta t_2^S = 98$, $\Delta t_3^S = 109$, $\Delta t_4^S = 129$, $\Delta t_5^S = 168$, $\Delta t_6^S = 405$. Поскольку при этом все шесть процессоров загружены практически равномерно, ускорение равно

$$S \approx \frac{1000^2}{2\times 1000\times 88 - 88^2} \approx 5.93.$$

Рекуррентное уравнение (8) не удается решить явным способом. Поэтому количество точек переключения $b_1 = \Delta t_1^S$, обрабатываемых процессором P_1, приходится подбирать таким образом, чтобы вычислительная загрузка всех процессоров, включая последний процессор, была примерно одинакова. При этом критерий окончания перебора удобно строить, исходя из того, что при несбалансированности загрузки процессора P_N корни уравнения (9) оказываются комплексными.

4. Пример

Рассмотрим летальный аппарат, уравнения движения центра масс которого в нормальной земной системе координат $0xyz$ описываются системой нелинейных дифференциальных уравнений (10), в которой v – скорость летательного аппарата, θ – угол наклона траектории, ψ – угол поворота траектории, y – высота летательного аппарата, n_t – тангенциальная перегрузка, n_n – нормальная перегрузка, γ_c – скоростной угол крена, g – ускорение свободного падения.

Управлениями летательного аппарата являются тангенциальная перегрузка, нормальная перегрузка и скоростной угол крена, так что $U = (n_t, n_n, \gamma_c)^T$. На управления наложены ограничения

$$n_t^{\min} \le n_t \le n_t^{\max},\ n_t^{\min} = -1.6,\ n_t^{\max} = 0.6;\ |n_n| \le n_n^{\max},\ n_n^{\max} = 8;\ \gamma_c \le |\pi|.$$

$$\begin{cases} \dot{v} = g \cdot (n_t - \sin\theta), \\ \dot{\theta} = g/v \cdot (n_n \cdot \cos\gamma_c - \cos\theta), \\ \dot{\psi} = -g \cdot n_n \cdot \sin\gamma_c / v \cdot \cos\theta, \\ \dot{x} = v \cdot \cos\theta \cdot \cos\psi, \\ \dot{y} = v \cdot \sin\theta, \\ \dot{z} = -v \cdot \cos\theta \cdot \sin\psi. \end{cases} \quad (10)$$

В работе [1] показано, что дальняя, ближняя и боковая границы области достижимости системы (10) формируются управлениями, принадлежащими классу кусочно-постоянных управлений. Ограничимся рассмотрением дальней границы области достижимости. Структура управлений, формирующих эту границу, представлена на Рис. 2.

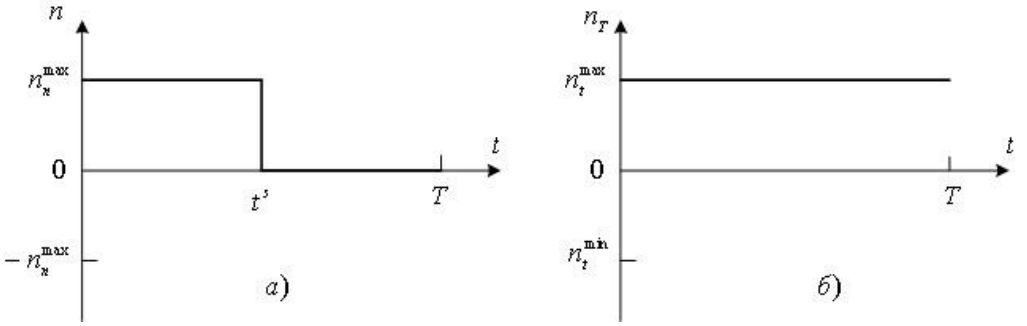


Рис. 2. Структура управлений, формирующих дальнюю границу области достижимости

Для экспериментального исследования эффективности метода мультифиниша разработана MPI-программа [4], реализующая этот метод. Эксперименты выполнены на виртуальном кластере, созданном с помощью программной системы VMware и функционирующим под управлением свободно распространяемой операционной системы Ubuntu. Результаты экспериментов иллюстрирует рис. 4, который показывает, что на числе процессоров от 2 до 10 достигается ускорение, близкое к расчетному.

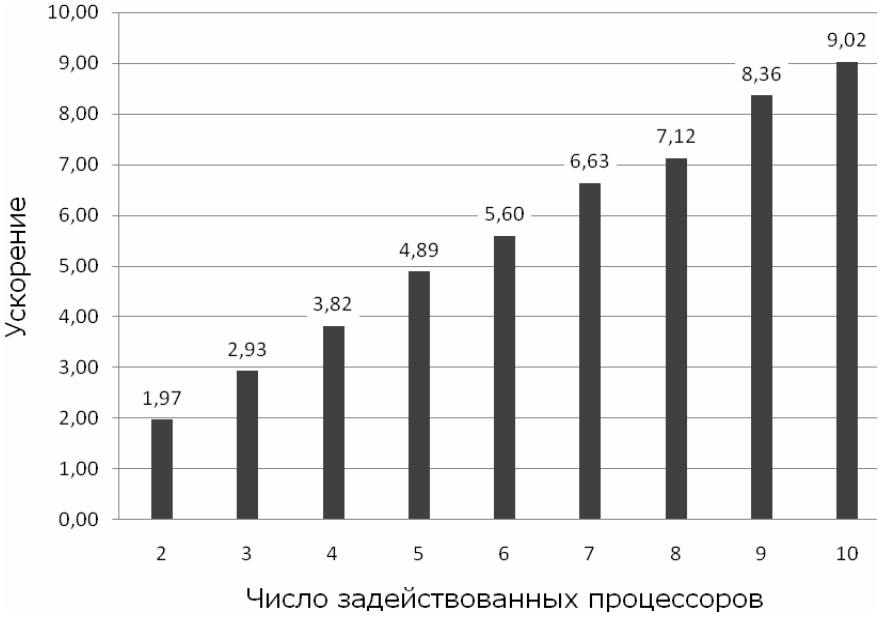


Рис. 4. Экспериментальные результаты

5. Заключение

В работе рассмотрен метод мультифиниша для построения области достижимости динамической системы. Предложен соответствующий параллельный алгоритм, ориентированный на вычислительные системы класса MIMD. Рассмотрены два метода балансировки загрузки этих вычислительных систем. Эффективность метода исследована аналитически, а также экспериментально на примере построения границ области достижимости высокоманевренного летательного аппарата. Исследование показало перспективность практического использования метода мультифиниша.

Авторы благодарят К.О. Вишневецкого за помощь в проведении вычислительных экспериментов.

Литература

1. Воронов Е.М., Карпунин А.А. Алгоритм оценки границ области достижимости летательного аппарата с учетом тяги // Вестник МГТУ. Сер. Приборостроение.- 2007.- №4(69).- С. 81-99.
2. Гурман В.И., Квоков В.И., Ухин М.Ю. Приближенные методы оптимизации управления летальным аппаратом // Автоматика и телемеханика.- 2008.- №4.- С. 191–201.
3. Claus Bendtsen. Parallel Numerical Algorithms for the Solution of Systems of Ordinary Differential Equations // PhD Dissertation, Institute of Mathematical Modeling Technical University of Denmark, June 1996.
4. Воеводин В.В., Воеводин Вл.,В. Параллельные вычисления.– СПб.: БХВ-Петербург, 2004.– 608 с.

Применение параллельных технологий к моделированию глобальной сейсмичности*

В.Л. Розенберг, Л.А. Мельникова

Рассматривается задача моделирования динамических процессов, происходящих в литосфере. Приводится обзор разработанных авторами модификаций сферической блоковой модели, при этом основное внимание уделяется модификации, учитывающей неоднородность земной коры. Обсуждаются некоторые аспекты подхода к программной реализации численных алгоритмов с применением параллельных технологий. Анализируется так называемая масштабируемость параллельного алгоритма с целью его оптимизации. Описываются результаты вычислительных экспериментов с аппроксимацией глобальной системы тектонических плит.

1. Введение

Задача изучения землетрясений на основе статистического и феноменологического анализа существующих каталогов таких событий достаточно сложна, поскольку надежные данные наблюдений покрывают относительно небольшой временной интервал (сто лет и меньше) по сравнению с продолжительностью геотектонических процессов, влияющих на сейсмическую активность. Поэтому моделирование сейсмичности, т.е. пространственно-временных последовательностей землетрясений в различных регионах, а также процессов, происходящих в земной коре, играет важную роль в исследовании характера реального сейсмического потока, выявлении или подтверждении закономерностей, предшествующих сильным толчкам (так называемых «предвестников») и, следовательно, является ключевым аспектом изучения сейсмического риска [1, 2]. Основным результатом моделирования сейсмичности литосферы является искусственный каталог землетрясений, в котором каждое событие характеризуется моментом времени, координатами эпицентра, глубиной, магнитудой и, в некоторых моделях, учитывающих геологическое строение региона, интенсивностью. Моделирование динамики земной коры предполагает получение поля скоростей движения точек на разных глубинах, действующих сил, обусловленных ими смещений, а также характера взаимодействия структурных элементов.

Отметим важные свойства литосферы, как сложной диссипативной системы, которые должны быть учтены при моделировании. Прежде всего, это наличие взаимодействия процессов различной природы и двух временных шкал. Здесь имеется в виду тот факт, что два главных механизма, включенных в сейсмотектонический процесс, именно, тектоническое нагружение с характеристической скоростью в несколько см/год и перераспределение упругого напряжения с характеристической скоростью в несколько км/сек должны рассматриваться в стандартной временной шкале как, соответственно, равномерное движение и мгновенный сброс напряжения. Далее, при моделировании следует принимать во внимание иерархическую блоковую или, возможно, «фрактальную» структуру литосферы, а также универсальность некоторых ее свойств, проявляющуюся как на региональном, так и на глобальном уровнях. Таким образом, идеальная модель призвана воспроизводить как тектонические движущие силы и движения земной коры, так и аккумуляцию напряжения и его сброс в форме землетрясения.

Существует множество различных подходов к моделированию процессов, происходящих в литосфере (см., например, работу [1] и библиографию к ней); тем не менее, среди них их можно выделить два основных направления. Первое, традиционное, направление опирается на детальное исследование одного специфического тектонического разлома или, нередко, одного конкретного сильного землетрясения с целью воспроизведения определенных пре- и/или постсейсмических явлений (характерных для данного разлома или события). Модели второго на-

* Работа выполнена в рамках Программы научно-исследовательских работ Президиума РАН «Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация», для первого автора также при финансовой поддержке РФФИ (грант 09-01-00378) и Уралосибирского междисциплинарного проекта.

правления, предложенные относительно недавно, трактуют сейсмотектонический процесс гораздо более абстрактно; основной задачей моделирования является получение универсальных свойств сейсмичности, обнаруженных эмпирическим путем (прежде всего, степенного закона распределения «размера» событий, именно, закона повторяемости Гутенберга–Рихтера, кластеризации, миграции событий, сейсмического цикла и т. д.). Представляется, однако, что адекватная модель, разрабатываемая в рамках второго направления, должна не только отражать некоторые общие свойства нелинейных систем, но и учитывать геометрию взаимодействующих тектонических разломов. Блоковые модели динамики и сейсмичности литосферы [2, 3] разрабатывались с учетом обоих требований. Подход к моделированию опирается на представление тектонических плит в виде системы абсолютно жестких блоков, находящейся в состоянии квазистатического равновесия; при этом модельное событие представляет собой резкий сброс напряжений, возникающих на разломах, разделяющих блоки, под действием внешних сил. Плоская блоковая модель [2, 3], в которой структура ограничена двумя горизонтальными плоскостями, является наиболее изученной; на ее основе построены аппроксимации реальных сейсмических регионов. Однако, при попытке моделирования динамики глобальных тектонических плит обнаруживаются существенные неточности, для преодоления которых введена сферическая геометрия. Настоящая работа, продолжающая исследования [4–6], фактически представляет собой обзор разработанных модификаций сферической блоковой модели и обсуждение некоторых результатов вычислительных экспериментов.

2. Различные модификации сферической блоковой модели

В сферической модели динамики и сейсмичности литосферы блоковая структура является ограниченной и односвязной частью шарового слоя глубиной H, заключенного между двумя концентрическими сферами, одна из которых (внешняя) интерпретируется как поверхность Земли, другая (внутренняя) — как нижняя граница упругой литосферы. Разделение структуры на блоки определяется пересекающими этот слой бесконечно тонкими разломами, каждый из которых представляет собой коническую поверхность, наклоненную под определенным углом к внешней сфере. Общие точки двух разломов на внешней и внутренней сферах называются вершинами. Участки разломов, ограниченные соответствующими парами соседних вершин, называются сегментами. Пересечения блока с ограничивающими сферами представляют собой сферические многоугольники, при этом пересечение с нижней (для блока) сферой называется подошвой. Предполагается, что вне блоковой структуры могут находиться граничные блоки, примыкающие к внешним сегментам. Другая возможность состоит в рассмотрении блоковой структуры, замкнутой на сфере. Блоки считаются абсолютно жесткими, все их смещения — бесконечно малыми по сравнению с линейными размерами, поэтому геометрия блоковой структуры не меняется в процессе моделирования, и структура не движется как единое целое. Гравитационными силами можно пренебречь, так как они слабо зависят от смещений блоков и блоковая структура в начальный момент времени находится в состоянии квазистатического равновесия. Блоки (в том числе и граничные) имеют шесть степеней свободы. Смещение каждого блока состоит из поступательной и вращательной компонент. Предполагается, что законы движения граничных блоков и подстилающей среды известны, при этом движение описывается как вращение на сфере, т.е. задаются положение оси вращения и угловая скорость.

Разработано несколько модификаций модели, зависящих от способа трактовки глубины сферического слоя. В первой модификации модели (без глубины), считалось, что все характеристики точек структуры определяются только их координатами и не зависят от глубины сферического слоя, поскольку данная глубина значительно меньше линейных размеров блоков. Главное преимущество модификации состоит в значительной экономии времени счета при моделировании, что может быть существенно при большом количестве запусков в эксперименте по вариации того или иного параметра; основной недостаток — формальный учет углов наклона разломов, фактически определяющих характер сейсмичности. Появившаяся позже модификация с постоянной глубиной использовала предположение об однородности литосферы по глубине (все блоки имели одну и ту же глубину H, а свойства всех частей блока (разлома) были одинаковыми). Модификация, разрабатываемая в настоящее время, предусматривает возмож-

ность задания различных глубин (в пределах H) для разных блоков и учета зависимости вязкоупругих свойств разлома от его глубины. Отметим, что по существу это является первой попыткой учета неоднородности литосферы (например, различий в строении континентальной и океанической коры и уменьшения вязкости коры с глубиной) в блоковых моделях.

Поскольку блоки являются абсолютно жесткими, все деформации имеют место на разломах и подошвах блоков; силы возникают на подошвах из-за смещения блоков относительно подстилающей среды и на поверхностях ограничивающих их разломов из-за смещений соседних блоков или их подстилающей среды. Приведем формулы для определения упругой силы (f_t, f_l, f_n), действующей на единицу площади разлома:

$$f_t = K_t(\Delta_t - \delta_t), \quad f_l = K_l(\Delta_l - \delta_l), \quad f_n = K_n(\Delta_n - \delta_n). \tag{1}$$

Здесь (t,l,n) — система координат, связанная с точкой приложения силы (оси t, l лежат в плоскости, касательной к поверхности разлома, ось n ей перпендикулярна); Δ_t, Δ_l, Δ_n — компоненты относительного смещения в системе (t, l, n) (а) соседних блоков в случае, если точка принадлежит части разлома, разделяющей блоки, и (б) блока и подстилающей среды соседнего блока в случае, если точка принадлежит части разлома, отделяющей блок от подстилающей среды соседнего блока; δ_t, δ_l, δ_n — соответствующие неупругие смещения, зависимость от времени которых описывается уравнениями

$$\frac{d\delta_t}{dt} = W_t f_t, \quad \frac{d\delta_l}{dt} = W_l f_l, \quad \frac{d\delta_n}{dt} = W_n f_n. \tag{2}$$

Коэффициенты K_t, K_l, K_n (1), характеризующие упругие свойства разлома, и коэффициенты W_t, W_l, W_n (2), характеризующие вязкие свойства разлома, могут быть различными для разных разломов и, кроме того, могут изменяться в зависимости от глубины.

Аналогично выглядят формулы для вычисления сил и неупругих смещений на подошвах блоков. Смещения любого внутреннего блока и углы его поворотов находятся из условия равенства нулю суммы всех сил, действующих на блок, и суммарного момента этих сил. Это условие обеспечивает состояние квазистатического равновесия системы и одновременно является условием минимума энергии. Поскольку в рассматриваемой модели зависимость сил от смещений и поворотов блоков является линейной, то система уравнений для определения этих величин также линейна и имеет вид

$$Aw = b. \tag{3}$$

Компонентами неизвестного вектора $w = (w_1, w_2, \ldots, w_{6n})$ являются смещения и углы поворота внутренних блоков (n — число таких блоков). Элементы матрицы A (размерности $6n \times 6n$) не зависят от времени и могут быть вычислены один раз в начале процесса. Для подсчета различных криволинейных интегралов выполняется дискретизация (разбиение на ячейки) сферической поверхности подошв блоков и сегментов разломов, при этом предполагается, что значения сил и неупругих смещений совпадают для всех точек ячейки. Система (3) решается в дискретные моменты времени t_i.

При вычислении компонент силы, действующей на разломе, определяется отношение напряжения к давлению:

$$\kappa = \frac{\sqrt{f_t^2 + f_l^2}}{P - f_n}. \tag{4}$$

Здесь P — параметр, который может интерпретироваться как разность между литостатическим и гидростатическим давлением. Взаимодействие между блоками (между блоком и сосед-

ней подстилающей средой) полагается вязкоупругим (нормальное состояние) до тех пор, пока величина κ (4) на части разлома, разделяющего элементы структуры, не превышает значение заданного порога. Если в какой-то момент времени достигается критическое значение (величины допустимых порогов задаются априори и могут быть различными для разных разломов), то, в соответствии с законом сухого трения, происходит сброс напряжения посредством изменения значений неупругих смещений δ_t, δ_l, δ_n, и эта ситуация интерпретируется как землетрясение. Считается, что те части разлома, в которых произошли землетрясения, находятся в состоянии крипа. Такое состояние отличается от нормального более быстрым ростом неупругих смещений и продолжается до тех пор, пока напряжение не уменьшится до определенного уровня. Основным результатом процесса моделирования является искусственный каталог землетрясений. Принадлежащие одному разлому ячейки, в которых произошло землетрясение в момент времени t_i, объединяются в одно событие, параметры которого: (а) время — t_i; (б) географические координаты эпицентра и глубина — взвешенные суммы координат и глубин ячеек; (в) магнитуда (энергетическая характеристика землетрясения) — величина, вычисленная по известной в сейсмологии формуле [7]

$$M = 0.98\lg S + 4.07, \qquad (5)$$

где S — сумма площадей ячеек (в км^2). Кроме того, в каждый момент времени модель позволяет получить картину мгновенной кинематики блоков и информацию о характере их взаимодействия вдоль границ.

Детальное описание основных принципов построения блоковых моделей динамики литосферы можно найти, например, в [2, 3].

3. Распараллеливание: ключевые моменты и эффективность

Сферическая блоковая модель, являющаяся достаточно ресурсоемкой при расчетах на последовательных ЭВМ, допускает эффективное применение параллельных технологий, что обуславливает ориентацию вычислительных модулей на многопроцессорную технику. При распараллеливании по стандартной схеме «мастер-рабочий» [5, 8] используется библиотека MPI, при этом единый загрузочный модуль запускается на всех процессорах. Блок-схема основной вычислительной процедуры представлена на Рис. 1. Приведем необходимые пояснения. В начале своей работы программа назначает один из процессоров мастером. Затем загружается информация о блоковой структуре, выполняются предварительные вычисления (например, считается матрица A системы (3)). На каждом шаге дискретного времени наиболее трудоемкой процедурой является определение значений сил и неупругих смещений во всех ячейках структуры. Поскольку эти вычисления могут быть проведены независимо друг от друга, их необходимо равномерно разделить между процессорами. Именно распределение ячеек на равные порции по процессорам и организация оптимального обмена информацией между процессорами являются ключевыми моментами описываемого алгоритма распараллеливания. Остановимся на них более подробно.

В рассматриваемой задаче естественным способом хранения основных данных по сегментам (координаты ячеек, силы, смещения) является организация двумерных массивов вида *f[i, j]*, где i — номер ячейки поверхностной дискретизации сегмента в глобальной нумерации по всем сегментам структуры, j — номер слоя при дискретизации соответствующего сегмента по глубине (такой способ обеспечивает совместимость данных для всех модификаций модели). Распределение ячеек дискретизации сегментов по процессорам выполняется без нарушения целостности слоя, как показано на Рис. 2. Динамическое распределение памяти под двумерные массивы на каждом процессоре осуществляется путем отведения памяти под массив (его длина равна количеству поверхностных ячеек, попавших на процессор) указателей на одномерные массивы переменной длины, отвечающие за дискретизацию по глубине.

Обмен информацией реализован по схеме, отраженной на Рис. 1, где операции, выполняемые только мастером, помечены символом «M», только рабочими — символом «W» (остальные операции выполняются на всех процессорах). На каждом шаге мастер вычисляет новые значе-

ния смещений блоков, граничных блоков и подстилающей среды (что требует незначительного времени из-за малой размерности системы (3)), после чего рассылает их на рабочие процессоры. Пересчитанные значения сил, неупругих смещений и вектора b возвращаются мастеру, происходит переход к следующему шагу. Отметим, что при реализации межпроцессорных пересылок используются как коллективные обмены (например, при рассылке мастером решения системы (3)), так и обмены типа «точка-точка» (например, при передаче мастеру информации о модельных событиях).

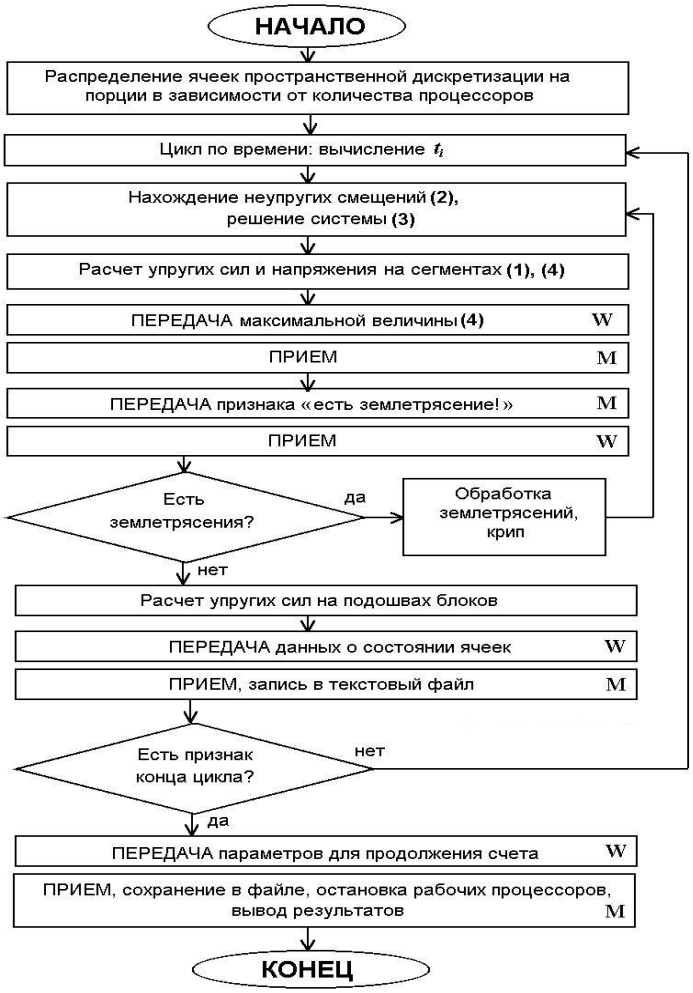


Рис. 1. Схема основной вычислительной процедуры

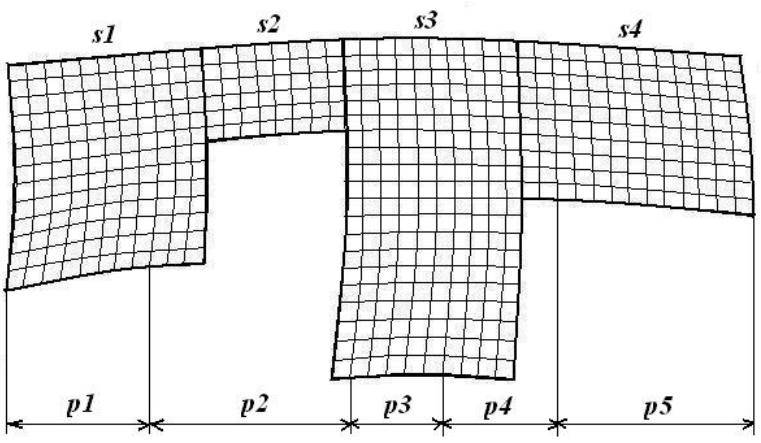


Рис. 2. Распределение ячеек дискретизации сегментов (*s1–s4*) по процессорам (*p1–p5*)

При обработке ситуации, трактуемой как землетрясение, схема несколько усложняется, поскольку в этом случае мастер опрашивает все рабочие процессоры до тех пор, пока существуют ячейки сегментов, находящиеся в критическом состоянии; после чего мастер получает информацию о происшедших событиях и записывает ее в файл специальной структуры. При такой организации время вычислительной работы на каждом рабочем процессоре оказывается значительно больше времени обмена данными с мастером, и за счет этого достигается довольно высокая полезная загрузка отдельного процессора.

Моделирование проводилось на вычислительных комплексах МВС-1000/64 (ИММ УрО РАН, г. Екатеринбург) и на МВС-50К (Межведомственный Суперкомпьютерный Центр, г. Москва). Дополнительно на МВС-50К были проведены эксперименты по изучению зависимости времени решения задачи от числа процессоров. Был выбран наиболее «времяемкий» вариант из просчитанных (с большим числом модельных землетрясений) и рассмотрено 100 итераций по времени (типовой расчетный вариант содержит 20000 шагов). Результаты тестирования представлены в таблице 1. Анализировались следующие величины: ускорение $S_p = T_1 / T_p$ и эффективность $E_p = S_p / p$; здесь T_p — время выполнения программы на многопроцессорной машине для p процессоров, T_1 — время работы последовательного алгоритма.

Таблица 1. Время счета, ускорение и эффективность для разного числа процессоров.

p	T_p (сек)	S_p	E_p
1	9988.42	—	—
2	4994.29	2.00	1.00
4	2680.78	3.73	0.93
8	1317.04	7.58	0.95
16	686.95	14.54	0.91
32	366.08	27.28	0.85
64	197.80	50.50	0.84

Из таблицы следует, что эффективность распараллеливания достаточно высока, причем она, уменьшаясь с ростом числа задействованных процессоров, не падает ниже разумного уровня. Представляется, что резервом для улучшения эффективности является оптимизации передачи/записи информации о модельных землетрясениях.

Перейдем к теоретическому анализу эффективности параллельного алгоритма. Отметим, что способность такого алгоритма эффективно использовать процессоры при повышении сложности вычислений является важной характеристикой выполняемых расчетов. Параллельный алгоритм называется масштабируемым (scalable), если при росте числа процессоров он обеспечивает увеличение ускорения при сохранении постоянного уровня эффективности использования процессоров (например, вследствие увеличения размерности задачи) [9].

Рассмотрим идеализированную модель, учитывающую различные характеристики функционирования алгоритма. Оценим время счета одного шага дискретного времени как время, необходимое для вычисления сил и смещений во всех ячейках пространственной дискретизации подошв блоков и сегментов разломов (затраты на остальные расчеты, включая решение системы (3), пренебрежимо малы):

$$t_{calc} = t_{sg} N_{sg} + t_{bl} N_{bl},$$

где t_{sg} и t_{bl} — средние времена вычислений в одной ячейке дискретизации сегмента разлома и подошвы блока; N_{sg} и N_{bl} — количества ячеек на сегментах и подошвах, соответственно. Фактически последние два параметра определяют размерность задачи для фиксированной блоковой структуры. Чтобы оценить время, необходимое для обмена информацией на каждом шаге, введем время t_{msg}, требующееся для отправки/получения сообщения длиной в L слов:

$$t_{msg} = t_s + t_w L,$$

где t_s — время инициализации обмена, t_w — время передачи одного слова (оба параметра являются характеристиками машины). В данном алгоритме $L = d_1 n_{bl} + d_2$ (n_{bl} — количество блоков в структуре, константы d_1 и d_2 могут быть выписаны явно). Следовательно, время обмена в случае запуска программы на p процессорах (мастер и $p-1$ рабочих), с учетом процедур отправки/получения, можно определить следующим образом:

$$t_{exch} = 2(p-1)(t_s + t_w(d_1 n_{bl} + d_2)).$$

Время работы последовательного алгоритма T_1 и время выполнения программы на p процессорах T_p могут быть найдены по формулам:

$$T_1 = t_{calc} = t_{sg} N_{sg} + t_{bl} N_{bl},$$

$$T_p = t_{calc} / p + t_{exch} = (t_{sg} N_{sg} + t_{bl} N_{bl}) / p + 2(p-1)(t_s + t_w(d_1 n_{bl} + d_2)).$$

Отсюда вычисляем эффективность:

$$E_p = T_1 / T_p p = \frac{t_{sg} N_{sg} + t_{bl} N_{bl}}{t_{sg} N_{sg} + t_{bl} N_{bl} + 2p(p-1)(t_s + t_w(d_1 n_{bl} + d_2))}.$$

Таким образом, эффективность уменьшается с ростом p, t_s, t_w и n_{bl}; увеличивается с ростом t_{sg}, t_{bl}, N_{sg} и N_{bl}. Для получения примерно постоянной эффективности необходимо добиться выполнения соотношения $T_1 \approx E_p T_p p$, т.е.

$$t_{sg} N_{sg} + t_{bl} N_{bl} \approx E_p (t_{sg} N_{sg} + t_{bl} N_{bl} + 2p(p-1)(t_s + t_w(d_1 n_{bl} + d_2))).$$

Чтобы сделать данное соотношение нечувствительным к росту p, следует выбрать зависимость между размерностью решаемой задачи и числом процессоров в виде $N_{sg} = c_1 p(p-1)$ и $N_{bl} = c_2 p(p-1)$, где c_1, c_2 — некоторые константы (приведенные соотношения можно считать функцией изоэффективности задачи). Получаем условие, при выполнении которого с ростом числа процессоров эффективность остается постоянной:

$$c_1 t_{sg} + c_2 t_{bl} \approx E_p (c_1 t_{sg} + c_2 t_{bl} + 2t_s + 2t_w(d_1 n_{bl} + d_2)).$$

Отсюда (при $E_p < 1$) можно найти подходящие значения коэффициентов c_1 и c_2. Итак, для фиксированной блоковой структуры увеличение количества ячеек на сегментах и подошвах (отметим, что при этом улучшается точность вычислений) пропорционально $p(p-1)$, приводя к увеличению времен счета T_1 и T_p, обеспечивает постоянную эффективность, не зависящую от p, и, следовательно, рост ускорения пропорционально p. С другой стороны, найденные соотношения позволяют по желаемым значениям N_{sg} и N_{bl} определять число процессоров p, на котором теоретически достигается предписанная эффективность E_p.

Дополнительно отметим, что из формулы для вычисления T_p может быть найдено оптимальное число процессоров в смысле минимизации теоретического времени счета при решении задачи с фиксированными значениями всех параметров (в том числе шагов пространственной дискретизации).

4. Некоторые результаты моделирования

Приведем некоторые результаты применения модификации сферической блоковой модели, учитывающей неоднородность литосферы, к исследованию динамики и сейсмичности глобальной системы крупнейших тектонических плит, покрывающих всю поверхность Земли.

Рассматриваемая структура включает 15 плит (Рис. 3), для которых используются следующие обозначения: N ─ Наска, SA ─ Южноамериканская, Co ─ Кокос, Ca ─ Карибская, NA ─ Североамериканская, P ─ Тихоокеанская, Af ─ Африканская, An ─ Антарктическая, E ─ Евразийская, Ar ─ Аравийская, I ─ Индийская, S ─ Сомалийская, Ph ─ Филиппинская, Au ─ Австралийская, F ─ Хуан де Фука. Движение подстилающей среды определяется как вращение на сфере согласно модели HS2-NUVEL1 [10]. Модельные глубины плит выбираются с учетом распределения по глубине реальных землетрясений.

Было исследовано поведение точек, принадлежащих границам плит, для которых четко определяется один из трех типов (дивергентные, конвергентные, трансформные). Рассмотрены такие характерные зоны, как, например, границы Южная Америка/Наска, Тихий Океан/Наска, Южная Америка/Африка, Индия/Евразия, вокруг Филиппин и т.д. С помощью двух смещений граничной точки в связанной с ней системе координат (как точки «правого» и «левого» блоков) вычислялось ее относительное смещение, тем самым устанавливался характер взаимодействия между плитами и типы границ (см. Рис. 3), в принципе соответствующие реальным [11].

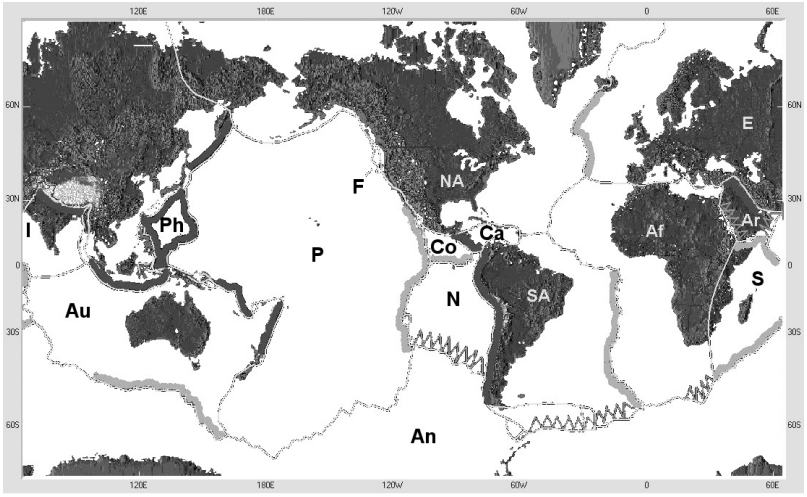


Рис. 3. Глобальная система тектонических плит и результаты моделирования характера межплитовых границ: дивергентные границы плит (растяжение, светлая штриховка), конвергентные границы плит (сжатие, темная штриховка), трансформные границы плит (скольжение, зубчатая штриховка)

В качестве основного параметра, определяющего качество моделирования, рассматривается пространственное распределение сильнейших событий. Был проведен сравнительный анализ искусственного каталога землетрясений, который создавался с использованием формулы (5), и реального, извлеченного из глобального каталога NEIC [12] и включающего события с магнитудой не менее 7.5 за период времени с 01.01.1900 по 30.06.2009 без ограничений по глубине и местоположению, см. Рис. 4.

Полученные искусственные каталоги землетрясений (на Рис. 5 приведен «лучший» из них в смысле соответствия реальному) обнаружили ряд черт, присущих реальной сейсмичности, например: (а) наличие двух основных сейсмических поясов, Тихоокеанского и Средиземноморско-Трансазиатского, где происходит большая часть сильных событий; (б) увеличение сейсмической активности вблизи точек, где сходятся три и более плит. Установлено соответствие сейсмически активных (границы Кокос/Карибы, Индия/Евразия, Наска/Южная Америка, район Калифорнии, Аравия/Евразия, юго-восток, восток, северо-восток и особенно север Австралийской плиты, вокруг Филиппин) и «спокойных» (юг Тихоокеанской плиты, Наска/Тихий Океан, восток и юго-запад Африки, Индия/Австралия, Северная Америка/Евразия) регионов реальным, что следует считать позитивным фактом.

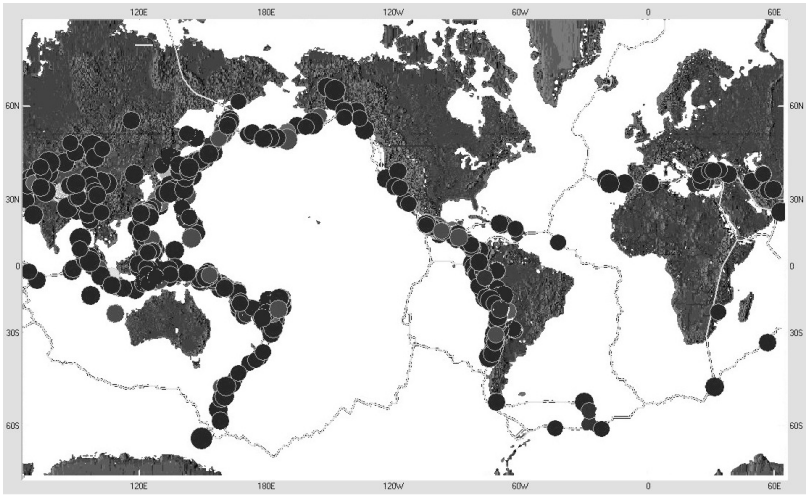


Рис. 4. Зарегистрированная сейсмичность: эпицентры сильнейших землетрясений с магнитудой не менее 7.5, каталог NEIC

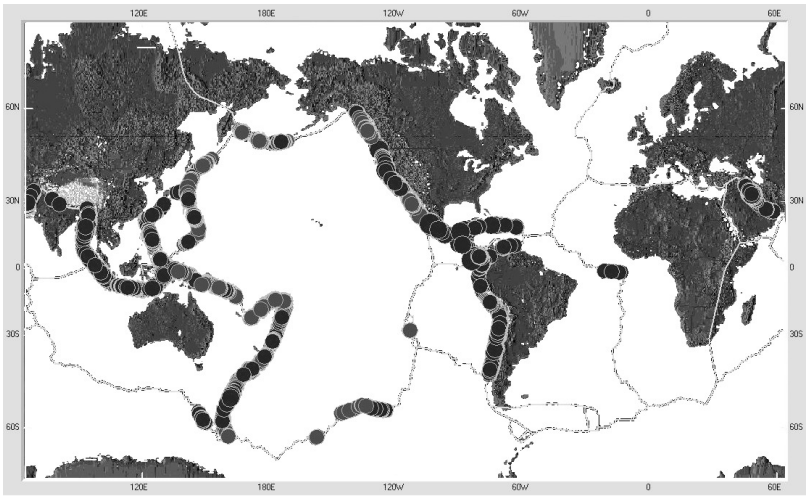


Рис. 5. Эпицентры сильнейших модельных событий с магнитудой не менее 7.5, модификация с переменной глубиной

Отметим, что заметное сходство в расположении эпицентров сильнейших землетрясений в модельной и реальной сейсмичности наблюдается уже в вариантах, полученных посредством модификации без глубины [4]. Для модификации с переменной глубиной дополнительным соответствием реальной сейсмичности является, в частности, появление мощных землетрясений на известных своей сейсмической активностью границах Южная Америка/Наска и Индия/Евразия (см. Рис. 5). Улучшение модельного распределения эпицентров происходит не по причине подбора параметров разломов (которые во всех экспериментах постоянны), а вследствие усовершенствования самой модели.

Анализ параметров закона Гутенберга–Рихтера (характеризующего зависимость количества землетрясений от магнитуды, Рис. 6) выявил не вполне удовлетворительные результаты. В частности, углы наклона графиков повторяемости в модельных вариантах больше реального, а сами графики линейны только в диапазоне средних магнитуд (в то время как график для зарегистрированной глобальной сейсмичности близок к линейному с угловым коэффициентом, равным 1). Для сглаживания различий в угловых коэффициентах следует увеличить число модельных событий вне области средних магнитуд; в этих целях предполагается провести дополнительные серии вычислительных экспериментов.

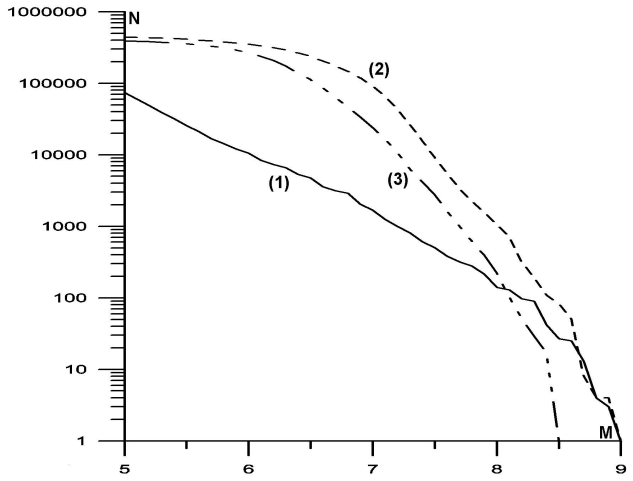


Рис. 6. Графики повторяемости, построенные по реальному (1) и двум модельным (2), (3) каталогам; N — аккумулированное число землетрясений, M — магнитуда

Изучались и другие характеристики качества моделирования: распределение событий по глубине, параметры сейсмического цикла, скорости смещения граничных точек и др. В результате анализа была скорректирована «классическая» аппроксимация глобальной системы тектонических плит, именно были дополнительно введены 5 блоков, представляющие собой искусственные геологические структуры. Предполагается, что это позволит получить модельные события на больших глубинах в регионах, где зарегистрирована сейсмичность такого рода, тем самым увеличить количество сильных землетрясений в модели и исправить угол наклона графика повторяемости. Новая аппроксимация (Рис. 7) включает 20 блоков (дополнительные с номерами 16–20), 213 вершин и 231 разлом.

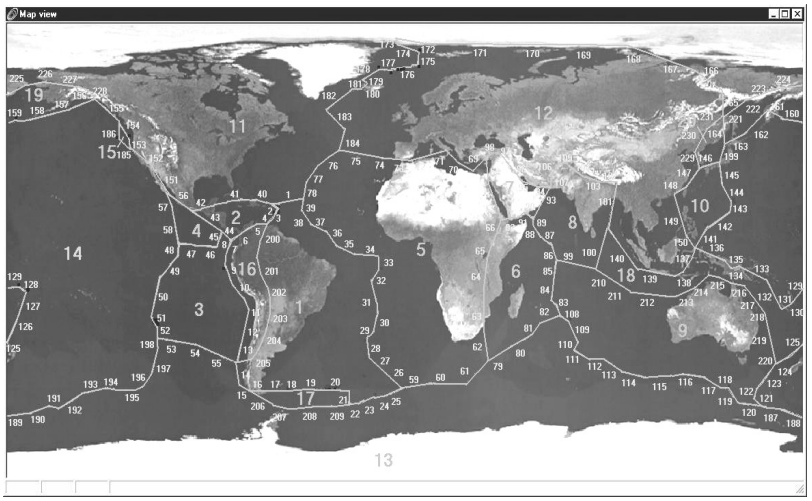


Рис. 7. Новая аппроксимация глобальной системы тектонических плит (приведены номера блоков и разломов)

5. Заключительные замечания

Тщательный количественный анализ реального и модельных распределений представляется на данном этапе преждевременным, поскольку, во-первых, зарегистрированная сейсмичность мала на многих границах плит (ввиду недостаточной длины интервала наблюдения), а во-вторых, диапазон изменения модельной магнитуды довольно узок и не вполне соответствует реальному. Задача приведения магнитудных интервалов в соответствие требует дополнительного анализа в том числе и потому, что такое соответствие может зависеть от конкретного сейсмического региона. Отметим, что упрощения, принятые в сферической блоковой модели, не дают возможности делать какие-либо выводы о качестве аппроксимации реального сейсми-

ческого потока модельным в конкретной точке. Однако, проведенный анализ (в частности, результаты, представленные в настоящей статье) позволяет судить о достаточной степени адекватности модели в смысле воспроизведения важных закономерностей, обнаруженных в зарегистрированной сейсмичности эмпирическим путем. Установлено, что модификация, учитывающая переменную глубину сферического слоя, в целом адекватнее описывает динамику и сейсмичность глобальной системы тектонических плит по сравнению с моделью без глубины, несмотря на то, что глубина слоя гораздо меньше линейных размеров плит и, казалось бы, при моделировании ей можно пренебречь. Полученные данные предполагается использовать как для дальнейшего развития модели (например, для реализации возможности использования результатов моделирования мантийных движений в качестве входа), так и для исследования характера сейсмического потока в конкретных регионах (например, с целью изучения регионального сейсмического риска).

Литература

1. Gabrielov A.M., Newman W.I. Seismicity Modeling and Earthquake Prediction: a Review // Geophysical monograph 83, IUGG. –1994. –Vol. 18. –P. 7–13.
2. Keilis-Borok V.I., Soloviev A.A. (Eds.) Nonlinear Dynamics of the Lithosphere and Earthquake Prediction. –Springer, 2003. –337 p.
3. Габриэлов А.М., Кейлис-Борок В.И., Левшина Т.А., Шапошников В.А. Блоковая модель динамики литосферы // Математические методы в сейсмологии и геодинамике: Выч. Сейсмология. Вып. 19. –М.: Наука, 1986. –С. 168–178.
4. Rozenberg V.L., Sobolev P.O., Soloviev A.A., and Melnikova L.A. The Spherical Block Model: Dynamics of the Global System of Tectonic Plates and Seismicity // Pure appl. geophys. –2005. –N. 162. –P. 145–164.
5. Розенберг В.Л., Мельникова Л.А. Новая модификация сферической блоковой модели: алгоритмическая и программная реализация на МВС // Параллельные вычислительные технологии (ПаВТ-2007): Тр. междунар. конф. –Челябинск, 2007. –Т. 1. – С.221–226.
6. Розенберг В.Л., Мельникова Л.А. Сферическая блоковая модель динамики и сейсмичности литосферы: модификации, алгоритмы и вычислительные эксперименты // Параллельные вычислительные технологии (ПаВТ-2008): Тр. междунар. конф. –Челябинск, 2008. – С.538.
7. Wells D.L., Coppersmith K.L. New Empirical Relationships among Magnitude, Rupture Length, Rupture Width, Rupture Area, and Surface Displacement // Bull. Seism. Soc. of America. –1994. –Vol. 84, N. 4. –P. 974–1002.
8. Soloviev A.A., Maksimov V.I., Rozenberg V.L., Ermoliev Y.M. Block Models of Lithosphere Dynamics: Approach and Algorithms // Lecture Notes in Computer Science 2328. Volume on Parallel Processing and Applied Mathematics. –Berlin-Heidelberg, Springer, 2001. –P. 572–579.
9. Foster I. Designing and Building Parallel Programs: [http://www.mcs.anl.gov/~itf/dbpp/], 1995.
10. Gripp A.E., Gordon R.G. Current Plate Velocities relative to the Hotspots Incorporating the NUVEL-1 Global Plate Motion Model // Geoph. Res. Let. –1990. –Vol. 17, N. 8. –P. 1109–1112.
11. Mutter J.C. Seismic Images of Plate Boundaries // Scient. American. –1986. –Vol. 254. –P. 66–75.
12. Global Hypocenters Data Base, NEIC/USGS. –Denver, CO, 2008.

Распараллеливание решения линейной обратной задачи гравиметрии на МВС-1000 и графических процессорах *

Е.Н. Акимова, Д.В. Белоусов

Для решения линейной обратной задачи гравиметрии о восстановлении переменной плотности в слое численно реализованы на многопроцессорном вычислительном комплексе МВС–1000 и графических процессорах на видеокартах NVIDIA регулярные параллельные итерационные алгоритмы. Для решения модельной задачи гравиметрии и задачи с реальными данными проведено сравнение времени счета параллельных алгоритмов на видеоускорителях GeForce и МВС–1000 с анализом эффективности и ускорения.

1. Введение

Рассматривается линейная обратная задача гравиметрии о восстановлении переменной плотности в горизонтальном или криволинейном слое. После предварительной обработки гравитационных данных по методике, предложенной П.С. Мартышко и И.Л. Пруткиным [1], задача нахождения неизвестной плотности сводится к решению линейного двумерного интегрального уравнения Фредгольма первого рода [2].

Задача гравиметрии являются существенно некорректной задачей, решение которой обладает сильной чувствительностью к погрешности правой части, полученной в результате измерений и предварительной обработки геофизических данных. При разработке методов решения задач используются идеи итеративной регуляризации [3].

Для решения линейной обратной задачи гравиметрии о восстановлении переменной плотности в слое численно реализованы на многопроцессорном вычислительном комплексе МВС–1000 и графических процессорах на видеокартах NVIDIA регулярные параллельные итерационные алгоритмы.

Для решения модельной задачи гравиметрии и задачи с реальными данными проведено сравнение времени счета параллельных алгоритмов на видеоускорителях GeForce GTX 285, GeForce GTX 260 и МВС–1000/64 с анализом эффективности и ускорения.

2. Параллельные алгоритмы решения линейной обратной задачи гравиметрии о восстановлении плотности в слое

Одной из важнейших моделей строения земной коры является модель горизонтальной слоистой среды.

Рассматривается задача о нахождении переменной плотности $\sigma = \sigma(x,y)$ в горизонтальном слое $П = \{(x,y,z) \in R^3 : (x,y) \in D,\ H_1 \le z \le H_2\}$, где H_1, H_2 – константы, либо криволинейном слое $П1 = \{(x,y,z) \in R^3 : (x,y) \in D,\ H_1(x,y) \le z \le H_2(x,y)\}$ по гравитационным данным, измеренным на площади $D = \{(x,y) \in R^2 : a \le x \le b,\ c \le y \le d\}$ земной поверхности. Используется априорная информация об отсутствии аномалий плотности вне слоя с криволинейными границами $H_1 = H_1(x,y)$ и $H_2 = H_2(x,y)$ такими, что $H_1 < H_2$ $\forall (x,y)$, и выполняется условие $H_i(x,y) \underset{\substack{x\to\pm\infty \\ y\to\pm\infty}}{\longrightarrow} h_i = \text{const}$.

При этом предполагается, что распределение плотности $\sigma = \sigma(x,y)$ внутри слоя не зависит от z (ось z направлена вниз) (рис. 1).

*Работа поддержана грантом РФФИ № 09-01-00053, Междисциплинарным проектом УрО РАН, Программой Президиума РАН.

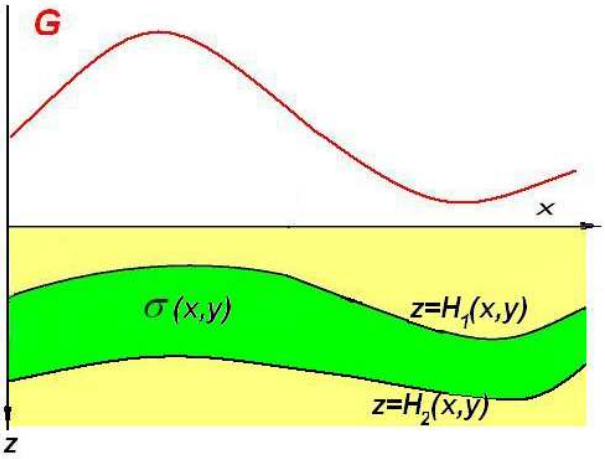


Рис. 1. Задача о нахождении плотности в слое

Задача нахождения неизвестной плотности $\sigma(x,y)$ сводится к решению линейного двумерного интегрального уравнения Фредгольма первого рода

$$A\sigma \equiv f \iint_{a\,c}^{b\,d} \left\{ \frac{1}{[(x-x')^2+(y-y')^2+H_1^2(x',y')]^{1/2}} - \right.$$

$$\left. - \frac{1}{[(x-x')^2+(y-y')^2+H_2^2(x',y')]^{1/2}} \right\} \sigma(x',y')dx'dy' = \Delta g(x,y), \tag{1}$$

где f — гравитационая постоянная, $\Delta g(x,y)$ — гравитационный эффект, порождаемый источниками в горизонтальном или криволинейном слое.

После дискретизации уравнения на сетке, где задана $\Delta g(x,y)$, и аппроксимации интегрального оператора по квадратурным формулам задача (1) сводится к решению системы линейных алгебраических уравнений (СЛАУ) либо с симметричной положительно определенной матрицей (горизонтальный слой), либо с несимметричной матрицей (криволинейный слой). Так как уравнение (1) относится к классу некорректно поставленных задач, то СЛАУ, возникающее в результате дискретизации уравнения, является плохо обусловленной и преобразуется к виду

$$(A+\alpha E)z = b, \tag{2}$$

где α — параметр регуляризации.

В случае криволинейного слоя исходная матрица СЛАУ несимметрична, поэтому эта система предварительно преобразуется к виду

$$(A^TA+\alpha' E)z = A^Tb, \tag{3}$$

где A^T — транспонированная матрица, α' — параметр регуляризации.

В.В. Васин предложил методику использования нескольких регулярных методов решения некорректных задач для сравнения результатов.

Для решения уравнений (2) и (3) используются регулярные итерационные методы градиентного типа [4]: метод минимальных невязок, метод наискорейшего спуска, метод минимальной ошибки и метод простой итерации (МПИ) в виде

$$z^{k+1} = z^k - \frac{1}{\lambda_{max}}[(A+\alpha E)z^k - b], \tag{4}$$

где λ_{max} — максимальное собственное значение матрицы $A+\alpha E$ (симметричный случай).

Условием останова итерационных процессов является следующее: $\dfrac{\|Az^k-b\|}{\|b\|} < \varepsilon.$

В работах [5]— [6] предложены и численно реализованы регулярные параллельные итерационные алгоритмы решения линейной обратной задачи гравиметрии (1) с помощью библиотеки MPI [7] на языке Фортран на многопроцессорном вычислительном комплексе МВС–1000 — российском массивно-параллельном суперкомпьютере кластерного типа с распределенной памятью, установленном в Институте математики и механики УрО РАН.

Алгоритмы были реализованы на следующих вычислителях:

1. МВС–1000/17ЕК (UM32), состоящем из 16 2-х процессорных модулей Xeon 2.4 ГГц, интерфейса GigabitEthernet и 68 Гбайт оперативной памяти;

2. МВС–1000/64 (UM64), состоящем из 14 2-х процессорных 2-х ядерных модулей AMD Opteron 64 bit (2.6 Ггц), интерфейса GbitEthernet и 112 Гб оперативной памяти.

Распараллеливание итерационных методов градиентного типа основано на разбиении матрицы A горизонтальными полосами на m блоков, а вектора решения z и вектора правой части b СЛАУ на m частей так, что $n = m \times L$, где n — размерность системы уравнений, m — число процессоров. На каждой итерации каждый из m процессоров вычисляет свою часть вектора решения. В случае умножения матрицы A на вектор z каждый из m процессоров умножает свою часть строк матрицы A на вектор z. В случае матричного умножения $A^T A$ каждый из m процессоров умножает свою часть строк транспонированной матрицы A^T на всю матрицу A. Host-процессор отвечает за пересылки данных и также вычисляет свою часть вектора решения (рис. 2). Для метода простой итерации (4) максимальное собственное значение λ_{max} матрицы $A + \alpha E$ находится с помощью степенного метода с использованием параллельного алгоритма умножения матрицы на вектор.

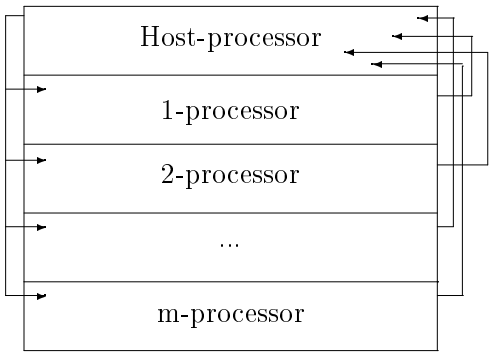


Рис. 2. Схема распределения данных по процессорам для итерационных методов

В предыдущих работах для решения линейной задачи гравиметрии проведен анализ эффективности и ускорения параллельных алгоритмов

$$S_m = T_1/T_m, \qquad E_m = S_m/m,$$

где T_m — время выполнения параллельного алгоритма на МВС–1000 с числом процессоров m ($m > 1$), T_1 — время выполнения последовательного алгоритма на одном процессоре. T_m представляет собой совокупность чистого времени счета и накладных расходов на межпроцессорные обмены $T_m = T_c + T_o$. Число процессоров m соответствует упомянутому разбиению векторов на m частей и разбиению исходной области на m подобластей.

В общем случае эффективность распараллеливания меняется в пределах $0 < E_m < 1$. В идеальном случае при равномерной и сбалансированной загрузке процессоров и минимальном времени обменов между ними E_m близко к единице, но при решении практических задач она уменьшается за счет накладных расходов.

При решении задачи о восстановлении плотности в слое на МВС–1000 с помощью параллельных алгоритмов матрица СЛАУ большой размерности формируется и хранится в памяти каждого процессора по частям, что дает эффективность распараллеливания $E_m > 1$.

3. Результаты численных экспериментов на МВС—1000

Задача 1. На многопроцессорном вычислительном комплексе МВС—1000/64 решена модельная задача о восстановлении плотности в горизонтальном слое между глубинами $H_1 = 1$ км и $H_2 = 1.5$ км для области S_1, имеющей размеры 10×10 км^2. Шаги сетки: $\Delta x \approx \Delta y \approx 0.05$ км. Гравитационная постоянная $f = 6.67 \cdot 10^{-8}$см^3/г$\cdot$с^2.

После дискретизации исходного уравнения на сетке, где задана функция $\Delta g(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам задача (1) сводится к системе линейных алгебраических уравнений с симметричной матрицей 40000×40000.

Для решения задачи использовался параллельный итеративно регуляризованный метод простой итерации с параметром регуляризации $\alpha = 0.0005$.

На рис. 3 изображено исходное аномальное гравитационное поле для области S_1.

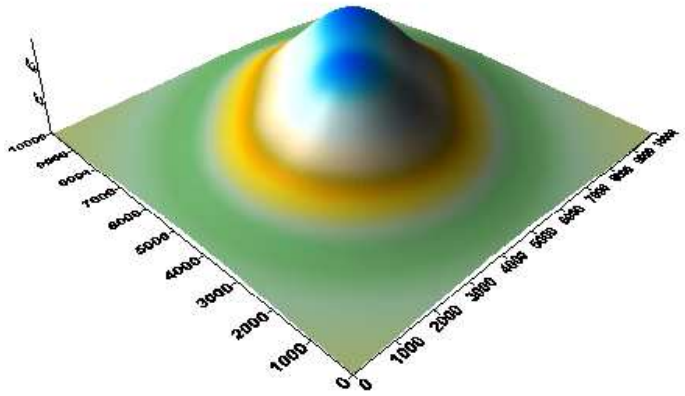


Рис. 3. Аномальное гравитационное поле $\Delta g(x, y)$ для области S_1

На рис. 4 изображены линии уровня и распределение плотности в слое, восстановленной по выделенному аномальному полю для области S_1.

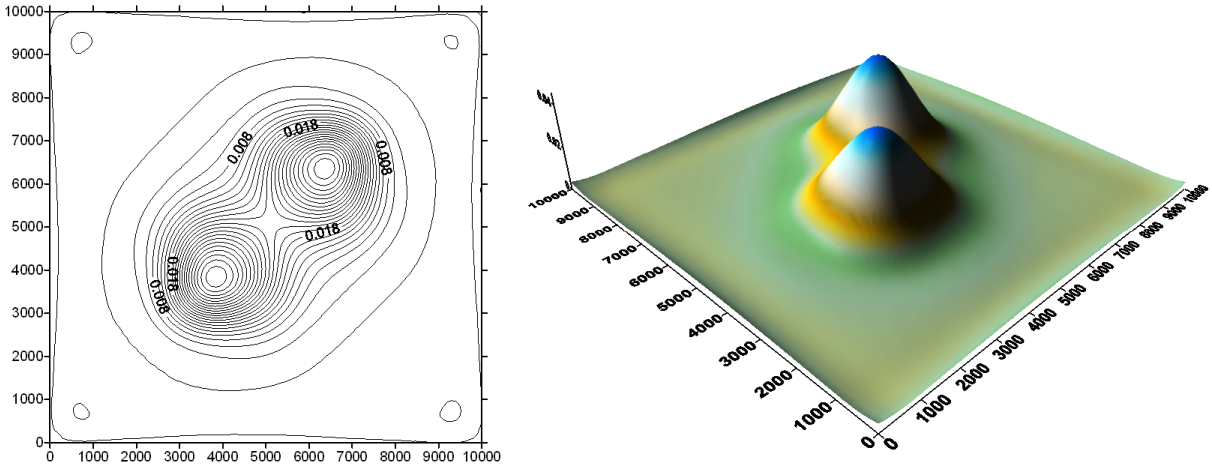


Рис. 4. Линии уровня и распределение восстановленной плотности в слое для области S_1

Задача 2. На многопроцессорном вычислительном комплексе МВС—1000/64 решена задача с реальными данными о восстановлении плотности в горизонтальном слое между глубинами $H_1 = 10$ км и $H_2 = 20$ км для области S_2, имеющей размеры 120×220 км^2. Шаги сетки: $\Delta x \approx 0.6$ км, $\Delta y \approx 1.1$ км. Гравитационная постоянная $f = 6.67 \cdot 10^{-8}$см^3/г$\cdot$с^2.

После дискретизации исходного уравнения на сетке, где задана функция $\Delta g(x, y)$, и аппроксимации интегрального оператора по квадратурным формулам задача (1) сводится к системе линейных алгебраических уравнений с симметричной матрицей 40000×40000.

Для решения задачи использовался параллельный итеративно регуляризованный метод простой итерации с параметром регуляризации $\alpha = 0.001$.

На рис. 5 изображено исходное аномальное гравитационное поле для области S_2.

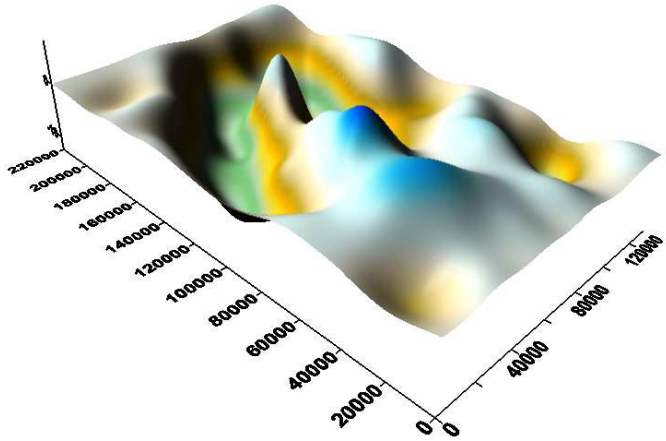


Рис. 5. Аномальное гравитационное поле $\Delta g(x, y)$ для области S_2

На рис. 6 изображены линии уровня и распределение плотности в слое, восстановленной по выделенному аномальному полю для области S_2.

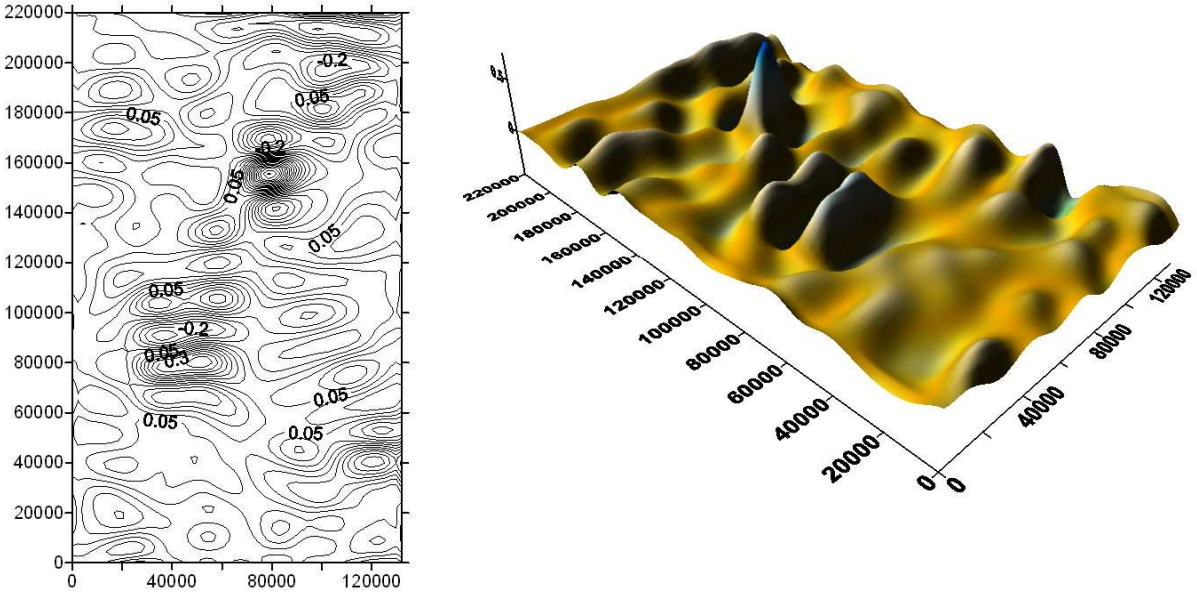


Рис. 6. Линии уровня и распределение восстановленной плотности в слое для области S_2

Результаты решения задачи 2 переданы специалистам в Институт геофизики УрО РАН для геофизической интерпретации.

В табл. 1 приведены времена счета и коэффициенты ускорения и эффективности решения модельной задачи о восстановлении плотности в слое с использованием параллельного алгоритма МПИ (число итераций — 72) на МВС–1000/64 для 200×200 точек сетки.

В табл. 2 приведены времена счета и коэффициенты ускорения и эффективности решения задачи гравиметрии с реальными данными с использованием параллельного алгоритма МПИ (число итераций — 430) на МВС–1000/64 для 200×200 точек сетки.

Отметим, что матрица СЛАУ формируется и хранится в памяти каждого процессора по частям.

Результаты вычислений показывают, что решение задач гравиметрии о нахождении плотности в слое на МВС–1000 с использованием параллельных алгоритмов существенно уменьшает время счета.

Таблица 1 . Решение модельной задачи 1 о восстановлении плотности в слое

m	T_m, min.	S_m	E_m
1	9.15	—	—
2	5.39	1.70	0.85
3	3.27	2.80	0.93
4	2.76	3.32	0.83
6	1.48	6.18	1.03
8	1.43	6.40	0.80
10	1.15	7.96	0.80
15	0.72	12.7	0.85

Таблица 2 . Решение задачи гравиметрии 2 о восстановлении плотности в слое

m	T_m, min.	S_m	E_m
1	55.82	—	—
2	32.96	1.69	0.85
3	20.83	2.80	0.93
4	15.13	3.69	0.92
8	7.72	7.23	0.90
10	6.26	8.92	0.89
15	4.16	13.4	0.89
20	3.28	17.0	0.85
30	2.12	26.3	0.88
40	1.80	30.0	0.76
60	1.19	46.9	0.78
80	0.88	63.4	0.79

4. Распараллеливание на видеоускорителях с помощью технологии CUDA и результаты численных экспериментов

Для организации параллельных вычислений актуальным в настоящее время является использование видеоускорителей (GPU) компании NVIDIA (рис. 7) [8]. Основой распараллеливания служит архитектура графических процессоров. В видеочипах NVIDIA базовым блоком является мультипроцессор, содержащий восемь—десять ядер, несколько сотен арифметико-логических устройств (ALU), несколько тысяч регистров и небольшое количество разделяемой общей памяти. Видеоускорители NVIDIA содержат быструю глобальную память с возможностью доступа к ней всех мультипроцессоров, локальную память в каждом мультипроцессоре, а также специальную память для констант. Работа нескольких ядер

мультипроцессора основана на архитектуре типа SIMD, т.е. каждый из процессоров выполняет одну и ту же команду над разными элементами данных.

Видеочипы GPU состоят из массивов исполнительных блоков, управляющих потоками блоков, разделяемой памяти небольшого объема и контроллеров памяти на несколько каналов. Такая архитектура позволяет чипу обрабатывать нескольких тысяч потоков данных, требующих высокой пропускной способности памяти.

Рис. 7. Видеоускоритель GeForce GTX 285

Для поддержки параллельных вычислений компания NVIDIA разработала технологию CUDA [9] — среду разработки программ на языке Си, позволяющую создавать программное обеспечение для решения сложных вычислительных задач.

Модель программирования в CUDA основывается на группировании потоков. Потоки объединяются в блоки потоков (thread block) — одномерные или двумерные сетки потоков, взаимодействующих между собой при помощи разделяемой памяти и точек синхронизации. Программа (ядро, kernel) выполняется над сеткой (grid) блоков потоков (thread blocks) (рис. 8). Каждый блок может быть одно-, двух- или трехмерным по форме и состоять из 512 потоков на текущем аппаратном обеспечении.

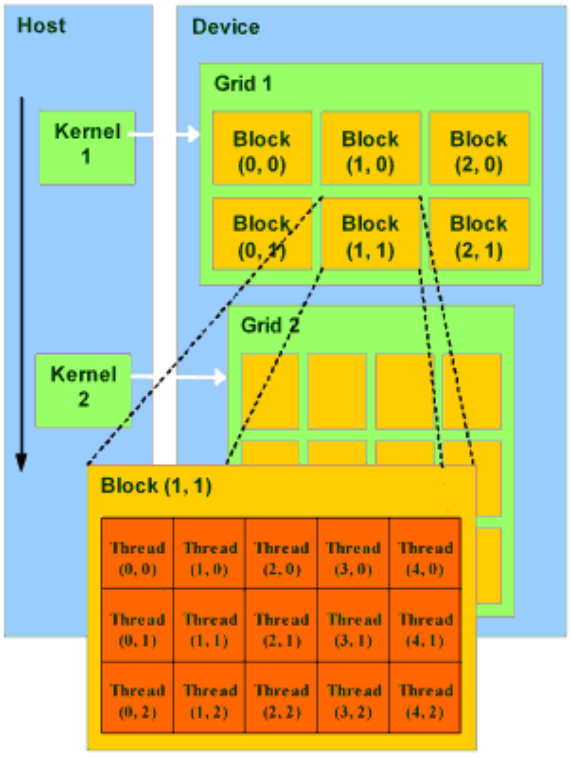


Рис. 8. Модель параллельной обработки данных

Основной процесс приложения CUDA работает на универсальном компьютере (host). CPU–код инициализирует GPU, распределяет память на видеокарте, копирует константы в память видеокарты, запускает несколько копий процессов kernel на видеокарте, копирует полученный результат из видеопамяти, освобождает память и завершает работу.

Параллельная часть кода программы, написанной на языке CUDA, выполняется как множество нитей (потоков). Для оптимизации работы с памятью нити группируют в блоки фиксированного размера. Блоки объединятся в группы блоков. Параллельная процедура выполняется над группой блоков. Типовая процедура (kernel) параллельной обработки массива Data выглядит следующим образом: kernel (Data) <<<blocks, threads >>>, где blocks — количество блоков в сетке, threads — количество потоков в блоке.

Рассмотрим распараллеливание решения обратной задачи гравиметрии (1) о восстановлении переменной плотности в слое методом простой итерации с помощью CUDA. Базовыми операциями для распараллеливания итерационных процессов (в частности, МПИ) является реализация параллельных функций над матрицами и векторами: параллельное умножение матрицы на вектор, параллельное умножение матриц и т.д.

Для оптимизации работы с памятью весь объем данных разбивается на блоки для вычисления в процедуре kernel. Для оптимизации счета матрица A порядка N и вектор Z размерности N расширяются до размерности M и дополняются нулями таким образом, чтобы M было кратно числу блоков. Размер блока $BLOCK\_SIZE$ (threads) выбирается равным 16, поскольку в одном блоке группируются до 512 потоков. Тогда количество блоков вычисляется по формуле: $\mathbf{blocks = M/BLOCK\_SIZE}$.

Синтаксис процедуры выглядит следующим образом:

$\mathbf{matrixMulVector <<< M/BLOCK\_SIZE, BLOCK\_SIZE >>> (X, A, Z)}$, где X — адрес массива из M ячеек для сохранения результатов. Тогда блок из $BLOCK\_SIZE$ потоков будет выполняться на одном мультипроцессоре. Потоки будут иметь общую разделяемую память и обрабатывать threads строк данных для нашей задачи (рис. 9).

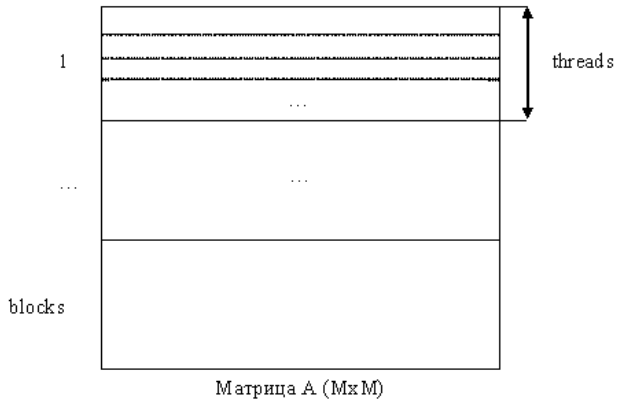


Рис. 9. Разбиение данных для процедуры CUDA

Описанные выше линейные задачи гравиметрии 1 и 2 были решены на видеоускорителях GeForce GTX 285 (GPU-1) и GeForce GTX 260 (GPU-2).

Результаты решения задач представлены на рис. 4 и 6.

В табл. 3 приводятся технические характеристики систем.

Отметим, что для эффективных расчетов объем памяти на host-компьютере должен превышать суммарный объем используемых в задаче массивов данных. Размер требуемой оперативной памяти для матрицы размерностм $M \times N$ вещественных чисел одинарной точности вычисляется по формуле: $\mathbf{size\_ts = M \times N \times sizeof\ (float)}$.

В табл. 4 приводятся результаты решения модельной задачи гравиметрии 1 и задачи гравиметрии с реальными данными 2 на Host-процессоре Intel Core I7 без использования видеоускорителей и на Host-процессоре с использованием видеоускорителей GeForce на более крупной сетке 150×150 (матрица СЛАУ 22500×22500).

Таблица 3 . Технические характеристики систем

Характеристики подсистемы GPU-1:	GeForce GTX 285
Количество процессорных ядер	240
Частота ядра (МГц)	648
Частота процессора (МГц)	1476
Частота памяти (МГц)	2484
Количество видеопамяти (Мб)	1024
Полоса пропускания памяти (Гб/сек)	159
Характеристики подсистемы GPU-2:	GeForce GTX 260
Количество процессорных ядер	192
Частота ядра (МГц)	576
Частота процессора (МГц)	1242
Частота памяти (МГц)	1998
Количество видеопамяти (Мб)	896
Полоса пропускания памяти (Гб/сек)	111.9
Характеристики CPU:	Host
Процессор (ГГц)	Intel Core I7 2.4
Оперативная память (Гб)	3
Разрядность ОС (Бит)	64

Таблица 4 . Решение задач гравиметрии

Система		Время, мин.
Host Intel Core I7	Задача 1	3.33
Host+GeForce GTX 285 (240 ядер)	Задача 1	1.25
Host+GeForce GTX 260 (192 ядра)	Задача 1	1.68
Система		Время, мин.
Host Intel Core I7	Задача 2	26.45
Host+GeForce GTX 285 (240 ядер)	Задача 2	7.42
Host+GeForce GTX 260 (192 ядра)	Задача 2	10.3

Для решения задач гравиметрии на более мелкой сетке 200×200 (матрица СЛАУ 40000×40000) требутся увеличение оперативной памяти на host–компьютере до 8 Гбайт.

Таким образом, время решения задач гравиметрии довольно большой размерности на видеоускорителях GeForce сокращает время счета и сравнимо с временем решения задач на МВС–1000/64 (см. табл. 1 и табл. 2).

5. Заключение

Для решения линейной обратной задачи гравиметрии о восстановлении переменной плотности в слое численно реализованы на многопроцессорном вычислительном комплексе МВС–1000 и графических процессорах на видеокартах NVIDIA регулярные параллельные итерационные алгоритмы.

Для решения модельной задачи гравиметрии и задачи с реальными данными проведено сравнение времени счета параллельных алгоритмов на видеоускорителях GeForce GTX 285, GeForce GTX 260 и МВС–1000/64 с анализом эффективности и ускорения.

Литература

1. Мартышко П.С., Пруткин И.Л. Технология разделения источников гравитационного поля по глубине // Геофизический журнал. 2003. Т. 25. № 3. С. 159–168.

2. Мартышко П.С., Кокшаров Д.Е. Об определении плотности в слоистой среде по гравитационным данным // Геофизический журнал. 2005. Т. 27. № 4. С. 678–684.

3. Васин В.В., Агеев А.Л. Некорректные задачи с априорной информацией. Екатеринбург: Наука, 1993. 262 с.

4. Васин В.В., Еремин И.И. Операторы и итерационные процессы Фейеровского типа. Теория и приложения. Екатеринбург: УрО РАН. 2005. 210 с.

5. Акимова Е.Н., Гемайдинов Д.В. Параллельные алгоритмы решения задачи гравиметрии о восстановлении плотности в слое // Труды института математики и механики УрО РАН. 2007. Т. 13. № 3. С. 3–21.

6. Акимова Е.Н. Параллельные алгоритмы решения обратных задач гравиметрии и магнитометрии на МВС-1000 // Вестник ННГУ. 2009. № 4. С. 181–189.

7. Baranov A.V., Latsis A.O., Sazhin C.V., Khramtsov M.Yu. The MVS-1000 System User's Guide. URL: http://parallel.ru/mvs/user.html.

8. URL: http://www.nvidia.ru/

9. URL: http://www.ixbt.com/video3/cuda-1.shtml

Параллельные вычисления в моделировании российской экономики с учетом социальной стратификации*

Н.Н. Оленёв

Экономике России для перехода к развитию необходима модернизация. В работе рассмотрена стратификация современного российского общества на десять страт, которые на плоскости с двумя осями – уровень образования по оси абсцисс и уровень доходов по оси ординат – составляют пирамиду. Описаны экономические функции указанных страт, динамика их демографической структуры, взаимодействие страт, получена динамика изменения валового национального продукта. Построенная модель используется для прогноза развития экономики России, для получения ответа на вопросы кто? и как? может участвовать в программе модернизации. Параллельные вычисления на кластерных суперкомпьютерах позволяют идентифицировать модель за приемлемое время.

1. Введение

Параллельные вычисления позволяют ускорить выполнение трудоемких расчетов, которые возникают при решении некоторых задач математического моделирования в экономике. Для проведения таких расчетов можно использовать естественный параллелизм исследуемой системы, состоящей из относительно независимых процессов (например, при расчете процессов в разных секторах экономики или процессов в разных слоях населения), расчет по каждому из которых производится независимо. Существенного ускорения при естественном распараллеливании на кластере добиться трудно, если процессы часто взаимодействуют. В настоящей работе рассмотрено разбитие общества на относительно независимые в экономическом плане социальные страты, что дает возможность получить ускорение расчетов при распараллеливании.

Параметры модели определяем параллельно по стратам. При этом, большую часть параметров модели невозможно определить непосредственно из данных статистики. Эти параметры определяем верификацией модели по статистическим данным, то есть косвенным образом, сравнивая близость расчетных и статистических временных рядов для макропоказателей.

Современная российская экономика, оказавшаяся в точке бифуркации в результате собственных проблем [1] и международного финансового кризиса, для развития нуждается в модернизации. Дальнейший рост за счет увеличения загрузки старых производственных мощностей уже невозможен, поскольку их текущая загрузка близка к предельной и может только уменьшаться по мере старения мощностей. Значит, в дальнейшем рост может идти только за счет развития, которое основано на строительстве новых производственных мощностей, сопровождаемом демонтажем старых мощностей. Как осуществить модернизацию и кто ее будет делать, зависит от сложившихся в обществе отношений между людьми.

В последнее время обсуждение произошедшей дифференциации доходов населения и его жесткой стратификации просочилась из специализированных социологических журналов в популярную прессу, чтобы стать предметом междисциплинарных обсуждений. В частности, в [2] указывается на невозможность решения назревших проблем с помощью социальной революции снизу в силу разобщенности экономических интересов страт, которые расщепляют общество поперек классов. При утрате общего языка общество становится управляемым и коррумпированным, поскольку междисциплинарную оценку заменяет финансовый интерес. В [2] указывается также на опасность самозащиты элиты: «бунт сытых – это война», конец которой вовсе не обязателен. Элита знает куда ударить, чтобы было по понятиям этой страты и наверняка больно для других страт. Для оздоровления общества требуется выработать междисциплинарные критерии оценки, новую общую эстетику, при этом применять принцип братства, который важнее

* Работа выполнена при поддержке РФФИ (проекты №№ 08-01-00377, 09-01-90201-Монг_а), РГНФ (проекты №№ 10-02-00300-а, 08-02-61201-а/Т), ПФИ Президиума РАН П-2, ПФИ ОМН РАН № 2, гранта Президента РФ по государственной поддержке ведущих научных школ (проект № НШ-3320.2010.1).

принципа соревнования. Чтобы проверить насколько такое представление о сложившихся отношениях внутри общества соответствует экономической действительности, можно построить математическую модель экономики страны, основанную на явном описании социальной стратификации с учетом демографических процессов, попробовать идентифицировать эту модель, а в дальнейшем с помощью идентифицированной модели проводить сценарные расчеты.

Для исследования сложившихся производственных и социальных отношений между людьми в настоящей работе предложена модель стратификация российского общества, в которой общество поделено на десять страт, каждая из которых характеризуется собственными особенностями и выполняет определенные экономические функции. Страты дифференцированы не только по уровню дохода, но также и по уровню образования, образуя своеобразную пирамиду. Описав экономические функции страт, демографическую динамику каждой страты, а также взаимодействие между ними, получим динамику изменения валового национального продукта. Построенную модель после ее идентификации можно будет использовать для прогноза развития экономики России.

Похожая конструкция стратификации общества используется при моделировании региональной экономики, основанном на математическом описании взаимодействий между шестью выделенными стратами в экономике Кировской области [3]. В этой стратификации удалось привязать страты к конкретным отраслям региональной экономики, что подчеркивает несовпадение экономических функций страт и позволяет дать наглядную интерпретацию их взаимодействия.

Параллельные расчеты для оценки неизвестных параметров модели (их идентификации) проводились на кластерных суперкомпьютерах ВЦ РАН, МСЦ РАН (МВС 100К) и ВятГУ (HP HPC Enigma X000), причем их удалось осуществить за приемлемое время.

2. Модель российской экономики с учетом социальной стратификации

В результате социально-экономических преобразований последних двух десятилетий произошло глубокое расслоение российского общества – его стратификация. Демократия дает всем равные возможности на старте, но люди имеют разные способности, предпочтения, получают разное образование, поэтому при выходе на экономическую сцену вместо равенства получаются почти не пересекающиеся страты, которые характеризуют основных социальных агентов современного общества. Если изобразить страты на плоскости, где по оси абсцисс указан уровень образования, а по оси ординат уровень дохода, мы получим своеобразную пирамиду (см. рис.1). Указанная пирамида характеризует сложившуюся стратификацию современного российского общества, которая как нам представляется состоит из десяти страт:

1. Властные структуры России и ее регионов - Администрация Президента и Правительства, вертикаль власти, Главы и Правительства регионов и населенных пунктов, Госдума РФ, СФ, Верховный суд.
2. Элита - элитные группы в управлении (включая резерв власти), политике (деятели партий, оппозиция), экономике (крупные предприниматели, CEO, олигархи), силовых структурах.
3. Менеджеры - предприниматели (кроме крупных), управляющие среднего звена, старшие офицеры, высококвалифицированные специалисты, новые русские.
4. Глобалисты – участники глобальных международных рынков, программисты, off-shore работники, которые легко перемещаются по странам мира.
5. Работники массовых профессий - офисные служащие, работники добывающих и перерабатывающих предприятий.
6. Работники торговли и сервиса – персонал торговли и общественного питания, финансового и страхового сервиса.
7. Интеллигенция – работники науки, образования, медицины и культуры.
8. Работники натуральных хозяйств – сельскохозяйственные работники, кустари, самозанятые.
9. Неработающие пенсионеры, безработные, беженцы.
10. Социальное дно – преступники, пьяницы, наркоманы, бродяги, бомжи и прочий андеграунд.

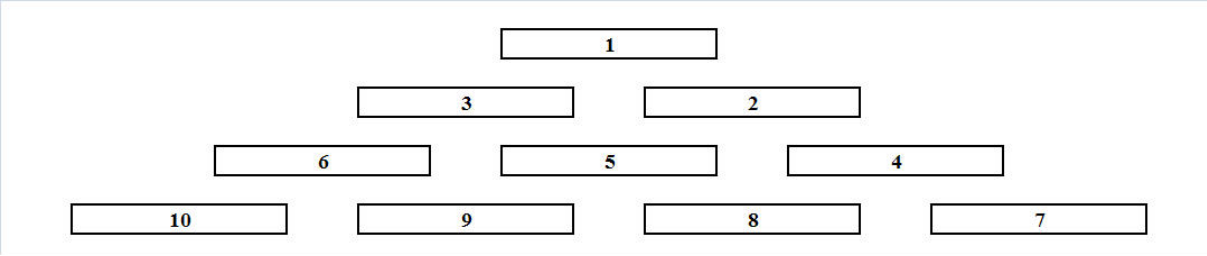


Рис.1. Стратификация современного российского общества по уровню образования (слева направо по оси абсцисс) и уровню дохода (снизу вверх по оси ординат). Краткие названия страт: 1) власть, 2) элита, 3) менеджмент, 4) глобалисты, 5) работники массовых профессий, 6) торговцы, 7) интеллигенция, 8) работники села, 9) неработающие пенсионеры, 10) социальное дно.

Модель экономики, построенная с учетом социальной стратификации, позволит не только делать прогнозные расчеты о развитии экономики России, но и позволит ответить на вопросы кто будет занят в программе модернизации и каким образом.

2.1 Динамика экономико-демографической структуры страты

Динамику численности населения, принадлежащего к определенной приносящей добавленную стоимость страте, описываем по схеме, представленной в [4,5]. Тем самым мы считаем, что страты отличаются не только уровнем дохода и уровнем образования, но также и демографическим характеристиками. В дальнейшем в этом разделе для простоты записи индекс страты опускаем. В каждый момент времени t рассмотрим динамику плотности распределения населения $x_{s,t,a}$ пола s ($s = f,m$ – женского, мужского) по возрасту a. Пусть $\beta_{s,t,a}$ - сила смертности, т.е. доля людей пола s не доживших до своего следующего дня рождения среди тех, которые в момент исполнения им a лет в году t были живы. Тогда по определению силы смертности:

$$x_{s,t+1,a+1} = x_{s,t,a}\left(1-\beta_{s,t,a}\right). \tag{1}$$

Пусть $\gamma_{t,a}$ - коэффициент рождаемости, т.е. число новорожденных, появившихся на свет от женщин, имеющих в году t возраст a в расчете на одну женщину. В модели считаем, что эти характеристики равны нулю при $a < 15$ и $a > 45$. Статистические данные соответствующим образом корректируем, перенеся внешние данные в крайние возрастные когорты. Также предположим, что $\gamma_{t,a}$ не зависит от плотности $x_{f,t,a}$. Тогда можем записать следующее соотношение для числа новорожденных в году t:

$$\Gamma_t = \sum_{a=15}^{a=45} x_{f,t,a}\gamma_{t,a} \ . \tag{2}$$

При этом число новорожденных мальчиков и девочек определяется долей мальчиков μ_t среди новорожденных: $x_{m,t,0} = \mu_t\Gamma_t$, $x_{f,t,0} = \left(1-\mu_t\right)\Gamma_t$.

Таким образом, зная плотность распределения населения по возрасту начальный год, а также коэффициенты смертности и рождаемости, можно дать прогноз эволюции этой плотности распределения. Для характеристики повозрастной динамики смертности используем формулу Гомперца-Мейкема [4,6]:

$$\beta_{s,t,a} = A_{s,t} + B_s e^{\delta_s a}. \tag{3}$$

где a – возраст, $A_{s,t}$ – фоновая или социальная составляющая смертности, которая существенно зависит от страты и может меняться со временем по мере изменения характерного уровня дохода. Параметры B_s и δ_s характеризуют потенциал биологической жизнеспособности страты, возрастную составляющую смертности, эти параметры мы считаем одинаковыми по всем стратам. Полагаем, что социальная составляющая смертности зависит от пола и уровня текущих доходов страты d_t. Если уровень доходов превысит некий минимальный уровень d^0, то социальная составляющая смертности начинает снижаться [6]. Опишем здесь эту зависимость следующей функцией:

$$A_{s,t} = A_s^+ \exp\left(-\lambda_s \left(d_t / d^0 - 1\right)_+\right), \tag{4}$$

где A_s^+ - максимальный уровень социальной смертности. Здесь и далее используется обозначение $(x)_+ = \max(0, x)$. Это означает, что социальная составляющая силы смертности падает только тогда, когда этот уровень превысит заданную величину d^0.

Разница между биологическим и фактическим уровнем средней продолжительности жизни является индикатором несовершенства социальных, экономических и экологических условий жизни страты общества. Чтобы оценить коэффициенты, характеризующие биологическую составляющую смертности, используем статистические данные Швеции за 1980-90е гг. [4], где можно пренебречь социальной составляющей. Параметры легко определятся по методу наименьших квадратов после логарифмирования.

Найденные коэффициенты подставим в нашу модель и попытаемся определить социальную составляющую смертности. Предполагаем, что она в основном определяется трудовой мотивацией, и ищем в виде функции от возраста и дохода. Согласно статистическим данным смертность растет до некоторого уровня дохода, затем начинает снижаться. Поэтому считаем, что социальная составляющая до этого уровня является постоянной величиной, а затем снижается обратно пропорционально некоторой степени дохода.

Коэффициенты социальных составляющих повозрастных коэффициентов смертности определяем методом наименьших квадратов, сравнивая со статистическими рядами.

Численность занятых в экономике считаем пропорциональной численности населения трудоспособных возрастов, причем пределы трудоспособности зависят от страт: более образованные страты позже приступают к работе из-за получения образования, но зато позднее оказываются принадлежащим к страте одиноких неработающих пенсионеров. Если a_1 – возраст начала работы населения в данной страте, а a_2 – возраст окончания работы, то численность занятого в экономике населения в указанной страте L_t определяется по формуле:

$$L_t = \sum_{s=f}^{m} \chi_s \sum_{a=a_1}^{a=a_2} x_{s,t,a}, \tag{5}$$

где неотрицательные параметры χ_s имеют смысл доли работающих соответствующего пола от численности людей трудоспособного возраста.

Заметим, что все показатели и параметры зависят от страты.

Мобильность населения между стратами определяется по заданным нормативам и осуществляется в возрасте получения соответствующего уровня образования.

2.2. Вычисление ВНП

Выпуск продукции стратой (добавленная ею стоимость) определяется численностью людей трудоспособного возраста в данной страте. Считаем, что в каждый момент времени t производительность труда каждой страты i задана нормой θ_t^i выпуска (добавленной стоимости) на единицу живого труда, который определяет выпуск y_t^i созданный стратой:

$$y_t^i = \theta_t^i L_t^i, \tag{6}$$

Сумма выпуска по всем стратам определяет валовой национальный продукт (ВНП), поскольку мы учитываем выпуск продукта гражданами России, отнесенными в ту или иную страту, независимо от их текущего места проживания.

Дадим теперь математическое описание образования добавленной стоимости внутри фиксированной страты. В модели считаем для простоты описания, что инфляция описывается единственным индексом цен: дефлятором валового внутреннего продукта $p(t)$, а все макропоказатели экономики с помощью этого индекса приведены в цены базового года.

Производительность труда θ_t^i в страте i зависит от среднего оборотного капитала k_t^i в ней и среднего уровня образования o_t^i:

$$\theta_t^i = \theta^i(k_t^i, o_t^i). \tag{7}$$

Считаем, что показатели k_t^i, o_t^i, определяющие производительность труда в i-й страте, зависят от среднего уровня доходов в соответствующей страте $s_t^i = d_t^i / L_t^i$. Здесь эти зависимости представлены соотношениями

$$k_t^i = \kappa^i s_t^i,\ o_t^i = \rho^i \sum_{a=0}^{A_i} s_{t-a}^i, \tag{8}$$

где κ^i, ρ^i - положительные константы, A^i - средний возраст обучения в i-й страте. При этом в качестве функции (7) возьмем производственную функцию леонтьевского типа: $\theta^i(k_t^i, o_t^i) = \min\left(k_t^i, o_t^i\right)$.

Пусть q^i – доля теневых доходов в i-й страте, n^i – уровень налогообложения в ней (налогообложение доходов разных страт отличается в силу принятой в России регрессионной шкалы), m^i – уровень собираемости штрафов за уклонение от налогов, тогда $q^i y_t^i$ – теневые доходы, $\left(1-q^i\right)y_t^i$ – легальные доходы, $n^i\left(1-q^i\right)y_t^i$ – налоговые отчисления, а $m^i q^i y_t^i$ – штрафные санкции с i-й страты. Налоговые отчисления и штрафные санкции поступают в консолидированный бюджет страны, образуя доходы бюджета, которыми распоряжается власть – страта 1. Власть осуществляет расходы бюджета, осуществляя трансферты во все страты. Доходы консолидированного бюджета D_t и его расходы R_t определяются соотношениями:

$$D_t = \sum_{i=1}^{10} \left(n^i - \left(n^i - m^i\right)q^i\right)y_t^i, \tag{9}$$

$$R_t = \sum_{i=1}^{10} r^i D_t \tag{10}$$

где $r^i \in [0,1)$ – доля доходов бюджета, идущая i-й страте. В случае сбалансированного бюджета $\sum_{i=1}^{10} r^i = 1$.

Для замыкания модели формирования доходов страт считаем, что доходы первой страты C_t^1 (включая коррупционные) пропорциональны числу членов этой страты трудоспособного возраста L_t^1, а расходы других страт ограничены их теневыми доходами:

$$C_t^1 = c^1 L_t^1 + \sum_{i=2}^{10} \min\left(c^i L_t^i, \left(1 - m^i\right) q^i y_t^i\right), \tag{11}$$

где c^i - норма прибыли первой страты с одного члена i-й страты (включая взятки). Из-за недостатка статистических данных, а также для простоты первоначальных расчетов дерево связей в передаче прибыли пока не строится.

Теперь можно определить реальные располагаемые доходы страт d_t^i после налогообложения, штрафных санкций и бюджетных трансфертов:

$$d_t^1 = C_t^1 - c^1 L_t^1 + r^1 D_t + \left(1 - n^1 + \left(n^1 - m^1\right) q^1\right) y_t^1, \tag{12}$$

$$d_t^i = r^i D_t + \left(1 - n^i + \left(n^i - m^i\right) q^i\right) y_t^i, \ \ (i = 2,...,10). \tag{13}$$

На основании полученных данных по имеющимся материально-сервисным и соответствующим им финансовым потокам определяем валовой национальный продукт (ВНП) как сумму первоначальных доходов отдельных ее отраслей-страт:

$$Y_t = \sum_{i=1}^{10} y_t^i . \tag{14}$$

Федеральное агентство по статистике представляет данные по валовому внутреннему продукту (ВВП). ВВП в рамках предложенной здесь модели можно вычислить, вычтя из ВНП (14) первоначальные доходы четвертой страты (глобалистов).

3. Параллельные вычисления в идентификации модели

Параллельные вычисления в идентификации параметров модели реализованы с использованием технологии MPI на языке C++, так как это описано в [7,8]. Параметры каждой страты идентифицировались параллельно по одинаковому набору параметров. Перетоки населения между стратами рассматривались как внешние заданные функции времени.

Каждый из параметров модели может изменяться заданное количество раз в определенном диапазоне. Нижнюю границу диапазонов изменения каждого параметра будем хранить в массиве limits1, верхнюю - в limits2, число изменений по каждому параметру будем хранить в массиве *N*. Таким образом, общее число итераций по всем параметрам для получения одного решения задачи (*NN*) вычисляется следующим программным кодом:

```
unsigned int NN = 1; // всего итераций
for (int p = 0; p < NUM_OF_PARAMS; p++) {
   NN *= N[p];
   step[p] = N[p] == 1 ? 0.0 : (limits2[p] - limits1[p])
                              / (N[p] - 1);
   }
```

Рис. 2. Расчет общего числа итераций и шага

Этот же программный код вычисляет величину шага изменения каждого параметра step.

Для удобства параллелизации процесса перебора параметров вместо пяти циклов будем работать с одним, с общим числом итераций NN. При каждой итерации этого общего цикла рассчитываются индексы виртуальных циклов при помощи процедуры calcIndexes(), которая выглядит следующим образом:

```
void calcIndexes(unsigned int iii, int *N, int *i) {
for (int j = NUM_OF_PARAMS-1; j >= 0; j--) {
i[j] = iii % N[j];
iii = (iii - i[j]) / N[j];
}
}
```

Рис. 3. Алгоритм расчета индексов виртуальных циклов

Статистические значения ВНП будем хранить в массиве Y_, а расчетные значения для каждого набора параметров в массиве Y. В качестве критериев близости расчетного и статистического временных рядов используем коэффициент близости *U(X, Y) = 1 - E(X, Y)*, где *E(X. Y)* – индекс Тейла [9].

Чем выше *U(X, Y)* (чем ближе он к единице), тем более близки ряды.

$$U(X,Y) = 1 - \frac{\sqrt{\sum_{t=t_0}^{T}\left(X_t - Y_t\right)^2}}{\sqrt{\sum_{t=t_0}^{T} X_t^2} + \sqrt{\sum_{t=t_0}^{T} Y_t^2}}$$

Расчет коэффициента близости производится при помощи функции bliz():

```
float bliz(float *x, float *y, int n) {
     float d1 = 0.0, d2 = 0.0, d3 = 0.0;
     for (int p = 0; p < n; p++) {
            d1 += pow(x[p] - y[p], 2);
            d2 += pow(x[p], 2);
            d3 += pow(y[p], 2);
     }
     return 1.0 - sqrt(d1 / (d2 + d3));
}
```

Рис. 4. Расчет коэффициентов близости

При расчетах используются несколько констант, которые предварительно идентифицированы из статистических данных. Константы, необходимые для расчетов хранятся в массиве fCONST.

Основной вычислительный цикл выглядит следующим образом

```
for (unsigned int j = jStart; j < jEnd; j++) {
     calcIndexes(j, N, i);

     for (int p = 0; p < NUM_OF_PARAMS; p++) {
            par[p] = limits1[p] + i[p] * step[p];
     }

     float fkLast, y[YEARS], Y[YEARS];
     for (int t = 0; t < YEARS; t++) {
```

```
                float fk;
                if (t == 0) {
                        fk = 1;
                } else {
                        fk = fkLast - par[3] * fkLast +
par[4] *
fCONST[3] * y[t-1] / fp(t-1);
                }
                fkLast = fk;

                y[t] = pow(par[0] * pow(fl(t), -par[1]) +
(1 - par[0]) * pow(fk, - par[1]), -par[2] / par[1]);
                Y[t] = y[t] * Y_[0];
        }

        float F = bliz(Y_, Y, YEARS);
        if (/*F < 0.990 && */F > bestF) {
                // найдено решение, лучшее, чем предыдущее
                copyArray(par, bestPar, NUM_OF_PARAMS);
                copyArray(Y, bestY, YEARS);
                bestF = F;
                // как менялось лучшее решение...
                fprintf(perProcLog, "%d %f: ", j, F);
                printFloatArray(par, NUM_OF_PARAMS,
                        perProcLog);

        }
   }

   float fp(int t) {
        return fCONST[0] + (1 - fCONST[0]) * (1 + t) *
exp(- fCONST[1] * t);
   }

   float fl(int t) {
        return exp(fCONST[2] * t);
   }
```

Рис. 5. Основной вычислительный цикл

В каждом вычислительном процессе переменные jStart и jEnd вычисляются автоматически согласно номеру процесса:

```
        int slice = (int) (NN / numproc);
        int jStart = procind * slice;
        int jEnd = (procind + 1) * slice;
```

Рис. 6. Расчет начального и конечного индексов

При завершении расчета каждый процесс, отличный от нулевого, посылает нулевому процессу результаты своих расчетов, свое локальное лучшее решение:

```
   MPI_Send(&bestF, 1, MPI_FLOAT, 0, 77,
MPI_COMM_WORLD);
   MPI_Send(bestPar, NUM_OF_PARAMS, MPI_FLOAT, 0, 77,
                        MPI_COMM_WORLD);
   MPI_Send(bestY, YEARS, MPI_FLOAT, 0, 77,
                        MPI_COMM_WORLD);
```

Рис. 7. Передача расчета от рабочих мастеру

Нулевой процесс, в свою очередь принимает от остальных процессов решения и выбирает из них наилучшее, отображая его на экран.

4. Результаты идентификации

Для идентификации модели (оценки ее параметров) в основном использовались статистические данные, представленные на сайте Федерального агентства по статистике http://www.gks.ru. Часть этих макропоказателей, относящихся к 2000-2006 годам представлена в третьей главе книги [1]. Для сокращения времени расчета, чрезвычайно большого в общем случае из-за проклятия размерности, число перебираемых параметров сокращено до приемлемого уровня за счет эвристической процедуры, частично описанной ниже.

Демографические данные модели были определены непосредственно по статистике рождаемости и смертности. Для определения естественно-биологических параметров смертности B_s и δ_S (см. формулу (3)), предполагаемых общими для всех выделенных страт, воспользуемся известными данными Швеции [4] общей смертности людей в возрасте от 60 до 84 лет. В результате расчетов были получены такие оценки: $B_f = 0.001276 \pm 0.000038$, $\delta_f = 0.1053 \pm 0.0028$, $B_m = 0.01791 \pm 0.00012$, $\delta_m = 0.083 \pm 0.015$ (1/год). Число мальчиков среди новорожденных считаем не зависимым от страты и года: $\mu_t^i = 0.512$. Социальная составляющая коэффициентов смертности предполагалась различной в разных стратах, зависимой от среднего уровня дохода в страте, а максимальный уровень предполагался зависим также от пола (см. формулу (4)). Статистическая информация по стратам России отсутствует, поэтому использовалась информация по регионам и прикидочные расчеты для оценки этих параметров. В результате получены следующие оценки:

Таблица 1. Оценка параметров социальной составляющей смертности РФ

Страты i	$A_f^{+,i}$	$A_m^{+,i}$	λ^i	$d^{0,i}$, тыс.руб./год
1	0.005 ± 0.001	0.006 ± 0.001	0.015 ± 0.001	6000 ± 200
2	0.007 ± 0.001	0.007 ± 0.001	0.011 ± 0.001	3000 ± 100
3	0.007 ± 0.001	0.007 ± 0.001	0.011 ± 0.001	3000 ± 100
4	0.008 ± 0.001	0.010 ± 0.001	0.017 ± 0.001	600 ± 20
5	0.008 ± 0.001	0.010 ± 0.001	0.017 ± 0.001	600 ± 20
6	0.008 ± 0.001	0.011 ± 0.001	0.017 ± 0.001	600 ± 20
7	0.011 ± 0.001	0.016 ± 0.001	0.015 ± 0.001	198 ± 5
8	0.012 ± 0.001	0.017 ± 0.001	0.015 ± 0.001	183 ± 5
9	0.015 ± 0.001	0.019 ± 0.001	0.015 ± 0.001	145 ± 5
10	0.025 ± 0.001	0.025 ± 0.001	0.015 ± 0.001	120 ± 5

Для простоты оценки уровень штрафных санкций полагаем равным 0, а уровни налогообложения всех страт – равными: $m^i = 0$, $n^i = n$. Среднее число лет обучения по стратам задаем экспертной оценкой: $A^1 = A^2 = 20$, $A^3 = A^4 = A^7 = 17$, $A^5 = 15$, $A^6 = A^8 = A^9 = 10$, $A^{10} = 7$.

Оставшиеся пятьдесят неизвестных параметров определяем с помощью параллельных вычислений на кластерном суперкомпьютере: число занятых среди трудоспособных обоего пола χ_f^i, χ_m^i, коэффициенты при оборотном капитале κ^i, коэффициенты при оценке уровня образования ρ^i, доли теневого оборота q^i, $i = 1,...,10$.

Расчеты осуществлялись на кластерных суперкомпьютерах ВЦ РАН, МСЦ РАН и ВятГУ. Время расчета составляло несколько часов при использовании от 16 до 512 процессоров.

Таблица 2. Результаты Оценка параметров модели

Страты i	χ_f^i	χ_m^i	κ^i	o^i	q^i
1	0.5 ± 0.1	0.7 ± 0.1	0.2 ± 0.1	15.0 ± 0.5	0.5 ± 0.1
2	0.5 ± 0.1	0.7 ± 0.1	1.5 ± 0.1	17.0 ± 0.5	0.5 ± 0.1
3	0.4 ± 0.1	0.7 ± 0.1	2.5 ± 0.1	0.5 ± 0.1	0.5 ± 0.1
4	0.7 ± 0.1	0.7 ± 0.1	1.5 ± 0.1	5.0 ± 0.5	0.6 ± 0.1
5	0.7 ± 0.1	0.7 ± 0.1	1.0 ± 0.1	0.5 ± 0.1	0.4 ± 0.1
6	0.6 ± 0.1	0.7 ± 0.1	0.5 ± 0.1	0.5 ± 0.1	0.9 ± 0.1
7	0.7 ± 0.1	0.7 ± 0.1	1.5 ± 0.1	5.0 ± 0.5	0.3 ± 0.1
8	0.7 ± 0.1	0.7 ± 0.1	1.3 ± 0.1	0.5 ± 0.1	0.5 ± 0.1
9	0.2 ± 0.1	0.3 ± 0.1	0.3 ± 0.1	0.5 ± 0.1	0.1 ± 0.1
10	0.5 ± 0.1	0.5 ± 0.1	0.1 ± 0.1	0.5 ± 0.1	1.0 ± 0.1

Указанные в табл. 2 оценки параметров страт следует считать предварительными, поскольку и исходные статистические данные требуют уточнения.

5. Заключение

Итак, в результате параллельных вычислений по оценке параметров на основе верификации валового внутреннего продукта и других макропоказателей экономики России 2000-2009 гг. получен работоспособный вариант математической модели экономики, основанный на описании предложенного расслоения населения России по десяти стратам. Расчеты по такой модели можно использовать для прогноза динамики макроэкономических показателей России, поскольку параметры модели идентифицированы с помощью высокопроизводительных параллельных вычислений на кластерах МСЦ РАН и ВятГУ по исходным статистическим данным. Явная стратификация населения дополнительно позволяет делать выводы о том, как осуществлять крупные экономические преобразования, ответив на вопросы: кто и как их будет делать.

Литература

1. Оленев Н.Н., Печенкин Р.В., Чернецов А.М. Параллельное программирование в MATLAB и его приложения. -М.: ВЦ РАН. 2007. -120 с. http://www.ccas.ru/mmes/distcompbook.pdf
2. Кантор М. Бунт Сытых// "Новая Газета", № 127, 16 ноября 2009 http://www.novayagazeta.ru/data/2009/127/16.html
3. Фетинина А.И. Высокопроизводительные вычисления при моделировании стратификации в региональной экономике // Наст. сб. -С.
4. Можжерина Е.Ю., Оленев Н.Н. К построению экономико-демографической модели с производственными фондами, дифференцированными по моментам создания // Математическое моделирование развивающейся экономики и экологии. ЭКОМОД-2009. Сборник трудов. -Киров: ВятГУ, 2009. -С.229-241.
5. Павловский Ю.Н., Белотелов Н.В., Бродский Ю.И., Оленев Н.Н. Опыт имитационного моделирования при анализе социально-экономических явлений . М.: МЗ Пресс. 2005. 136 с.
6. Величковский Б.Т. Жизнеспособность нации. Роль социального стресса и генетических особенностей популяции в развитии демографического кризиса и изменении состояния здоровья населения России. М.: РАМН, 2009. 176 с.
7. Кощеев А.В., Оленев Н.Н. «Модель взаимодействия региональных экономических систем. Учебное пособие». Киров: Изд-во ВятГУ, 2009. –40 с.

8. Оленев Н.Н., Стародубцева В.С. Исследование влияния теневого оборота на социально-экономическое положение в Республике Алтай // Региональная экономика: теория и практика. 2008. № 11 (68). -С.32-37.
9. Оленев Н.Н., Фетинина А.И. Моделирование экономики Кировской области с применением технологий параллельного программирования// Научно-технический вестник СПбГУ ИТМО. Январь-февраль 2010. № 1(65). С. 108-113.

Использование шаблонного метапрограммирования при реализации параллельных эвристических алгоритмов оптимизации

О.И. Булычов

В статье рассматривается подход к разработке параллельных эвристических алгоритмов оптимизации, реализованный в библиотеке параллельных метаэвристик HeO. Подход основан на использовании шаблонов проектирования, что позволяет получать универсальные и гибкие алгоритмы с возможностью гибридизации. Приводятся примеры реализации параллельного генетического алгоритма и алгоритма имитационной нормализации, и их гибридов.

1. Введение

Оптимизационные задачи возникают в различных областях науки и техники, и для их решения разработан богатый арсенал методов. Однако нахождение точного решения многих задач на практике оказывается невозможным. Это может быть вызвано разными причинами: сложностью решаемой задачи, погрешностями входных данных, временными и аппаратными ограничениями и др. В связи с этим актуальным остается вопрос разработки эффективных приближенных методов решения таких задач.

Благодаря интенсивному развитию компьютерной техники вот уже несколько десятилетий для нахождения приближенного решения многих трудных оптимизационных задач (прежде всего задач дискретной оптимизации) с успехом применяются метаэвристические алгоритмы (см., например, [1]). Популярность данных алгоритмов обусловлена целым рядом факторов, из которых можно отметить следующие два:

- простота и ясность концепций, лежащих в их основе;
- высокий уровень абстракции.

Первый фактор обуславливает относительную простоту реализации метаэвристических алгоритмов на различных программно-аппаратных платформах, а второй позволяет применять данные алгоритмы к широкому кругу оптимизационных задач (как дискретных, так и непрерывных).

Бурное развитие параллельных вычислительных технологий в последние годы способствовало возрождению интереса к метаэвристическим алгоритмам, что обусловлено возможностью их эффективного распараллеливания. Используя различные модели параллельных вычислений в метаэвристических алгоритмах, можно либо улучшать качество получаемых решений, либо ускорять процесс их поиска. На сегодняшний день разработано большое количество программных средств, в которых реализованы те или иные параллельные метаэвристические алгоритмы для решения конкретных задач. Одной из тенденций последнего десятилетия является создание универсальных библиотек[1] параллельных метаэвристических алгоритмов, которые можно применять для решения широкого круга оптимизационных задач.

Кроме того, в последние годы большое внимание уделяется гибридизации метаэвристик, что позволяет на основе базовых алгоритмов получать новые, более эффективные методы оптимизации. Вопросы гибридизации метаэвристических алгоритмов в разных программных продуктах решаются по-разному. Учитывая большое разнообразие метаэвристических алгоритмов, универсального метода их гибридизации просто не существует, однако можно выделить некоторые общие подходы к гибридизации, применяемые на практике. Хороший обзор методов гибридизации метаэвристик приведен в [2].

В настоящей статье рассматриваются практические вопросы программной реализации и гибридизации параллельных метаэвристических алгоритмов в библиотеке HeO, разработанной с использованием современных технологий программирования, метапрограммирования и па-

[1] В англоязычной литературе часто используется термин *frameworks*.

раллельных вычислений на языке C++. Подробное описание библиотеки можно найти в [3]. В следующем разделе будут приведены наиболее важные сведения об этой библиотеке.

2. Библиотека HeO

Библиотека HeO (**He**uristic **O**ptimization) является проектом с открытым исходным кодом и распространяется на основе лицензии MIT. Цель проекта — обеспечить исследователей современными и простыми в использовании средствами для решения широкого круга оптимизационных задач.

Ключевыми особенностями библиотеки являются:

- кроссплатформенность, поддержка архитектур x86 и x64;
- реализация методов оптимизации в виде проблемно-независимых алгоритмических каркасов;
- реализация каждого метода оптимизации с использованием двух технологий параллельного программирования: MPI и OpenMP;
- использование оригинальной обертки (wrapper) для функций MPI;
- наличие мастеров (wizards), облегчающих создание пользовательских проектов;
- прозрачность параллелизма для пользователей библиотеки.

Официальная страница проекта располагается по адресу: http://www.code.google.com/p/heo.[1]

Библиотека представляет собой набор классов, которые можно разделить на две большие группы:

- основные классы;
- вспомогательные классы.

Основные классы участвуют в непосредственной реализации того или иного метода оптимизации. Ядро основных классов библиотеки составляют так называемые *решатели* (solvers). Каждый решатель реализует определенный метод оптимизации в соответствии с определенной технологией параллельного программирования (MPI или OpenMP).

Решатели выполнены в виде шаблонов языка C++ в соответствии с идеологией алгоритмических каркасов. Это означает, что в них:

- реализована только проблемно-независимая часть алгоритма;
- скрыты все аспекты параллельной реализации.

Эти особенности позволяют применять решатели к широкому кругу задач и делают параллелизм прозрачным для пользователя библиотеки. Для использования решателей пользователем должны быть реализованы проблемно-зависимые классы, описывающие конкретную задачу оптимизации и ее решения. Кроме того, в проблемно-зависимых классах обязательно должны быть реализованы методы, специфические для того или иного алгоритма оптимизации.

Вспомогательные классы библиотеки выполняют всевозможные служебные функции (работа с конфигурационными файлами, запуск решателей, передача данных по сети, генерация псевдослучайных чисел и т.п.).

Для демонстрации возможностей библиотеки разработаны несколько проектов, в которых реализованы проблемно-зависимые классы для нескольких задач дискретной и непрерывной оптимизации:

- ONE-MAX — максимизация числа единиц в двоичной строке (тестовая задача для метаэвристических алгоритмов);
- MAX-SAT — задача максимальной выполнимости;
- GridMin — задача минимизации числа покрывающих блоков в #-отношениях недетерминированных конечных автоматов;
- Rastrigin — минимизация функции Растригина;
- Rosenbrock — минимизация функции Розенброка;
- ODE1IVP — численное решение задачи Коши для ОДУ первого порядка.

[1] В настоящий момент для скачивания (в разделе Downloads или через SVN-клиент) доступна версия 1.1 библиотеки.

3. Реализация параллельных метаэвристических алгоритмов и их гибридизация с использованием технологии шаблонного проектирования

В настоящий момент в библиотеке HeO реализованы два метаэвристических метода оптимизации: генетический алгоритм (GA — Genetic Algorithm) и метод имитационной нормализации или имитации отжига (SA — Simulated Annealing), а также их гибрид (GA+SA). Описания последовательных версий методов GA и SA можно найти, например, в [4]. Ниже будет рассмотрен использованный в библиотеке подход к параллельной реализации этих методов, основанный на технологии шаблонного проектирования. Подробное изложение методов шаблонного проектирования приведено в [5].

3.1 Реализация параллельных алгоритмов оптимизации

Как уже было отмечено выше, каждый из методов оптимизации реализован в библиотеке с помощью двух технологий параллельного программирования (MPI и OpenMP). В текущей версии библиотеки основной моделью параллельных вычислений является *мультистартовая* модель[1]. В этой модели алгоритм работает в несколько потоков, независимо решающих задачу с периодической кооперацией.

На рис. 1. представлен псевдокод потоков параллельного генетического алгоритма.

```
сгенерировать начальную популяцию хромосом
вычислить функцию стоимости для каждой хромосомы
while not(выполнено условие остановки) do
  отобрать родительские хромосомы
  выполнить кроссовер (получить потомков)
  применить мутацию к потомкам
  вычислить функцию стоимости для потомков
  сформировать новую популяцию
  if (условие кооперации) then выполнить кооперацию
end do
Выход: лучшие хромосомы (решение задачи)
```

Рис. 1. Псевдокод потоков параллельного генетического алгоритма

Кооперация в параллельном GA заключается в периодическом обмене выбранными с помощью некоторой процедуры решениями и фактически является аналогом миграции в островной модели GA.

Псевдокод потоков параллельного метода имитации отжига приведен на рис. 2.

```
сгенерировать начальное решение
вычислить функцию стоимости для начального решения
выбрать начальное значение температуры T
установить счетчик начала итераций k равным 0
while not(выполнено условие остановки) do
  if (условие кооперации) then выполнить кооперацию
  выбрать случайное решение из окрестности текущего
  вычислить его стоимость
  сделать новое решение текущим с вероятностью P(T)
  увеличить счетчик итераций k
  изменить температуру T
end do
Выход: текущее решение
```

Рис. 2. Псевдокод потоков параллельного алгоритма имитации отжига

Кооперация в параллельном SA представляет собой периодический обмен генерируемыми случайными решениями (ходами).

[1] Для генетического алгоритма с использованием технологии OpenMP реализована также модель глобальной популяции.

В библиотеке реализованы два режима кооперации: асинхронный и синхронный, схемы которых приведены на рис. 3.

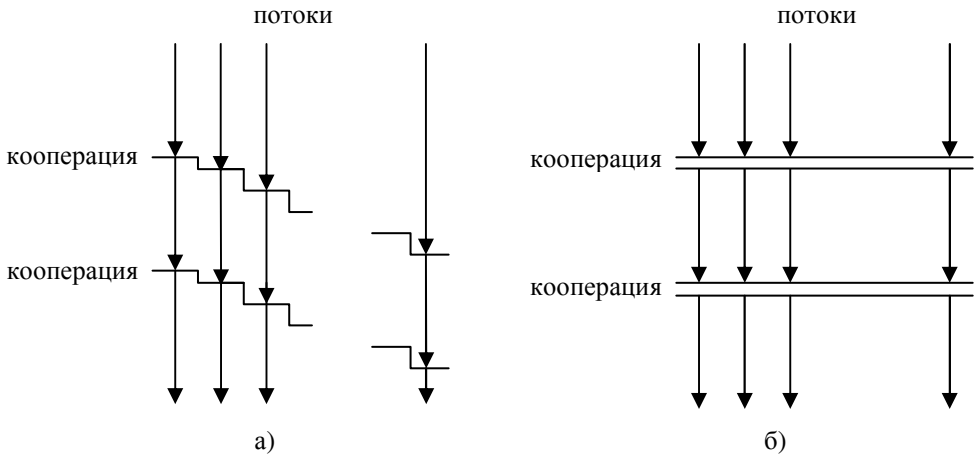


Рис. 3. Схема асинхронной (а) и синхронной (б) кооперации потоков в мультистартовой модели

При синхронной кооперации осуществляется блокирование выполнения потоков и обмен данными между ними через фиксированное число итераций. Асинхронная кооперация инициируется первым потоком через фиксированные промежутки времени и выполняется без блокировки потоков по мере их готовности к обмену данными. Использование синхронной кооперации может приводить к большой нагрузке на сеть в моменты кооперации. Асинхронный режим позволяет эффективно решать задачу на неоднородных кластерах и снизить нагрузку на сеть.

При использовании кооперации возникает проблема *сериализации*, т.е. перевода структур данных программы в последовательность битов и обратно. В большинстве существующих библиотек параллельной оптимизации загрузка, сохранение и передача по сети каждого поля класса производится индивидуально. Это приводит к тому, что при внесении изменений в структуру класса приходится вручную изменять программный код в каждом месте, где требуется сериализация.

С целью упрощения этих операций в библиотеке HeO используются приемы шаблонного метапрограммирования, позволяющие реализовать элементы технологии Reflection (Отражение) для строго типизированного доступа к полям классов. Реализованный алгоритм сериализации базируется на шаблоне проектирования Visitor (Посетитель), позволяющем определять новые операции над классами библиотеки, не изменяя исходный код классов. Для его использования достаточно определить метод `accept()` в требуемых классах, и перечислить в нем необходимые поля класса (см. рис. 4).

```
int                         member1_;
std::vector<float>          member2_;
ref_ptr<some_interface> member3_;
...
template<class V> void accept(V& v)
{
    v(member1_, "member1");
    v(member2_, "member2");
    v(member3_, "member3");
    ...
};
```

Рис. 4. Пример определения метода accept()

В случае если поле само является структурой или классом, сериализация осуществляется рекурсивно. Для некоторых классов, в частности для `std::string`, `std::vector` и классов "умных" указателей, определены особые служебные классы, осуществляющие их сериализацию.

К сожалению, в языке C++ нет встроенного автоматического механизма Reflection. При изменении структуры класса, придется вносить изменение и в функцию `accept()`.

Кооперация потоков в алгоритмах GA и SA реализована с использованием шаблона Strategy (Стратегия). Данный шаблон обеспечивает различные варианты кооперации без изменения исходного кода алгоритмов-клиентов. В настоящий момент реализована стратегия кооперации с обменом данными между потоками по кольцу, однако она может быть заменена на любую другую.

3.2 Гибридизация параллельных алгоритмов

Для разных типов метаэвристических алгоритмов могут быть предложены разные способы гибридизации. В частности, для алгоритмов GA и SA, наиболее популярными способами являются интеграционная (integrative) и кооперационная (cooperative) гибридизация. При интеграционной гибридизации один из алгоритмов является частью другого, например, SA может рассматриваться как один из генетических операторов, выполняемых в основном цикле GA. При кооперационной гибридизации алгоритмы обмениваются друг с другом решениями.

Наибольшие трудности возникают при практической реализации первой стратегии. Как правило, в существующих реализациях гибридного метода GA+SA жестко фиксируется структура алгоритма, а для ее модификации приходится вносить изменения в исходный код программы. Для решения этой проблемы в библиотеке HeO используется шаблон проектирования Factory (Фабрика), позволяющий сделать код создания объектов более универсальным, не привязываясь к конкретным классам, а оперируя лишь общим интерфейсом.

Все операторы гибридного алгоритма наследуются от абстрактного интерфейса `abstract_operator`. В исходном коде гибрида и фабрики имена классов-операторов нигде не упоминаются, что упрощает добавление новых операторов. Каждый оператор регистрируется на фабрике с помощью вспомогательной функции. На рис. 5 показаны отношения между гибридом, фабрикой и операторами.

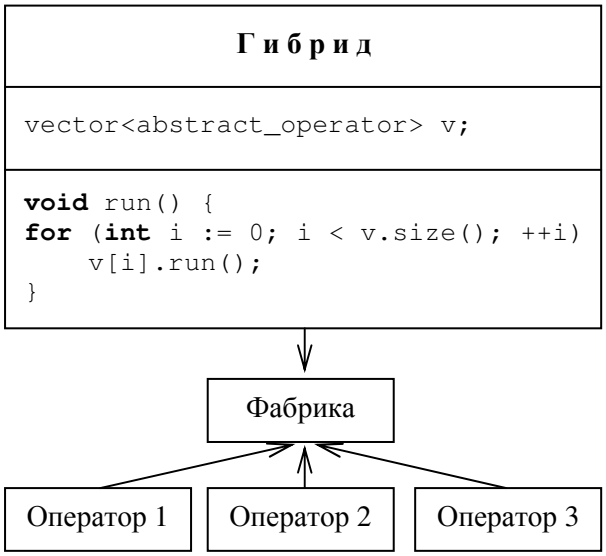


Рис. 5. Диаграмма отношений между гибридом, фабрикой и операторами

Одним из недостатков фабрики считается трудность восстановления *класса* создаваемого ею продукта по его абстрактному интерфейсу, с целью выполнения операции, зависящей от класса (к примеру, сериализации). Реализованный механизм Reflection позволяет эффективно обойти этот недостаток.

Параметры решателей в библиотеке HeO хранятся в конфигурационных файлах в формате xml. При выполнении кода, изображенного на рис. 6, в решателе `solver` будет создан вектор операторов, заданных в xml-файле конфигурации.

```
xml::in in(env.property_tree_);
in >> solver;
```

Рис. 6. Пример инициализации класса

На рис. 7 и 8 приведены фрагменты xml-файлов, задающих различные векторы операторов для генетического алгоритма.

```
<operator>
  <crossover .../>
  <sa .../>
</operator>
```

Рис. 7. Фрагмент конфигурационного файла

В первом случае будет создан вектор операторов [`crossover`, `sa`], а во втором [`sa`, `crossover`, `mutation`], которые будут последовательно применяться на каждой итерации алгоритма.

```
<operator>
  <sa .../>
  <crossover .../>
  <mutation .../>
</operator>
```

Рис. 8. Фрагмент конфигурационного файла

Аналогично с помощью фабрики осуществляется выбор операторов отбора родительских и дочерних хромосом.

Поскольку для правильной работы программы требуется лишь один экземпляр конкретной фабрики, то для ее конструирования используется шаблонный класс Singleton (Одиночка).

При обмене данными между решателями, реализующими различные алгоритмы, может потребоваться необходимость преобразования проблемно-зависимых классов, поскольку представления данных в разных алгоритмах могут быть различными. Для решения этой проблемы в библиотеке HeO используется шаблон проектирования Adapter (Адаптер), позволяющий не принимать во внимание различия между проблемно-зависимыми классами разных алгоритмов.

Такой подход полностью избавляет пользователя от изменения исходного кода гибридного метода и позволяет получать гибридные алгоритмы, наиболее подходящие для решения конкретной задачи.

4. Примеры

В качестве практических примеров использования параллельных эвристических алгоритмов оптимизации рассмотрим две задачи: задачу максимальной выполнимости и задачу поиска минимума функции Розенброка.

4.1 Задача максимальной выполнимости

Задача максимальной выполнимости (MAX-SAT) может кратко быть сформулирована следующим образом: дана конъюнктивная нормальная форма (КНФ) функции N переменных из M клауз. Требуется найти двоичный набор, при котором в этой функции будет выполнено наибольшее число клауз. Клаузы могут быть как фиксированной, так и переменной длины. Описание различных формулировок задач MAX-SAT можно найти, например, в [4].

В данном разделе будут приведены результаты тестирования для задачи ndhf_xits_22_SAT из набора SAT 2009. Параметры задачи: число переменных — 4692, число клауз — 582514. Данная задача является выполнимой, поэтому максимальное число выполнимых клауз совпадает с общим числом и равно 582514.

Задача решалась на системе с общей памятью следующей конфигурации: процессор — Intel Core 2 Quad Q6600 @ 2.4 ГГц; ОЗУ — 4 Гб; операционная система — MS Windows XP

Professional SP 3. Использовалась OpenMP-версия генетического алгоритма, в которой реализована модель глобальной популяции.

Задача решалась с числом потоков $k = 1, 2, 3, 4$. При каждом k выполнялось 10 запусков программы. Для каждого запуска запоминалось время выполнения и число невыполненных клауз. Затем результаты усреднялись.

Основные настройки алгоритма GA приведены на рис. 9.

```
<ga population_size="200"
    offspring_size="100"
    only_offspring="false"
    migration_size="3">
  <parent_selection>
    <tournament tournament_size="16"/>
  </parent_selection>
  <offspring_selection>
    <tournament tournament_size="16"/>
  </offspring_selection>
  <operator>
      <crossover probability="0.7"/>
      <mutation probability="0.002"
                fadeout="false"/>
  </operator>
  <stop_condition max_step="500"/>
  <coop async_mode="true"
        cooperation_rate="25"/>
</ga>
```

Рис. 9. Настройки GA

Результаты решения задачи приведены в таблице 1. Полученные результаты являются приемлемыми для универсальных алгоритмов.

Таблица 1. Результаты решения. N — число потоков, ε — число невыполненных клауз, Δ — процент невыполненных клауз, t — время решения (сек.)

N	**ε**	**Δ**	***t***
1	1100	0,189	143
2	1094	0,188	81
3	1149	0,197	58
4	1128	0,194	44

График ускорения вычислений приведен на рис. 10.

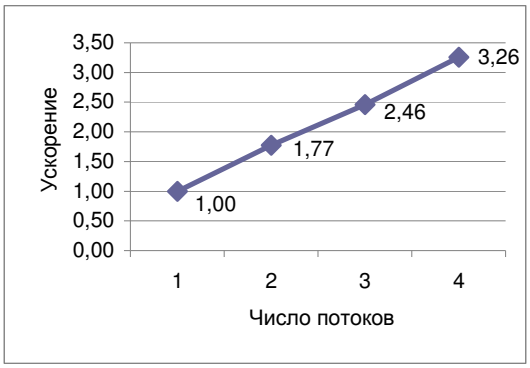


Рис. 10. График ускорения вычислений

4.2 Задача поиска минимума функции Розенброка

Рассмотрим задачу нахождения глобального минимума функции Розенброка n переменных, которая задается следующим образом:

$$f(x)=\sum_{i=1}^{n-1}\left(100\left(x_{i+1}-x_i^2\right)^2+\left(x_i-1\right)^2\right)$$

где $x=(x_1,x_2,...,x_n)$, $x_i\in[-2.048;\,2.048]$. График функции Розенброка для случая двух переменных приведен на рис. 11.

Для этой функции характерно наличие плоского плато, что затрудняет нахождение ее глобального оптимума с помощью генетических алгоритмов (затрудняет сходимость). Точным решением данной задачи является точка $x_0=(1,1,...,1)$, значение функции в которой равно 0.

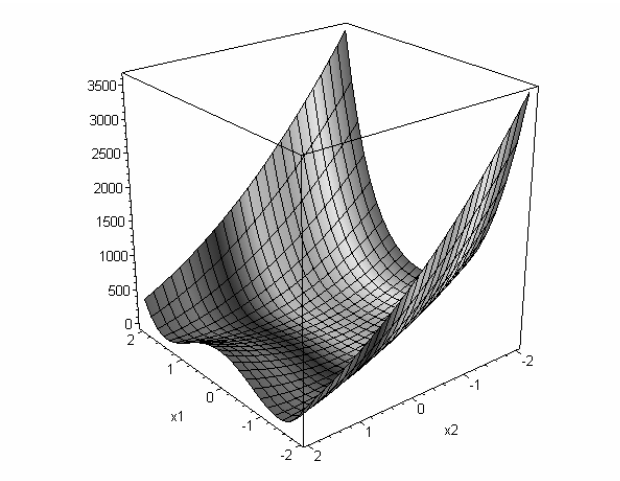


Рис. 11. График функции Розенброка

Данная задача решалась на кластере Томского политехнического университета с помощью MPI-версий обычного генетического алгоритма (GA) и гибрида (GA+SA). В этих версиях реализована островная модель GA. Решение задачи осуществлялось для $n=10$ и $n=50$ с числом потоков 2^k ($k=0,\ 1,\ 2,\ ...,\ 6$). При каждом k выполнялось 10 запусков программы. Фиксировалось время выполнения и стоимость получаемого решения. Для каждого запуска вычислялась невязка найденного решения по формуле $r(x)=\sqrt{\sum_{i=1}^{n}(1-x_i)^2}$. Полученные результаты усреднялись.

Основные настройки алгоритмов GA и GA+SA приведены на рис. 12 и 13.

```
<ga population_size="100"
    offspring_size="200"
    only_offspring="false"
    migration_size="3">
  <parent_selection>
    <roulette/>
  </parent_selection>
  <offspring_selection>
    <tournament tournament_size="5"/>
  </offspring_selection>
  <operator>
      <crossover probability="0.7"/>
      <mutation probability="0.002"
                fadeout="false"/>
  </operator>
  <stop_condition max_step="100000"/>
  <coop async_mode="false"
        cooperation_rate="25"/>
```

```
</ga>
```

Рис. 12. Настройки GA

Как видно из рис. 13, в xml-файле гибридного алгоритма оператор мутации заменен на оператор SA. Операторы `crossover` и `sa` можно было бы переставить местами, и добавить к ним оператор `mutation`, получив при этом новый гибридный алгоритм, не изменяя программного кода.

```
<ga population_size="50"
    offspring_size="100"
    only_offspring="false"
    migration_size="3">
  <parent_selection>
    <roulette/>
  </parent_selection>
  <offspring_selection>
    <tournament tournament_size="3"/>
  </offspring_selection>
  <operator>
    <crossover probability="0.7"/>
    <sa probability="0.03"
        move_probability="0.02"
        initial_temperature="1.5"
        cooling_rate="0.994"
        heating_rate="1.2"
        isotherm_moves="10"
        update_policy="0">
        <stop_condition max_step="100000"/>
    </sa>
  </operator>
  <stop_condition max_step="40"/>
  <coop async_mode="false"
        cooperation_rate="25"/>
</ga>
```

Рис. 13. Настройки GA+SA

В табл. 2 приведено среднее время решения задачи обоими методами в зависимости от числа потоков и n.

Таблица 2. Среднее время решения, сек.

Число потоков	**GA**		**GA+SA**	
	***n* = 10**	***n* = 50**	***n* = 10**	***n* = 50**
1	6,555023	18,92139	2,402968	9,61315
2	6,606100	18,92139	2,577363	10,79885
4	7,029273	18,97877	2,623410	11,54005
8	6,704668	19,63942	2,738245	11,54005
16	7,873530	19,15085	2,890340	11,34775
32	6,843558	20,72200	2,890340	11,66562
64	8,593803	19,35119	3,112738	13,45320

Так как число особей в популяции для каждого потока оставалось неизменным (равным 50), то происходит не ускорение вычислений, а повышение качества получаемого решения. На рис. 14 и 15 приведены графики зависимости невязки решения от числа потоков для каждого метода при $n = 10$ и $n = 50$.

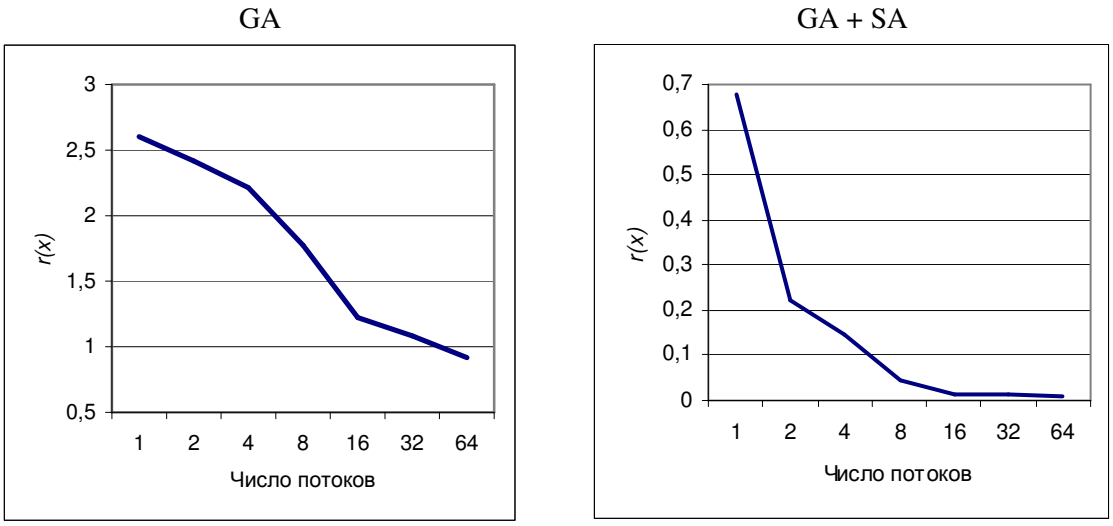


Рис. 14. Графики зависимости невязки решения от числа потоков при $n = 10$

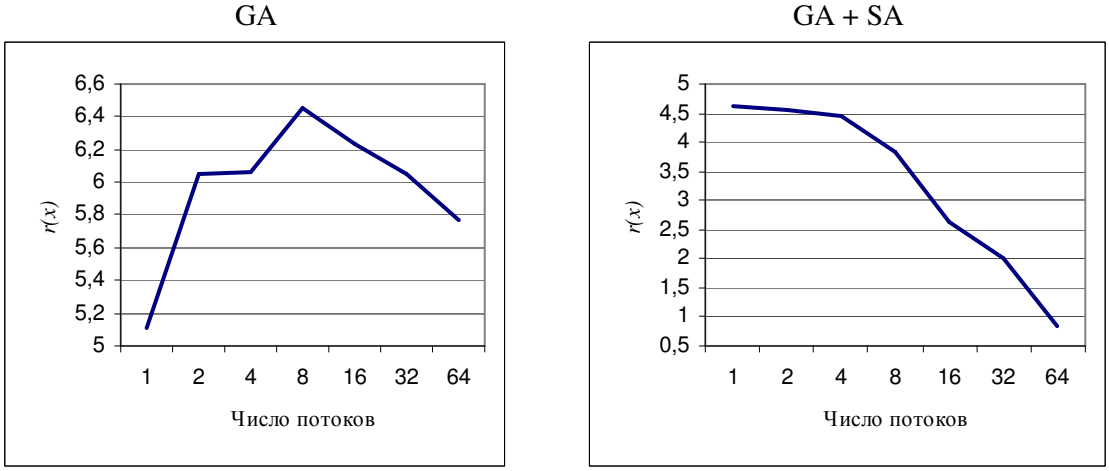


Рис. 15. Графики зависимости невязки решения от числа потоков при $n = 50$

Из приведенных таблицы и графиков видно, что гибридный алгоритм позволяет за меньшее время получить для данной задачи гораздо более точное решение.

5. Заключение

Рассмотренный в работе подход к реализации параллельных эвристических алгоритмов и их гибридов, основанный на использовании технологии шаблонного проектирования, отличается универсальностью и высокой гибкостью и может быть использован при разработке параллельных алгоритмов других типов. Данный подход позволяет:

- повысить возможность повторного использования исходного кода;
- повысить наглядность программы;
- облегчить модификацию программы;
- упростить создание гибридных алгоритмов.

Однако использование такого подхода требует высокой квалификации программиста и знакомство с основами технологии шаблонного проектирования.

В настоящее время в рамках библиотеки HeO планируется реализация уже имеющихся методов с использованием технологии параллельного программирования Intel Threading Building Blocks (TBB).

Автор выражает благодарность руководству и сотрудникам суперкомпьютерного кластера «СКИФ-политех» Томского политехнического университета.

Литература

1. Handbook of metaheuristics / Ed. by F. Glover, G. A. Kohenberger. – Kluwer Academic Publishers, 2003.
2. Raidl G.R., A unified view on hybrid metaheuristics // Proceedings of Hybrid Metaheuristics, Third International Workshop, volume 4030 of LNCS / Ed. by F.Almeida et al. – Springer, 2006. – P. 1-12.
3. Цыганов А.В., Булычов О.И. НеО: библиотека метаэвристик для задач дискретной оптимизации // Программные продукты и системы. – 2009. – № 4. – С. 148-151.
4. Громкович Ю. Теоретическая информатика, 3 изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2009. – 360 с.
5. Влиссидес Дж., Джонсон Р., Гамма Э., Хелм Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования, – СПб.: Питер, 2007 г.

Комбинированная MPI+threads параллельная реализация метода блоков для моделирования тепловых процессов в структурно-неоднородных средах

Д.Б. Волков-Богородский, Г.Б. Сушко, С.А. Харченко

В работе разрабатываются комбинированные MPI+threads параллельные алгоритмы аппроксимации решений нестационарного уравнения теплопроводности с фазовыми переходами на основе аналитического метода блоков. Метод блоков основан на приближении решения краевой задачи специальными функциями, являющимися фундаментальными решениями уравнения Гельмгольца. При этом возникает система линейных алгебраических уравнений с блочно-разреженной структурой и плотными подматрицами. Интенсивные вычисления с плотными подматрицами распараллеливаются на основе потоков вычислений с использованием общей памяти. Относительно независимые вычисления с блочно разреженной структурой распараллеливаются по распределенной памяти с помощью MPI. Предлагаемый комбинированный подход к организации параллельных вычислений позволяет эффективно использовать неоднородную структуру организации памяти в современных кластерных системах.

1. Введение

В настоящей работе развиваются численно-аналитические методы для моделирования теплофизических процессов в структурно-неоднородных средах. Эти методы основаны на аппроксимации решений соответствующих краевых задач механики сплошных сред рядами по специальным системам функций, являющимся фундаментальными решениями уравнения Гельмгольца, и обладают высокой степенью точности. В структуре аппроксимирующих функций, для построения которых предложен конструктивный метод квазиразделения переменных, учитываются аналитические особенности геометрии области, что обуславливает эффективность аппроксимации. Метод построения фундаментальной системы функций позволяет выявить аналитические свойства используемых систем, в частности получить простые рекуррентные соотношения между функциями и их производными, что эффективно используется в алгоритмах вычисления необходимых при решении задач характеристик (температура, теплопотоки, внутренняя энергия).

В качестве механизма аппроксимации используется несколько обобщений метода наименьших квадратов на многоблочные структуры; в этом случае область разбивается на подобласти более простой структуры, в каждой из которых используется своя аппроксимирующая система функций. Сам по себе метод наименьших квадратов является эффективным, к сожалению, только для областей простой геометрии, однако его обобщение на многоблочные структуры позволяет существенно расширить область применимости этого метода. Для задач теории упругости и акустики многоблочный вариант метода наименьших квадратов был предложен в работе [1], и получил дальнейшее развитие в [2-5]. Метод позволяет контролировать точность аппроксимации по невязке между решениями на границе блоков.

Рассматриваются регулярные среды с периодической структурой и с тремя основными видами включений, имеющими форму слоя, цилиндра или шара , см. рис. 1-3. Для расчета тепловых процессов в таких средах применяется или асимптотический метод усреднения Бахвалова [6], или непосредственное моделирование, когда с каждым отдельным включением связывается свой отдельный блок. Могут быть рассмотрены также среды и с нерегулярной структурой.

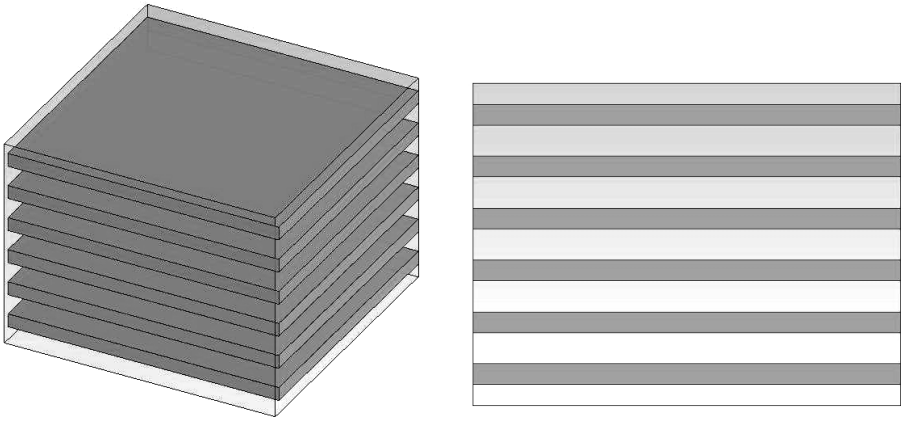


Рис. 1. Примеры регулярной структуры – слои

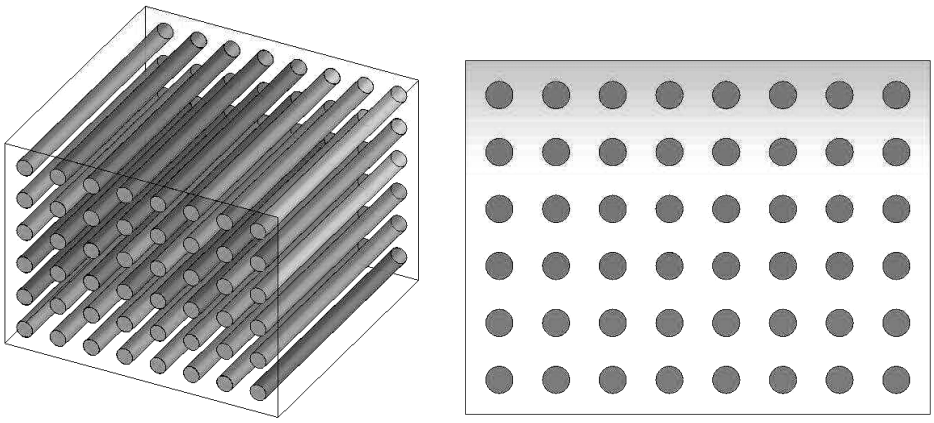


Рис. 2. Примеры регулярной структуры – цилиндры

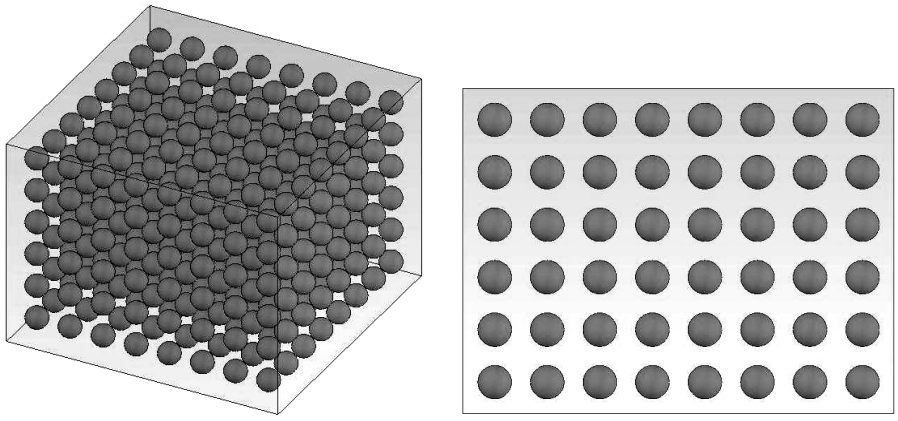


Рис. 3. Примеры регулярной структуры – сферические включения

Метод позволяет получать решение с высокой степенью точности и в аналитическом виде, однако требует больших вычислительных затрат. Поэтому актуальной задачей является разработка параллельной версии блочного метода наименьших квадратов.

Основным моментом в блочном методе является необходимость решения системы линейных алгебраических уравнений, структура которой соответствует структуре блочного разбиения исходной области: система линейных уравнений имеет блочно-разреженную структуру с плотными блоками-подматрицами. Естественным образом здесь возникает два уровня вычислений: интенсивные вычисления с плотными подматрицами на уровне общей памяти и относительно независимые вычисления между блоками на уровне распределенной памяти. В работе разрабатывается соответствующая реализация метода решения блочной системы уравнений.

2. Аппроксимация решений нестационарного уравнения теплопроводности

2.1 Фундаментальная система решений уравнения Гельмгольца

Аппроксимация тепловых процессов в структурно-неоднородных средах основана на представлении общего решения исходного уравнения в каждой из фаз материала в виде суперпозиции частных решений следующего вида:

$$\theta(P,t) = \Phi(P)\, e^{-\kappa^2 \text{Fo}}, \qquad \nabla^2 \Phi(P) + \kappa^2 \Phi(P) = 0, \qquad P = (x,\, y,\, z) \tag{1}$$

где $\text{Fo} = \dfrac{\lambda}{C} t$ – безразмерное время Фурье, λ –теплопроводность, C – теплоемкость, κ – постоянная, определяющая скорость установления температуры; представление (1) для частных решений уравнения теплопроводности согласуется с известными аналитическими представлениями,.

Таким образом, аппроксимация основана на фундаментальных решениях уравнения Гельмгольца. Решения вида (1) характерны для тепловых процессов, составляя в частности ядро интегрального преобразования Лапласа по временной координате, являющегося эффективным аналитическим инструментом для получения решений уравнения теплопроводности [7,8].

На поверхности включения для частного решения (1) выполняются условия сопряжения – непрерывность температуры и нормальной составляющей теплового потока $\chi(P,t) = -\lambda \nabla \theta(P,t)$; для функции Φ эти условия приобретают следующий вид:

$$\left[\frac{\lambda \kappa^2}{C}\right] = 0, \quad [\Phi] = 0, \quad \left[\lambda \frac{\partial \Phi}{\partial n}\right] = 0. \tag{2}$$

Таким образом, мы рассматриваем среды с естественными условиями сопряжения для тепловой составляющей решения на поверхности включений.

Необходимая для аппроксимации система функций строится в настоящей работе при помощи метода квазиразделения переменных [5] (комплексных или вещественных). Таким путем строится несколько стандартных классов фундаментальных решений уравнения Гельмгольца, позволяющих получить эффективную аппроксимацию решения в средах, содержащих включения в виде слоя, цилиндра или шара.

Метод квазиразделения переменных позволяет построить теорию специальных функций – решений уравнения Гельмгольца, включающую детальное рассмотрение аналитических свойств используемых классов функций (рекуррентные соотношения при дифференцировании и при вычислении специальных функций). Эти свойства эффективно используются при численной реализации методов решения краевых задач.

Система аппроксимирующих функций определяется с помощью формального ряда на основе анализа уравнения Гельмгольца в переменных x, y, z или $w = x + iy$, $\overline{w} = x - iy$ и z:

$$\Phi(P) = \Phi(x,\, y,\, z) = \sum_p \phi_p(x,\, y)\, U_p(z), \tag{3}$$

$$\nabla^2 \Phi + \kappa^2 \Phi = \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} + \frac{\partial^2 \Phi}{\partial z^2} + \kappa^2 \Phi = 0. \tag{4}$$

Между функциями $\phi_p(x,y)$ и $U_p(z)$ устанавливаются рекуррентные соотношения так, чтобы ряд (3) в целом удовлетворял уравнению (4). При этом применяются разные подходы, при которых по направлению z выполняется или интегрирование, или дифференцирование (а по оставшимся направлениям – наоборот). В зависимости от этого получаются разные системы функций, наиболее подходящие для определенной геометрии включений.

Подход, при котором по направлению z выполняется интегрирование, соответствует средам со слоистой структурой, для них условия сопряжения (2) обеспечиваются в замкнутом конечном виде:

$$U_p'' + \kappa^2 U_p = U_{p-1}, \qquad U_0'' + \kappa^2 U_0 = 0, \tag{5}$$

$$\nabla^2\phi_p + \phi_{p+1} = 0\,, \qquad \phi_0 - \text{любая}\,. \tag{6}$$

Подход, при котором по направлению z выполняется дифференцирование:

$$\nabla^2\phi_p + \kappa^2\phi_p + \phi_{p-1} = 0\,, \qquad \nabla^2\phi_0 + \kappa^2\phi_0 = 0\,, \tag{7}$$

$$U_p'' = U_{p+1}\,, \qquad U_0 - \text{любая}\,; \tag{8}$$

соответствует средам с цилиндрической и сферической структурой, поскольку здесь получаются системы функций, для которых условия сопряжения (2) выполняются аналитически на поверхности цилиндра или шара.

Нетрудно убедиться, что при выполнении условий (5) – (6) или (7) – (8) ряд (3) автоматически удовлетворяет уравнению (4). Система рекуррентных соотношений (5), (6) определяет процедуру продолжения решений одномерного уравнения Гельмгольца (плоской волны) по заданному закону в ортогональной плоскости, определяемому функцией ϕ_0. Соответственно система рекуррентных соотношений (7), (8) определяет процедуру продолжения некоторых решений ϕ_0 двумерного уравнения Гельмгольца во все пространство по заданному закону, определяемому функцией $U_0(z)$.

Отметим, что система (5) разрешается в аналитическом виде при помощи простых рекуррентных соотношений. А система (7) может быть решена явно в комплексных координатах w и $\overline{w}$ для функций Бесселя [9], что предопределяет построение необходимых для аппроксимации классов функций в средах с включениями цилиндрической и сферической формы:

$$U_p = \frac{z\,\mathcal{U}_{p-1}}{2p}\,, \quad \mathcal{U}_p = \frac{(2p-1)\,\mathcal{U}_{p-1} - zU_{p-1}}{2p\kappa^2}\,, \quad U_0 = \cos(\kappa z)\,, \quad \mathcal{U}_0 = \sin(\kappa z)/\kappa\,, \tag{9}$$

$$\phi_p(w,\overline{w}) = \frac{\Gamma(\mu+1)}{(\kappa/2)^{\mu/2}}\,\frac{(-1)^p\, w^{p+\mu/2}\,\overline{w}^{p-\mu/2}}{4^p\, p!\,(\mu+1)_p}\,J_{\mu+p}(\kappa r)\,, \quad r = \sqrt{w\,\overline{w}}\,. \tag{10}$$

С помощью (10) мы можем выразить окончательное решение (3) для случая (7), (8) в виде бесконечного ряда, который определяется комплексной функцией $\psi_0 = w^\mu$, являющейся главным членом в представлении (10) для функции ϕ_0, и законом продолжения $U_0(z)$:

$$\Phi(P) = \sum_p \frac{(-1)^p\,\overline{w}^p}{4^p\,p!}\,\psi_0^{(-p)}(w)\left(\frac{d^2}{dz^2} + \kappa^2\right)^{(p)} U_0(z)\,; \tag{11}$$

здесь $\psi_0^{(-p)}$ обозначает первообразную порядка p; функция $\Phi(P)$ имеет структуру ряда (3) и тождественно удовлетворяет уравнению (4) при любых начальных функциях ψ_0 и U_0.

Окончательное решение (3) для случая (5), (6) может быть также записано в виде бесконечного ряда, который определяется начальной функцией ϕ_0 и одним из решений (9), симметричным $U_0^{(+)} = U_0$ или антисимметричным $U_0^{(-)} = \mathcal{U}_0$:

$$\Phi^{(\pm)}(P) = \sum_p (-1)^p\,\nabla^{2p}\phi_0(x,y)\,U_p^{(\pm)}(z)\,; \tag{12}$$

здесь $\nabla^{2p} = \left(\nabla^2\right)^{(p)}$. Отметим, что ряд (12) будет конечным, если ϕ_0 будет полиномиальной функцией; аналогично ряд (3), (8), (10) будет конечным, если U_0 будет многочленом. Представления (11), (12) наряду с формулами (9) позволяют построить системы рекуррентных соотношений для вычисления функций и их производных, используемые в алгоритмах блочного метода [5].

Выбирая разные начальные функции ψ_0, U_0, ϕ_0 в (11), (12) мы получаем необходимые для аппроксимации фундаментальные системы уравнения Гельмгольца. Для слоистых сред (рис. 1) выбираем полиномиальное представление для ϕ_0, тогда функции, отвечающие условиям сшивки температуры и тепловых потоков на границе слоя, записываются в виде конечной суммы:

$$\Omega_n^{(\pm)m} = \sum_{k,l} (-n+m)_{2k-2l}\,(-m)_{2l}\,A_{kl}^{(\pm)}\,\Phi_{n-2k}^{(\pm)m-2l} \quad , \quad |z| < b \, , \quad l \le k \, , \tag{13}$$

$$\Omega_n^{(\pm)m} = \Phi_n^{(\pm)m} + \sum_{k,l} (-n+m)_{2k-2l}\,(-m)_{2l}\,B_{kl}^{(\mp)}\,\Phi_{n-2k}^{(\mp)m-2l} \quad , \quad |z| > b \, , \tag{14}$$

где $\Phi_n^{(\pm)m}$ является решением (12) при $\phi_0 = x^{n-m} y^m$, b – полуширина слоя, а коэффициенты $A_{kl}^{(\pm)}$, $B_{kl}^{(\pm)}$ вычисляются последовательно по рекуррентным формулам через параметры задачи. Здесь система функций $\Omega_n^{(+)m}(P)$ – четная по z, система $\Omega_n^{(-)m}(P)$ – нечетная по z (соответственно меняется знак у коэффициентов $B_{kl}^{(\mp)}$ при симметричном продолжении); слой расположен симметрично относительно оси z.

Аналогичное представление получаем для цилиндрических включений, если в качестве функций $\Phi_n^{(\pm)m}$ будут выбраны решения (11) при $\psi_0 = w^{\pm m}$ и $U_0 = z^{n-m}$:

$$\Omega_n^{m} = \sum_{k,l} (-n+m)_{2k-2l}\,(-m)_{2l}\,A_{kl}\,\Phi_{n-2k}^{(+)m-2l} \quad , \quad |w| < r_0 \, , \quad l \le k \, , \tag{15}$$

$$\Omega_n^{m} = \Phi_n^{(+)m} + \sum_{k,l} (-n+m)_{2k-2l}\,(-m)_{2l}\,B_{kl}\,\Phi_{n-2k}^{(-)m-2l} \quad , \quad |w| > r_0 \, ; \tag{16}$$

здесь r_0 – радиус цилиндрического включения, система координат связана с центральной осью цилиндра. Отметим, что $\psi_0 = w^m$, $m \ge 0$, соответствует функциям Бесселя первого рода, а $\psi_0 = w^{-m}$ соответствует функциям Бесселя второго рода.

И наконец, для сферических включений решение, отвечающее условиям сшивки температуры и тепловых потоков выражается через обобщенные шаровые функции Φ_n^m, которые соответствуют решениям (11) при $\psi_0 = w^m$ и $U_0 = J_{n+1/2}(\kappa z) \big/ z^{m+1/2}$ (см. [5]):

$$\Omega_n^{m} = A_n\,\Phi_n^m \quad , \quad R < R_0 \, , \qquad R = \sqrt{x^2 + y^2 + z^2} \, , \tag{17}$$

$$\Omega_n^{m} = \Phi_n^m + B_n \Phi_{-n-1}^m \, , \quad R > R_0 \, ; \tag{18}$$

здесь R_0 – радиус сферического включения, система координат связана с центром шара.

Таким образом, с помощью метода квазиразделения переменных мы получаем необходимый аналитический аппарат для аппроксимации тепловых полей в подобластях-блоках, содержащих включения сферической, цилиндрической или пластинчатой формы с другими теплофизическими свойствами. Соответствующие функции (13) – (18) аналитически точно удовлетворяют уравнению (1) и естественным условиям сопряжения (2), и записываются в виде конечной суммы через функции Бесселя первого, второго рода или тригонометрические функции. Для этих функций, с помощью следующих из представлений (11), (12) рекуррентных соотношений, реализованы эффективные алгоритмы их вычисления и дифференцирования, необходимые для формирования блочной системы линейных алгебраических уравнений, возникающей при аппроксимации тепловых полей методом блоков.

2.2 Блочный метод наименьших квадратов

При аппроксимации процесса теплопередачи в структурно-неоднородных средах надо иметь в виду, что общее решение уравнения теплопроводности (в ограниченной области) может быть представлено в виде суммы равновесной и неравновесной составляющей [11]:

$$\theta(P,t) = \theta_0(P,t) + \sum_{p=1}^{\infty} \Phi_p(P)\, e^{-\kappa_p^2 t} \, , \tag{19}$$

где равновесная составляющая $\theta_0(P,t)$ удовлетворяет уравнению без начальных условий, а неравновесная представляет собой сумму отдельных гармоник, каждая из которых убывает

экспоненциальным образом со своей скоростью $\kappa_p^2 \lambda / C$; причем функции $\Phi_p(P)$ являются собственными функциями уравнения Гельмгольца соответствующими собственным значениям κ_p^2 и однородным краевым условиям на границе области.

Равновесная составляющая решения определяется единственным образом как решение задачи без начальных условий, и является функцией, обратимой по времени, характеризующей устойчивую реакцию системы на внешние краевые условия; для постоянных граничных условий T_0 не зависит от времени. Неравновесная составляющая определяется начальными и граничными условиями и характеризует инерционность системы по отношению к изменению внешних условий. Характерной особенностью является необратимость по времени, определяемая экспоненциальным характером зависимости от времени: решение экспоненциально возрастает при обращении времени.

Исходя из этих особенностей, решение в каждом блоке, содержащем неоднородность, представляется в виде конечной суммы:

$$\theta(P,t) = \sum_{p=0}^{L} \Phi_p(P)\, e^{-p^2 t}, \quad \Phi_p(P) = \sum_{n=0}^{M} \sum_{m=0}^{n} \left[a_{nm}\, \Omega_n^{(+)m}(P-P_0) + b_{nm}\, \Omega_n^{(-)m}(P-P_0) \right], \qquad (20)$$

где функции $\Omega_n^{(\pm)m}$ из формул (13) – (18) соответствуют показателю $\kappa = p$, P_0 – центр блока; функции $\Omega_n^{(\pm)m}$ для формул (15) – (18) определяются как $\Omega_n^{(+)m} = \mathrm{Re}\, \Omega_n^m$, $\Omega_n^{(-)m} = \mathrm{Im}\, \Omega_n^m$. Комбинация (20) аппроксимирует одновременно равновесную и неравновесную составляющие решения на некотором ограниченном интервале времени $t \in (0, t_0)$; расширение этого интервала предполагает итерационное применение схемы аппроксимации (20).

Сшивка локальных решений в блоках осуществляется при помощи блочного варианта метода наименьших квадратов (см. [1,2]), или при помощи модифицированного функционала метода наименьших квадратов, содержащего энергетические слагаемые, выполняющие роль регуляризатора Тихонова для вырожденного функционала наименьших квадратов [3-5].

Одновременная минимизация всей системы функционалов реализуется в виде блочной системы линейных алгебраических уравнений для нахождения неизвестных коэффициентов в представлениях (20) для каждого блока:

$$(T_k + \varepsilon B_k)\, \vec{X}_k + \sum_l T_{kl}\, \vec{X}_l = \vec{H}_k, \qquad k = 1,\ 2,\ \ldots\ N; \qquad (21)$$

здесь T_k – комплексная матрица Грама аппроксимирующей системы функций, B_k – матрица жесткости для функционала энергии, ε – параметр регуляризатора Тихонова, $\vec{X}_k$ – неизвестные коэффициенты в разложении (20), T_{kl} – матрицы, обеспечивающие сшивку локальных решений между блоками B_k, B_l и состоящие из скалярных произведений аппроксимирующих функций этих блоков, $\vec{H}_k$ – вектор граничных условий в блоке, N – общее число блоков. Размер каждого блока $L \times M_k$ совпадает с числом неизвестных коэффициентов в локальном представлении решения (20).

Разбиение расчетной области на подобласти-блоки, $\bar{G} = \bigcup \bar{B}_k$, $\mathrm{B}_k \bigcap \mathrm{B}_l = \varnothing$, $k \neq l$, т.е. введение блочной структуры, должно соответствовать структуре неоднородностей в материале. Для областей с регулярной структурой метод асимптотического усреднения [6] сводит задачу к одной ячейке с включением. В этом случае ячейка трактуется как представительный элемент неоднородного материала; в принципе она может иметь сложную структуру, и тогда она разбивается на более простые подобласти-блоки.

С каждым блоком B_k в блочной структуре разбиения расчетной области связывается норма наименьших квадратов, соответствующая скалярному произведению в интегральной норме по границе области в классе функций, аналитически точно удовлетворяющих оператору задачи:

$$F_k(\theta_k,\theta_l)=\|\theta_k\|^2_{B_k}\Big|_{t=0}+\int_0^{t_0}\left\{\left\|\alpha_k\,\theta_k+\beta_k\,\frac{\partial\theta_k}{\partial n}\right\|^2_{S_k}+\sum_l\left\|\alpha_{kl}\,(\theta_k-\theta_l)+\beta_{kl}\,\frac{\partial(\theta_k-\theta_l)}{\partial n}\right\|^2_{S_{kl}}\right\}dt\,, \qquad (22)$$

где $S_{kl}=\partial \mathrm{B}_k\cap\partial \mathrm{B}_l$ – общая часть границы для соседних блоков, $S_k=\partial \mathrm{B}_k\cap\partial G$ – общая часть границы блока и границы области G.

Коэффициенты α_{kl}, β_{kl}, α_k, β_k должны обеспечить, с одной стороны, невырожденность нормы в блоке при условии равенства нулю функций из соседних блоков, а с другой стороны, сшивку функций и нормальных производных на границах между блоками; это, в частности, предполагает, что матрица сшивочных коэффициентов является невырожденной:

$$F_k(\theta_k,0)=0 \;\Leftrightarrow\; \theta_k\equiv 0\,, \quad P\in \mathrm{B}_k\,, \qquad (23)$$

$$\det\begin{Vmatrix}\alpha_{kl} & \beta_{kl}\\ \alpha_{lk} & \beta_{lk}\end{Vmatrix}\neq 0\,, \quad P\in S_{kl}\,. \qquad (24)$$

Для области в целом водится норма наименьших квадратов на многоблочной структуре по следующей формуле:

$$F(\theta)=\max_k F_k\,, \quad \theta=\theta_k\,, \quad P\in \mathrm{B}_k\,. \qquad (25)$$

Аппроксимация решения обеспечивается сходимостью к точному решению по норме на многоблочной структуре: $F(\theta-\theta_0)\to 0$, где θ_0 – точное решение задачи. Одновременная минимизация функционалов F_k (составляющих общую норму) на своем наборе функций дает алгоритм для оценки сверху абсолютного минимума нормы $F(\theta-\theta_0)$ и сводится к решению блочной системы уравнений (21) (при $\varepsilon=0$).

Изложенный подход к аппроксимации обеспечивает удовлетворение начальным условиям, а также одновременную сшивку температуры и теплопотоков на границе блоков, однако не учитывает потери энергии в случае фазовых превращений; при этом предполагается, что граница включений проходит по границе блоков. Фазовые переходы связываются с модификацией функционала энергии задачи, который должен учитывать еще скрытую энергию фазовых превращений в области или на поверхности, где происходят превращения.

Эти моменты могут быть учтены в подходе к аппроксимации [3-5], основанном на введении полунормы $F(\theta)$, обеспечивающей только начальное распределение и сшивку температуры, и функционала энергии $E(\theta)$, выполняющего роль регуляризатора для полунормы $F(\theta)$ и учитывающего фазовые превращения в материале; при таком подходе минимизируется модифицированный функционал с регуляризатором, $F(\theta)+\varepsilon E(\theta)=\min$. При достаточно малом ε модифицированный функционал обеспечивает одновременную сшивку функций и минимизацию энергии с учетом энергии фазовых превращений и приводит к блочной системе уравнений (21) при $\varepsilon\neq 0$.

3. Комбинированный параллельный алгоритм решения блочной системы уравнений

Основной операцией при формировании блочной системы (21) является вычисление скалярных произведений моментных характеристик аппроксимирующей системы функций (температура, теплопотоки, общая энергия), они вычисляются через производные до второго порядка с помощью интегрирования по границе блоков. Здесь содержится значительный ресурс параллелизма, поскольку формирование блочной системы производится с помощью коллокационного вектора, составленного из значений аппроксимирующей системы функций в одной точке. Поэтому операции полностью независимы на уровне разных блоков и даже на уровне разных узлов квадратуры Гаусса для поверхностного интегрирования. Вычисление же моментных характеристик сводится к рекуррентным преобразованиям коллокационного вектора на основе особых дифференциальных свойств используемой системы функций, следующих из представлений (11), (12).

Основная специфика системы линейных уравнений (21) состоит в следующем:

1) аппроксимация строится на относительно небольшом числе элементов;
2) качество аппроксимации достигается, прежде всего, за счет увеличения числа базисных функций в блоке.

С точки зрения линейной алгебры матрица системы уравнений является вещественной или комплексной плотно блочно разреженной матрицей, с относительно небольшим числом блоков и относительно большим размером каждого блока. Кроме того, в силу специфики построения системы уравнений среди диагональных блоков матрицы системы встречаются очень плохо обусловленные.

Блочная система уравнений (21) решается масштабированным блочно-предобусловленным алгоритмом GMRES, в котором в качестве переобуславливания использовалось блочное неполное LU разложение второго порядка точности. Предварительно, перед вычислением неполного разложения матрицы, проводится блочно-диагональное масштабирование системы уравнений, в результате которого диагональные блоки коэффициентов становятся единичными матрицами.

Неполное блочное разложение для системы уравнений строится на основе соотношения

$$A + E = L * U + L * R + W * U , \tag{26}$$

где блочно-треугольные матрицы L и U содержат блочные элементы разложения “первого порядка” точности, а блочно-треугольные матрицы W и R содержат блочные элементы “второго порядка” точности; E – некоторая матрица ошибки. Это разложение строится как несимметричное блочное обобщение алгоритма из работы [11].

Итерации по решению системы уравнений проводятся с использованием переобусловленного варианта алгоритма GMRES [12]. Алгоритм GMRES основан на следующих матричных соотношениях:

$$b = P_1 g_1 , \tag{27}$$

$$A * (LU)^{-1} * P_k = P_{k+1} * H_k , \tag{28}$$

где H_k – верхняя хессенбергова форма с размером $(k+1)\times k$, b – вектор правой части, P_k – матрица с ортонормированными столбцами размера $N \times k$. Для построения матричных соотношений (27) и (28) требуется на каждой итерации алгоритма: умножение на матрицу A, решение систем уравнений с блочно треугольными матрицами L и U, а также ортогонализация. Для обеспечения численной устойчивости вычислений ортогонализации осуществляются неявным образом на основе преобразований Хаусхолдера. Новое приближение к решению системы линейных уравнений строится по формуле $x_k = (LU)^{-1} * P_k * y_k$, где y_k есть решение задачи минимизации $\| H_k \, y_k - e_1 \, g_1 \| = \min$.

Генерация и решение блочной системы уравнений (21) является наиболее затратной частью метода блоков. Вычислительные затраты в описанном алгебраическом алгоритме растут как минимум кубическим образом в зависимости от числа базисных функций в блоке. По этой причине для уменьшения времени счета необходимо подходящим образом распараллелить вычисления.

Современные суперкомпьютерные вычислительные системы как правило имеют неоднородную архитектуру. С одной стороны, имеется набор вычислительных узлов с распределенной памятью, обмен данными между которыми может быть осуществлен по быстрой обменной сетке. С другой стороны, каждый узел представляет собой многопроцессорный/многоядерный компьютер с общим доступом к оперативной памяти.

Специфической особенностью метода блоков является относительно малое число используемых блоков и как правило большое число базисных функций в каждом блоке. На малом числе блоков непросто обеспечить эффективность параллельных вычислений по распределенной памяти при использовании большого числа процессоров/ядер. С другой стороны, огромная необходимая вычислительная работа с плотными подматрицами может быть относительно просто распараллелена по общей памяти.

Для снижения доли распределенных параллельных вычислений и увеличения использования общей памяти в алгоритме блоков для распараллеливании вычислений была выбрана комбинированная модель параллельного программирования MPI+threads. В этой модели при рас-

параллеливании вычислений между узлами для организации обменов и синхронизаций используется стандарт обмена сообщениями MPI. При этом на каждом узле имеется только один MPI процесс, который затем порождает на этом узле нужное количество потоков вычислений, одновременно работающих по общей памяти.

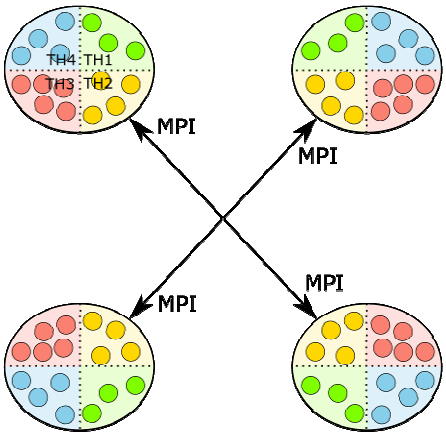
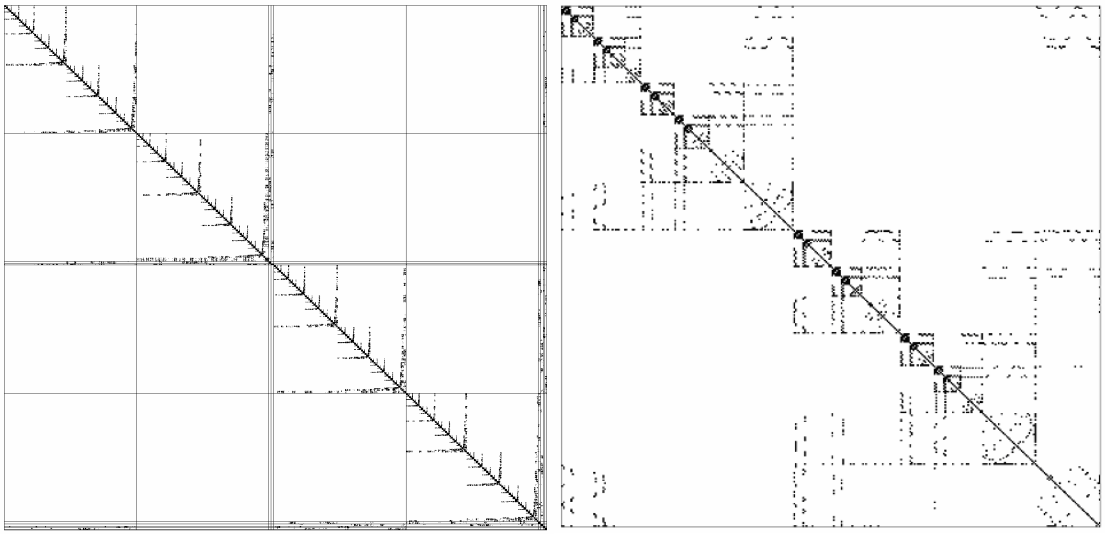


Рис. 4. Комбинированная MPI+threads организация параллельных вычислений.

При параллельном решении системы уравнений (21) использовалась методика распараллеливания, аналогичная представленной в работе [13]. Распределение вычислительной работы по процессорам/ядрам осуществляется и на этапе генерации, и на этапе решения систем линейных уравнений на основе анализа графа блочной разреженности матрицы по блочным строкам/столбцам матрицы исходной системы уравнений. Граф блочной разреженности матрицы содержит информацию о геометрических связях между блоками расчетной сетки. Области, в которых решается задача, как правило, существенно трехмерные, а значит, декомпозиция задачи должна существенным образом учитывать эту трехмерность. В данной работе мы следует технике декомпозиции задачи решения системы уравнений на основе упорядочивания типа вложенных сечений ND (Nested Dissection) [14] с учетом декомпозиции поверхностных межпроцессорных границ [15].

Упорядоченная блочная структура матрицы для некоторой тестовой задачи имеет вид в построенном крупно-блочном биении, показанный на Рис. 5 а), а результирующая крупно-блочная структура матрицы в терминах крупно-блочного биения для паралельных вычислений показана на рис. 5 б).



а) *б*)

Рис. 5. Структура разреженности матрицы: а) при блочном биении, б) при крупно-блочном биении

Следует отметить, что крупно-блочная структура разреженности матрицы, показанная для тестовой задачи на рис. 5 б), демонстрирует также все основные зависимости между собой крупно-блочных вычислений в описанном выше алгоритме решения системы уравнений. Это утверждение имеет место при условии, что неполное разложение строится в крупно-блочном смысле “по позициям”.

Основная идея комбинированного распараллеливания вычислений при решении системы уравнений в методе блоков заключается в различной трактовке основных вычислений. С точки зрения распределенной памяти зависимости в вычислениях трактуются в терминах крупно блочного биения. И наоборот, с точки зрения общей памяти зависимости в вычислениях трактуются в терминах блоков, блочных строк и блочных столбцов. По этой причине разбиение задачи на набор гиперблоков осуществляется в соответствии с числом MPI процессов. Каждая гиперблочная строка/столбец матрицы и предобуславливателя приписывается соответствующему узлу вычислений. Вычисления внутри гиперблочной строки распараллеливаются по потокам следующим образом.

При вычислении неполного блочного разложения проводится update каждой блочной строки через некоторые предыдущие. При этом требуется произвести некоторое количество независимых умножений плотной подматрицы на плотную подматрицу, эти вычисления распараллеливаются по потокам вычислений. Число таких независимых вычислений равно числу блоков в блочной строке, по этой причине это ограниченный ресурс распараллеливания вычислений. Тем не менее, при небольшом числе вычислительных ядер в узле этого ресурса параллелизма оказывается достаточно при вычислении неполного блочного треугольного разложения. Возможно дальнейшее распараллеливание неполной блочной факторизации по общей памяти для большого числа ядер в узле. Для этого необходимо распараллелить по потокам собственно умножение плотной подматрицы на плотную подматрицу. Однако в этом случае для обеспечения масштабируемости вычислений потребуется также распараллелить по потокам вычислений сингулярное разложение при факторизации диагонального блока.

При распараллеливании вычислений по общей памяти в итерационной схеме также использовались зависимости вычислений между собой, которые описываются блочной структурой разреженности соответствующих матриц. Зависимости в вычислениях по распределенной памяти на разных узлах разрешались в соответствии с зависимостями по данным в крупно блочном биении.

4. Численные эксперименты

Численные эксперименты проводились на компьютере, имеющем два 6-ти ядерных процессора AMD Opteron 8435 (Istanbul), работающие по общей памяти на одной плате. Каждый процессор был подключен к двухканальной памяти DDR2 533, 12 Gb.

В качестве тестовой была рассмотрена задача по определению эффективных теплофизических свойств в регулярной среде с включениями пластинчатой, цилиндрической или сферической формы (рис. 1-3) методом асимптотического усреднения Бахвалова [6]. Для этого нужно решить в классе периодических функций задачу на ячейке периодичности с контактными условиями на границе включений Γ:

$$L_{\xi\xi}\left(N_{i_1}+\xi_{i_1}\right)=0,\quad \xi\notin\Gamma\ ;\quad \left[N_{i_1}\right]\Big|_{\xi\in\Gamma}=\left[\lambda_{ij}\frac{\partial\left(N_{i_1}+\xi_{i_1}\right)}{\partial\xi_j}n_i\right]\Bigg|_{\xi\in\Gamma}=0\,,\quad i_1=1,2,3\,; \qquad (29)$$

$$L_{\xi\xi}=\frac{\partial}{\partial\xi_i}\left(\lambda_{ij}(\xi)\frac{\partial}{\partial\xi_j}\right),\quad \lambda_{ij}(\xi)=\lambda_I\,\delta_{ij},\quad \xi\in G_I\,,\quad \lambda_{ij}(\xi)=\lambda_M\,\delta_{ij},\quad \xi\in G_M\,,$$

и вычислить эффективный тензор теплопроводности по формуле $\lambda^{\mathrm{e}}_{i_1i_2}=\left\langle\lambda_{i_1i_2}+\lambda_{i_1j}\,\partial N_{i_2}/\partial\xi_j\right\rangle$, где λ_I – коэффициент теплопроводности во включении G_I, а λ_M – коэффициент теплопроводности в матрице G_M, $\langle\cdot\rangle$ – среднее значение величины по объему ячейки.

Для решения краевой задачи (29) использовались обычные функции $\Phi_n^{(\pm)m}$ из раздела 2, а также блочное разбиение ячейки с включением на криволинейные блоки, как это показано для примера круглого включения на рис 6; результаты расчета этих характеристик блочным методом и их сопоставление между собой приведены на рис. 7. Даны графики эффективного модуля $\hat{\lambda}$ в сопоставлении с правилом смеси $\hat{\lambda} = \lambda_I f + \lambda_M (1-f)$, где f – коэффициент объемного наполнения.

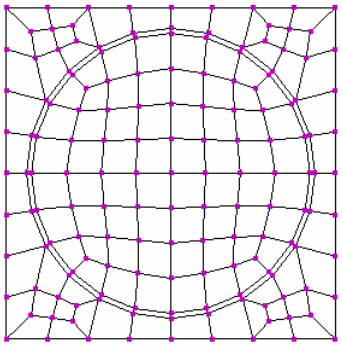
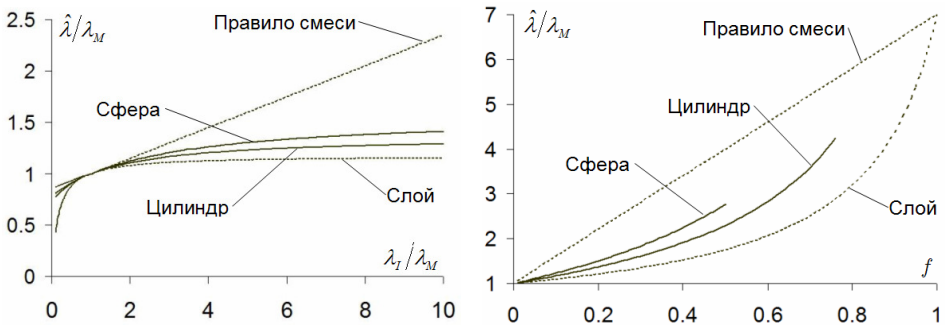


Рис. 6. Разбиение на криволинейные блоки ячейки с включением:



а) б)

Рис. 7. Эффективные характеристики периодических структур:
а) $f = 0.15$, б) $\lambda_I / \lambda_M = 7$

Как видим, простое правило смеси дает верхнюю границу и является грубым приближением для эффективного модуля теплопроводности; это обосновывает необходимость разработки специальных математических методов, учитывающих структуру и геометрию области.

Таблица 1 содержит результаты численных экспериментов для тестовой задачи с числом блоков $N = 1148$ при различном числе MPI процессов и потоков на один MPI процесс. Недостаточно высокая масштабируемость при решении тестовой задачи по видимому связана с недостаточной пропускной способностью подсистемы доступа к оперативной памяти при большом числе используемых ядер.

Таблица 1 Результаты численных экспериментов.

MPI x Threads	1x1	2x1	1x2	4x1	1x4	2x2	1x6	2x6	4x3	8x1
Ускорение	1	1.9	1.6	2.8	2.2	3.0	2.4	4.8	5.8	4.2

5. Заключение

В работе представлена комбинированная MPI+threads параллельная реализация метода блоков для моделирования теплофизических процессов в структурно-неоднородных средах. На приме-

ре тестовой задачи показана его эффективность по сравнению с чисто MPI параллельной реализацией [4], представленной ранее. Текущая параллельная реализация ориентирована на адекватный учет особенностей блочного метода, порождающего одновременно плотные блоки и разреженную структуру матрицы.

Работа поддержана программой Президиума РАН П-2, а также грантом № 09-01-135333 офи-ц..

Литература

1. *Волков-Богородский Д.Б.* Разработка блочного аналитико-численного метода решения задач механики и акустики // Сборник трудов школы-семинара “Композиционные материалы”. – М.: ИПРИМ РАН, 2000. – С. 44-56.
2. *Волков-Богородский Д.Б.* Подход к задачам о взаимодействии акустической и упругой среды с помощью блочного метода мультиполей // “Динамические и технологические проблемы механики конструкций и сплошных сред”. Материалы XI Международного симпозиума. – М.: МАИ, 2005. - Т. 2. - С. 17-22.
3. *Волков-Богородский Д.Б.* О вычислении эффективных характеристик композиционных материалов с помощью блочного аналитико-численного метода // “Динамические и технологические проблемы механики конструкций и сплошных сред”. Материалы XII Международного симпозиума. Избранные доклады. - М.: МАИ, 2006. - С. 41-47.
4. *Волков-Богородский Д.Б., Харченко С.А.* Параллельные вычисления в методе блоков для связных задач волновой виброакустики // Труды Международной научной конференции “Параллельные вычислительные технологии”, Санкт-Петербург, 28 января – 1 февраля 2008 г. – Челябинск: Изд-во ЮУрГУ, 2008. – С. 347-352.
5. *Волков-Богородский Д.Б.* Применение аналитических расчетов на основе метода блоков в связных задачах механики сплошных сред // “Прикладные исследования в механике”. Труды всероссийской научно-практической конференции “Инженерные системы-2008”, 7-11 апреля 2008. – М.: Российский университет дружбы народов, 2008. – С. 123-138.
6. *Бахвалов Н.С., Панасенко Г.П.* Осреднение процессов в периодических средах. – М.: Наука, 1984. – 352с.
7. *Карслоу Г., Егер Д.* Теплопроводность твёрдых тел. – М.: Наука, 1964. – 488с.
8. *Лыков А.В*. Теория теплопроводности. – М.: Высшая школа, 1967. – 599с.
9. *Бейтмен Г., Эрдейи А.* Высшие трансцендентные функции. – Т. 1. М.: Наука, 1973. – Т. 2. М.: Наука, 1974.
10. *Тихонов А.Н., Самарский А.А.* Уравнения математической физики. – М.: Наука, 1977. – 735с.
11. *Kaporin I.E.* High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$ decomposition // Numer. Linear Algebra Appl. – 1998. – V. 5. – P. 483-509.
12. *Saad Y., Schultz M.H.* GMRES: A generalized minimum residual algorithm for solving non-symmetric linear systems // SIAM J. Sci. Comput. – 1986. – V. 7. – P. 856-869.
13. Сушко Г.Б., Харченко С.А. “ Экспериментальное исследование на СКИФ МГУ "Чебышев" комбинированной MPI+threads реализации алгоритма решения систем линейных уравнений, возникающих во FlowVision при моделировании задач вычислительной гидродинамики” // Труды международной научной конференции Параллельные вычислительные технологии (ПаВТ'2009), Нижний Новгород, 30 марта – 3 апреля 2009 г. Челябинск, Изд. ЮУрГУ, 2009, с.316-324.

14. *George A., Liu J.W.* Computer Solution of Large Sparse Positive Definite Systems // Series in Computational Mathematics. – Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

15. *Харченко С.А.* Влияние распараллеливания вычислений с поверхностными межпроцессорными границами на масштабируемость параллельного итерационного алгоритма решения систем линейных уравнений на примере уравнений вычислительной гидродинамики // Труды Международной научной конференции “Параллельные вычислительные технологии”, Санкт-Петербург, 28 января – 1 февраля 2008 г. – Челябинск: Изд-во ЮУрГУ, 2008. – С. 494-499.

Результаты масштабирования бенчмарка NPB UA на тысячи ядер суперкомпьютера Blue Gene/P с помощью PGAS-расширения OpenMP*

А.А. Корж

В статье рассмотрено распараллеливание бенчмарка Unstructured Adaptive из пакета NAS Parallel Benchmarks в парадигме PGAS, дополняющей парадигму OpenMP, для машин с распределенной памятью. Рассматривается реализация данной парадигмы на суперкомпьютере IBM Blue Gene/P. Приводятся результаты исследования производительности рассмотренного бенчмарка. На 2048 ядрах системы Blue Gene/P установленной в Московском Университете получены результаты, превосходящие ранее известные результаты для OpenMP-версии на машинах с общей памятью.

1. Введение

Будущая серия машин Cray Baker, разрабатываемая в рамках военной программы DARPA HPCS, ожидается на рынке во втором квартале 2010 года. Оставаясь в рамках традиционной MPP-архитектуры (десятки тысяч узлов, в каждом из которых по два процессора AMD Opteron, узлы соединены коммуникационной сетью с топологией 3D-тор) Baker будет использовать принципиально новую коммуникационную сеть с кодовым названием Gemini. Одним из существенных отличий Gemini от интерконнекта Seastar2+ машин серии Cray XT будет являться значительно более высокий темп передачи коротких сообщений (Message Rate) и аппаратная поддержка парадигмы PGAS. Все это значит, что сеть обеспечит эффективную передачу десятков миллионов сообщений в секунду. Другой проект, разрабатываемый фирмой IBM в рамках DARPA HPCS – PERCS. В рамках этого проекта планируется представить суперкомпьютер IBM Blue Waters в 2011 году, который также будет использовать процессоры со стандартной суперскалярной архитектурой POWER7, соединенные между собой коммуникационной сетью с пропускной способностью 400 Гбит/с на узел. Среди новшеств также значится аппаратная поддержка парадигмы общей памяти PGAS. Все это говорит о том, что вскоре PGAS получит большую популярность и имеет возможность потеснить парадигму MPI в высокопроизводительных вычислениях.

С другой стороны, более 99 процентов используемого на современных суперкомпьютерах кода написано с применением библиотеки MPI (на языках Fortran и C) и опыт показывает, что переходить на другие парадигмы и языки программирования пользователи не хотят, не видя существенных преимуществ той или иной парадигмы.

Целью данной работы – показать пример задачи, для которой использование парадигмы PGAS сможет дать значительные преимущества относительно реализации с использованием стандартных MPI/OpenMP. Предположительно, такую задачу следует выбирать из класса задач, имеющих нерегулярный шаблон доступа к данным, то есть мелкую гранулярность обращений, низкую пространственную и временную локализацию обращений к памяти. В частности, расчеты на нерегулярных адаптивных сетках относятся к данному классу. По этой причине автором была выбрана модельная задача UA (Unstructured Adaptive) из известного пакета бенчмарков NASA NAS Parallel Benchmark (NPB). Впервые бенчмарк NPB UA был добавлен в версию 3.1 пакета NPB с целью исследования производительности суперкомпьютеров на задачах с нерегулярным динамически изменяемым шаблоном доступа.

В бенчмарке NPB UA решается задача Дирихле уравнения теплопереноса в трехмерной кубической области на нерегулярной декартовой сетке. Источник тепла представляет собой шар, движущийся с постоянной скоростью. Самым существенным, с точки зрения шаблона доступа к памяти, является тот факт, что для решения задачи используется нерегулярная сетка. Причем каждые несколько шагов происходит адаптация сетки: на областях с большим градиентом тем-

* Работа выполнена при поддержке РФФИ, грант № 09-07-13596-офи_ц.

пературы сетка измельчается, с малым — укрупняется. Для решения задачи применяется спектральный метод конечных элементов (SEM) с применением метода конечных мортаров. Подробное описание применяемого численного метода и его преимуществ можно прочесть у авторов бенчмарка в [2].

До сих пор существовало лишь две известные реализации данного бенчмарка — последовательная версия и OpenMP-версия (NPB-OMP UA). MPI-версии, показывающей сколь-нибудь хороший результат на стандартных Infiniband-кластерах, так никому и не удалось представить. В связи с этим, разработанная автором версия, которая может работать на стратегическом суперкомпьютере IBM Blue Gene/P без аппаратной поддержки общей памяти представляет существенный интерес. С другой стороны, бенчмарк представляет собой реальную программу на Fortran77, суммарный объем кода составляет 8000 строк без учета комментариев. Поэтому, тот факт, что на распараллеливание было потрачено около двух недель, доказывает крайне высокую продуктивность программирования в стиле PGAS.

2. PGAS/OMP-версия NPB-UA

2.1 PGAS-расширение OpenMP

Модели параллельного программирования можно разделить на два класса, в зависимости от того, на коммуникациях какого типа они основаны: односторонних (обращения к удалённой памяти) либо двусторонних (передача сообщений). В двусторонних коммуникациях (используемых в библиотеке MPI версии 1) активное участие принимают две стороны: одна отправляет записываемое слово, а вторая — ждёт прихода слова, после чего копирует полученное слово из буфера приёма в нужную ячейку памяти. Адрес, куда необходимо записать слово в памяти второго узла, указывается самим получателем. В односторонних коммуникациях активное участие принимает лишь инициатор: при записи он отсылает записываемое слово, которое, достигнув по сети адресата, напрямую записывается в память второго узла (при этом процессорное время на ожидание и запись вторым узлом не тратится). Адрес, куда записать слово в памяти второго узла, указывается отправителем.

Система программирования SHMEM (от shared memory — общая память) была разработана фирмой Cray более 15 лет назад, как интерфейс односторонних коммуникаций, способный стать эффективной альтернативой и дополнением к MPI и PVM. Интерфейс SHMEM поддерживается всеми MPP-системами фирмы Cray (Cray T3E, Cray XT3/4/5/6), Silicon Graphics (SGI Altix), интерконнектами Quadrics (QsNetIII). Также библиотека SHMEM (с некоторыми дополнениями) реализована в системе МВС-Экспресс (разработана под руководством А. О. Лациса).

По сути SHMEM реализует простейший вариант программирования в стиле PGAS (Partitioned global address space). У каждого узла есть локальная память; каждому узлу также доступна удалённая память: узел может напрямую обращаться к локальной памяти любого узла системы. Поскольку обращения к удалённой памяти происходят через коммуникационную сеть, время их выполнения заметно больше, а темп — меньше, чем у обращений к локальной памяти. Ожидать выполнения каждой одиночной операции крайне дорого, поэтому требуется, чтобы программист явно выделял обращения к нелокальным ячейкам памяти.

В отличие от других PGAS-языков, например, UPC, SHMEM *заставляет* программиста явно выделять внешние обращения с помощью функций, при этом дальнейшая группировка обращений и оптимизация выполняются аппаратно. Здесь можно добавить сравнение с парадигмой общей памяти OpenMP, в которой программисту следует разрезать вычисления на части, не заботясь о распределении памяти. Но учесть различие в цене доступа к памяти NUMA систем, в особенности систем без кэш-когерентной общей памяти, данная парадигма не в силах. Именно поэтому поддержка OpenMP не смогла быть реализована эффективно на системах с распределенной памятью, хотя безуспешные попытки и предпринимались (Intel Cluster OpenMP и ScaleMP vSMP). Парадигма PGAS расширяет парадигму общей памяти OpenMP тем, что программисту надо не только распределить вычисления, но также распределить данные, а при распределении вычислений учесть то, как были распределены данные, что приводит к более эффективно работающему коду, учитывающему локальность распределения данных.

Основу SHMEM составляют две операции: `shmem_put` — запись в память удалённого узла и `shmem_get` — чтение из памяти удалённого узла. Синхронизация происходит с помощью встроенной функции `shmem_barrier_all`. Возможность напрямую обращаться к удалённой памяти даёт большинство преимуществ работы с «общей памятью», не накладывая никаких дополнительных ограничений на то, как память распределена физически.

Родной реализации SHMEM на суперкомпьютере IBM Blue Gene/P нет. Однако используемый в суперкомпьютере Blue Gene/P интерфейс DCMF [1] среднего уровня содержит функции `DCMF_Put, DCMF_Get, DCMF_Send` и другие, на основе которых довольно легко реализовать интерфейс SHMEM. Несколько нюансов содержится в реализации отвечающей парадигме SHMEM синхронизации с помощью `shmem_barrier_all`, которая не только выполняет синхронизацию процессов, но и гарантирует получение всех отправленных другими процессорами операций `shmem_put`. Однако эти вопросы остаются за рамками данной статьи.

Сравнения ради укажем, что в сделанной реализации темп выдачи сообщений `shmem_put` размером в 8 байт составляет 1 млн/с, при использовании одного ядра и 2 млн/с при использовании двух и четырех ядер на одном узле. Одной из причин такой низкой производительности является использование режима DMA, который эффективен для передачи длинных сообщений, но не оптимален в случае коротких сообщений. В частности, текущая реализация использует технику callback-ов для получения подтверждений, о том, что аппаратура DMA прочла все посланные сообщения. Исследуются возможности более эффективной реализации SHMEM с помощью интерфейса нижнего уровня SPI. Также перспективным представляется техника агрегации сообщений, которая впрочем, неприменима при экстремальных уровнях масштабирования.

Для сравнения, использование макета М3 специализированной сети, разработанной в НИЦЭВТ под руководством автора дает темп выдачи 15 млн/с, система МВС-Экспресс [3], разработанная под руководством А.О. Лациса имеет темп выдачи 12 млн/с. Однако следует учесть, что узлы Blue Gene/P имеют в 4 раза меньшую частоту, и в 8 раз меньшую производительность, чем стандартные x86-узлы, при этом доступное их количество составляет сотни-тысячи узлов.

2.2 Описание структуры программы NPB UA

С программистской точки зрения один временной шаг бенчмарка NPB UA состоит из следующих этапов: 1) продвижение конвекционной части уравнения явным методом Рунге-Кутта 4-го порядка 2) продвижение диффузионной части уравнения 3) изменение сетки, если прошло заданное число шагов. Основным по трудоемкости является продвижение диффузионной части, так как оно включает решение систем линейных уравнений с помощью метода сопряженных градиентов с предобуславливанием. Например, на процессоре PowerPC 450, используемом в узле суперкомпьютера Blue Gene/P, время счета на последовательной версии на задаче класса C распределяется следующим образом:

Адаптация сетки	0.40 %
Конвекция	24.4 %
Прочие вычисления	3.80 %
400 операций `scatter/gather`	1.90 %
Диффузия	69.5 %

Всего задача класса C содержит 4900 операций `gather/scatter` (4200 из которых содержится в продвижении диффузионной части), а это примерно 23 % времени счета.

2.3 Метод конечных мортаров и его распараллеливание

Использование метода MEM (Mortar Elements Method), описанного в [3], и перенумерация элементов согласно мортоновской нумерации каждый раз после изменения сетки значительно облегчают процесс распараллеливания. В этой вариации метода конечных элементов каждый элемент покрывается сеткой из $(N+1)^3$ точек. Из-за нерегулярности сетки, точки находящиеся на границе смежных элементов могут не совпадать друг с другом. Для решения проблемы переноса температуры смежных элементов применяется два множества точек. Первое и основное

множество точек называется точками коллокации и состоит из всех точек дискретизации каждого элемента, таким образом, их общее число составляет $N_{elt}\ (N+1)^3$.

Очевидно, что часть точек на границе элементов может содержаться в двух или трех соседних элементах, при этом это будут разные точки коллокации. Второе множество, является подмножеством точек коллокации, не содержит совпадающих точек и образует сетку. Очевидно, что точки коллокации, лежащие во внутренностях элементов идентичны точкам сетки не лежащим на границе. Отличаются же точки коллокации и точки сетки только теми точками, которые лежат на границе элементов, такие точки будем называть точками согласования. Изначально сетка содержит один элемент, совпадающий с кубической областью. В начале теста и каждый пятый шаг происходит адаптация сетки: в зависимости от градиента температуры на каждом элементе, он может быть измельчен на восемь новых кубических элементов или, наоборот, восемь кубиков одного размера могут быть объединены в один. При этом накладывается ограничение, что смежные элементы имеют либо одинаковый размер, либо отличаются размером стороны в два раза.

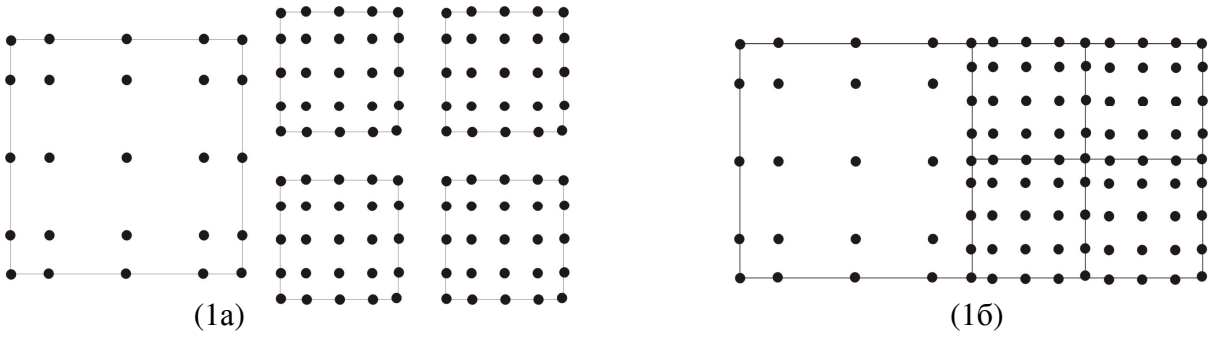


Рис. 1 Точки коллокации (1а) и точки сетки (1б) на грани элемента

Все массивы, с которыми оперирует исходная программа, разделяются на три группы:

- массивы, содержащие информацию об элементах и точках сетки
- массивы с плавающей точкой, индексированные точками коллокации (tx-массивы)
- массивы с плавающей точкой, индексированные точками согласования (tmor-массивы)

С точки зрения программиста, почти все вычисления производятся в циклах с индексом, пробегающим или множество элементов или множество точек согласования. При этом каждая итерация цикла оперирует соответственно или с точками коллокации данного элемента или с точками согласования. Поэтому, достаточно распределить точки согласования и точки коллокации между процессорами, и соответственно распределятся все итерации таких циклов, причем между итерациями циклов нет зависимости по данным. Самыми нетривиальными с точки зрения распараллеливания операциями являются операции `gather` и `scatter`. Эти операции отображают решение из точек коллокации в точки согласования и обратно, причем они выполняются на каждой итерации метода сопряженного градиента.

Для первоначальной версии было принято решение не распараллеливать процедуру адаптацию сетки. Учитывая тот факт, что она занимает 0.4 процента времени, масштабирование такого варианта будет теоретически ограничено коэффициентом 250. Также увеличивается суммарное потребление памяти, так как массивы, содержащие информацию об элементах и точках сетки, будут продублированы на каждом узле. Также, при переходе от параллельной части потребуется дополнительно выполнить коллективную операцию `all2all` для того, чтобы собрать данные по температуре с прошлой итерации на каждом узле перед адаптацией сетки, которая выполняется каждым узлом с одними и теми же данными независимо. Это также негативно отразится на масштабировании задачи.

2.4 Операция gather

Упрощенно, код, выполняемый операцией `gather`, переносящей вычисленное значение температуры с точек согласования на граничные точки коллокации, выглядит следующим образом:

```
do i=1,nelt
 do j=1,125
  tx(i,j) = a(j,1)*tmor(idmo(i,f(j,1))) + a(j,2)*tmor(idmo(i,g(j,1)))+...
 end do
end do
```

В данном фрагменте `nelt` — число элементов сетки N_{elt}, 125 = $(N+1)^3$ — число точек коллокации в каждом элементе, массив `tx` индексируется точками коллокации, массивы `a`, `f` и `g` являются константами, `idmo` - индекс вектор, индексируемый точками коллокации, массив `tmor` индексируется точками согласования.

Именно в данном цикле содержится существенная нерегулярность метода, так как соответствующие точки коллокации относящиеся к одному элементу сетки разбросаны случайным образом по массиву. Проиллюстрировать нерегулярность можно следующим графиком (см. Рис. 2), на котором изображен шаблон коммуникаций для маленького класса задачи W (число элементов — 600, число точек согласования — 30000).

На графике прекрасно видно, как шаблон нерегулярности меняется каждые 5 шагов. С другой стороны, можно отметить, что шаблон доступа к элементам не является полностью случайным: довольно большое количество обращений сконцентрировано на двух «прямых».

Отметим, что нижняя «толстая» прямая соответствует вершинам элементов, а диагональная «прямая» — точкам на гранях и ребрах элементов. Точки, которые выбиваются из этих «прямых» соответствуют граничным точкам элементов, соседних с данным. Исходя из сделанного анализа, осуществлялось распределение данных по процессорам.

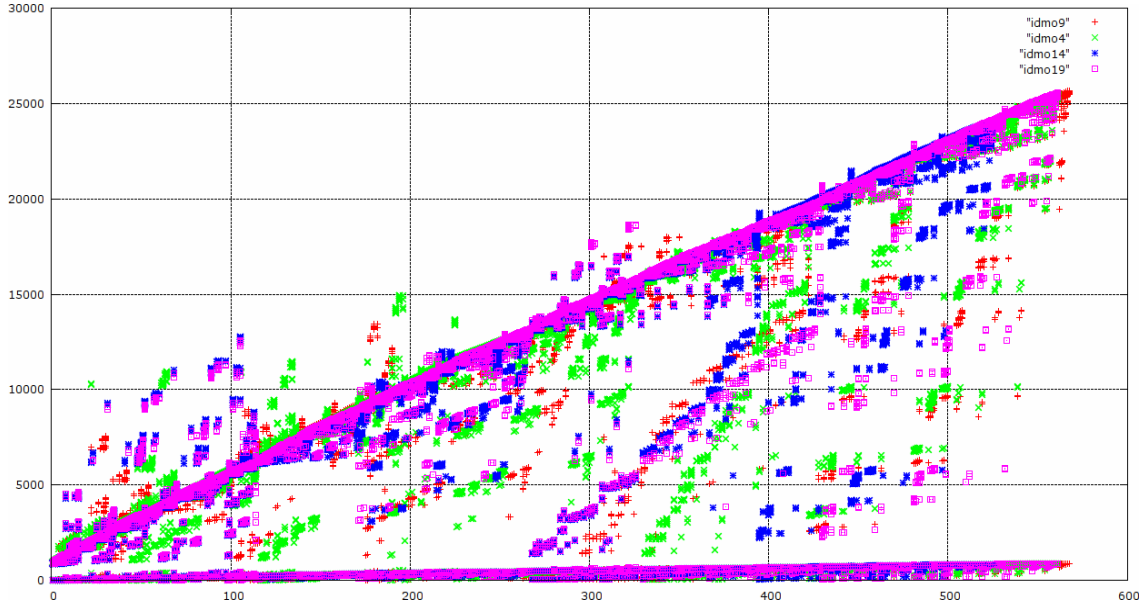


Рис. 2 – Соответствие элементов сетки (по оси абсцисс) точкам согласования (по оси ординат) в операциях Gather/Scatter в NPB UA (class W) для различных временных шагов.

2.5 Распределение данных

Выбор стратегии распределения данных имел своей целью следующее: обеспечение максимальной локализации обращений к данным, то есть уменьшение числа удаленных обращений к памяти (RMA или Remote Memory Accesses), обеспечение равного количества удаленных обращений к памяти от разных узлов для того, чтобы избежать простоя тысяч процессоров, ожидающих последнего на барьере. Причем важно не только количество исходящих обращений, но и количество входящих обращений, чтобы отдельные узлы не стали узким местом системы.

Были приняты следующие решения по распределению данных: элементы сетки и массивы, индексируемые точками коллокации, распределяются блочно по узлам распределенной системы. Таким образом, обращение к данным точек коллокации во всех циклах будет осуществляться последовательно и регулярно, как это происходит и в последовательной версии теста. Это относится к массивам `tx`, `idmo` из приведенного упрощенного кода операции `gather`.

Все нелокальные обращения происходят к массивам, индексированным точками согласования. Адреса всех таких обращений записаны в огромном индекс-векторе `idmo`, более того, все обращения к точкам сетки осуществляются только через данный индекс-вектор. Рассматривалось три варианта распределения: блочный, циклический и «двублочный».

Блочный вариант распределения точек согласования по узлам аналогичен распределению точек коллокации и является самым простым вариантом. Преимуществами данного метода является его простота реализации и некоторый учет шаблона обращений — большая часть обращений становится локальной, так как почти все обращения на диагональной линии становятся локальными. Недостатком данного распределения является значительный дисбаланс обращений от разных узлов: например, в первые узлы адресовано больше обращений, чем в другие узлы, из-за того, что первыми в нумерации точек согласования идут вершины (нижняя «толстая» прямая на графике).

Циклический вариант распределения (номер процессора определяется как остаток от деления индекса элемента на общее число процессоров) лишен недостатка дисбаланса входящих сообщений, но при этом полностью не учитывает локальность обращений — почти все обращения становятся удаленными, что увеличивает нагрузку на сеть, приводя к тому, что значительная часть времени уходит на коммуникации.

«Двублочный» вариант учитывает характер обращений к точкам согласования и особенность их нумерации, которая заключается в том, что первые N_{vertex} элементов содержат вершины всех элементов, а остальные элементы содержат точки на ребрах и гранях. Поэтому вершины и остальные точки независимо друг от друга разбиваются блочным образом, а i-узел локально хранит i-й блок вершин и i-й блок остальных точек. Этот вариант максимально сохраняет локальность обращений, но при этом сбалансирован по числу исходящих и входящих RMA.

Модификация «двублочного» варианта распределения точек согласования заключалась в том, что в i-м узле находились точки i-го блока вершин и (N-i)-го блока остальных точек, где N — число процессоров. Этот вариант в эксперименте показал лучшую производительность.

Для предоставления возможности гибкой реализации любого распределения точек согласования, в программу автором была встроена возможность выбора любого заданного пользователем распределения точек согласования. Достаточно реализовать лишь следующие функции: `redistributemor` — осуществляет вычисление констант распределения, `islocalmor` — определяет по глобальному индексу, является ли точка согласования локальной для данного процессора, `pemor` и `localmor` — определяют номер процессора и локальный индекс по глобальному индексу точки согласования.

Следует отметить, что каждые 5 шагов распределение будет изменяться, поскольку после адаптации сетки количество элементов изменится. Однако в силу гибкости кода это не составит никакой проблемы. Единственно, при распараллеливании адаптации сетки добавится несколько нелокальных обращений, так как часть элементов переедет с одних процессоров на другие. Но пока, мы решаем эту проблему с помощью сбора операцией `all2all` всех точек коллокации (точнее значения температуры на них) и выполнения адаптации сетки локально.

Следует отметить, что в таких языках, как UPC и системах с аппаратной поддержкой общей памяти (таких как Cray T3E/X1/X2/XMT) такой гибкости в распределении массивов по узлам не имеется. Как правило, аппаратно поддерживается лишь блочное, блочно-циклическое и циклическое распределение массивов по узлам, причем, как размер массива, так и шаг распределений может быть лишь степенью двойки и отсутствует возможность изменять распределение сегментов в ходе работы программы. Поэтому при реализации данной задачи придется считать распределение программно, что сделает всю это громоздкую поддержку аппаратной виртуальной сегментно-страничной памяти попросту ненужной. С другой стороны не удастся воспользоваться продуктивностью языков PGAS-класса, таких как UPC, также не обладающих гибкостью при распределении данных по узлам, что показывает достаточность таких средств как Cray SHMEM и CAF (Co-Array Fortran).

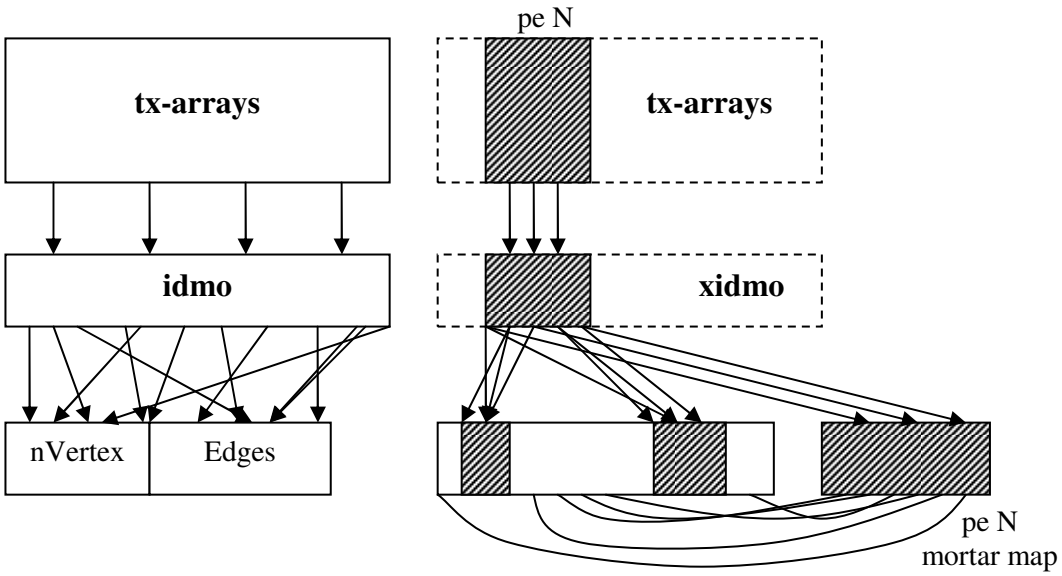
2.6. Пересчет индекс вектора

Как уже было указано, каждый процессор имеет свою локальную порцию массива `tx` и индекс-вектора `idmo`. Индекс-вектор `idmo` управляет внешними обращениями к ячейкам уда-

ленной памяти: каждая ячейка данного массива содержит индекс точки согласования, используемой при вычислении некоторого элемента массива `tx`.

Отметим, что количество элементов индекс-вектора несколько больше, чем количество элементов массива `tx`. Это отражает тот факт, что для вычисления значения температуры на точке коллокации используется не одна, а несколько соседних точек согласования. Подобное упрощение сделано лишь для простоты изложения и никак не влияет на предложенную схему распараллеливания.

Основной идеей является предлагаемый пересчет индекс-вектора с целью разделить локальные обращения и удаленные обращения, которые мы хотим выполнить перед общим циклом, реализуя парадигму DAE (Decoupled Access Execute). Для этого, каждый раз при изменении сетки, после формирования индекс-вектора (точнее той его части, что будет использоваться на локальном процессоре) мы выполняем ряд действий, которые призваны обнаружить все элементы, которые данный процессор собирается получить с других процессоров, и сохранить отсортированный список уникальных элементов в специальном массиве. Таким образом, формируется список точек согласования, находящихся на других узлах, значения которых необходимы для вычислений на локальном процессоре. Данный список будем называть картой доступа. Преимуществом нашего подхода является составление списка только из уникальных индексов, так как к каждому элементу в среднем происходит 6 обращений, а выполняя лишь одно удаленное чтение, мы значительно экономим на коммуникациях. С другой стороны, составление карты доступа приводит накладным расходам, однако следует учесть, что время подготовки карты доступа уменьшается с ростом числа узлов. К тому же, формирование карты происходит только один раз при изменении сетки, а используется она в каждой операции `gather/scatter`. Сортировка и выявление уникальных индексов выполняется с помощью отдельного кода, написанного на C++ и использующего стандартный шаблон `hash_table`.



а) б)

Рис. 3 – Доступ по индекс-вектору в последовательной версии (а) и доступ по измененному индекс-вектору к локальным и удаленным данным (1б) в PGAS-версии бенчмарка.

Далее карта доступа используется в начале каждой операции `gather`: в специальный массив `morloc` с помощью операции `shmem_get` подкачиваются значения температуры в точках согласования, находящихся на других процессорах, причем доступ к данному массиву на локальном процессоре выполняется регулярно и последовательно. После завершения функции синхронизации, гарантирующей получение всех запрошенных значений, выполняется главный цикл, который не содержит операций доступа к удаленным ячейкам памяти. С этой целью главный цикл был лишь незначительно модифицирован, все обращения к индекс вектору `idmo` были заменены на обращения к модифицированному индекс-вектору `xidmo`, который указыва-

ет на локальные элементы tmor-массива и на подкаченные с помощью построенной карты доступа элементы вспомогательного массива.

С учетом вышеизложенного, упрощенный код распараллеленной версии gather выглядит следующим образом:

```
      call shmem_barrier_all
c.....request outstanding tmor accesses to morloc array
      do i = 1,mormap_size
         pe = pemor(mormap(i))
         call shmem_double_g(morloc(i),tmor(localmor(mormap(i))),pe)
      end do
c.....gets synchronization
      call shmem_sync_gets

      do ie=startelt,endelt
         do j=1,125
          tx(i,j) = a(j,1)*readmor(xidmo(i,f(j,1)))
     &              + a(j,2)+readmor(xidmo(i,g(j,1))) + ...
       end do
      end do
```

Функция readmor заменяет все обращения к массиву tmor и содержит следующий код

```
      if(ig.gt.endmor) then
        readmor = morloc(ig-endmor)
       else
        readmor = tmor(ig)
       end if
```

Стандартным приемом программирования в стиле PGAS является замена подкачки с помощью операций shmem_get на посылку узлом-владельцем элементов с помощью операции shmem_put. На множестве реальных задач этот прием дает значительный выигрыш, так как снижает нагрузку на сеть, из-за того, что операция get посылает по сети два пакета - запрос и ответ, а операция put на многих системах реализована посылкой пакета только в одну сторону. Но для возможности такой замены каждому узлу придется составить обратную карту доступа, которая содержит номера узлов и номера ячеек в массиве morloc где ожидает данные удаленный узел. Есть два варианта составления такой обратной карты - вычислять ее полностью локально, или составить с помощью пересылок каждым узлом тех ячеек, которые содержатся в прямой карте доступа. Очевидно, что первый вариант не масштабируется, поэтому даже на небольшом числе узлов сильно проигрывает второму, который и был использован.

Таким образом, операция составления одной карты по коммуникациям эквивалентна одной операции gather. Эксперименты показали преимущество версии с put над версией с get в два раза, как и ожидалось. Другим плюсом стало увеличенная переносимость программы - для ее запуска на новой архитектуре теперь достаточно реализовать всего две функции shmem_put и shmem_barrier_all, если не считать функции инициализации (shmem_init, shmem_finalize, shmem_alloc, shmem_mype, shmem_numpes).

Альтернативой парадигме DAE при реализации операции gather могла бы стать реализация, которая прозрачно заменяет каждое чтение удаленного элемента массива операций shmem_get, однако из-за большой задержки на доступ к каждому элементу (несколько микросекунд) скорость выполнения цикла упала бы на порядок. Это было бы не так страшно, если бы вычислитель состоял из множества низкоскоростных ядер или тредов, обеспечивающих толерантность к коммуникационной задержке. Однако тренд в области HPC заключается в том, что современные суперкомпьютеры используют процессоры традиционной архитектуры с высокой производительностью одного треда, так как такие проекты, как Cray XMT не доказали своей применимости на широком классе задач, и остались узкоспециализированными системами, не применяемые в том числе и по сугубо экономическим причинам. Также, из-за отсутствия кэширования удаленной памяти в подобных nccNUMA системах, обращения к одной и той же удаленной ячейке выполнялись бы несколько раз, увеличивая расходы на коммуникации. Все это говорит о нецелесообразности аппаратной реализации nccNUMA систем, так как для реаль-

ных задач достаточно эффективной реализации иллюзии общей памяти на уровне эффективной реализации библиотеки односторонних коммуникаций, такой как Cray SHMEM.

2.7 Распараллеливание scatter

Операция `scatter` является обратной операцией к операции `gather`, и ее распараллеливание полностью аналогично, так как используется тот же индекс вектор `idmo`, а упрощенно код выглядит следующим образом:

```
do i=1,nelt
 do j=1,125
   tmor(idmo(i,f(j,1))) =   tmor(idmo(i,f(j,1))) + tx(i,j)*a(i,j)
   tmor(idmo(i,g(j,1))) =  tmor(idmo(i,g(j,1))) + tx(i,j)*b(i,j)
 end do
end do
```

Шаблон доступа scatter отличается от gather тем, что нужно выполнять изменение удаленных ячеек, а не только их чтение. Это вызвано тем, что значение температуры на одной точке согласования получается путем суммирования значений полученных в разных итерациях цикла, которые будут выполняться потенциально на разных процессорах. В OpenMP-версии такие конфликты по данным улаживаются с помощью блокировок `omp_lock` или секций `OMP_ATOMIC`. Это, в частности, одна из причин плохой масштабируемости OpenMP версии бенчмарка. В многоузловой версии также потребуются атомарное сложение с плавающей точкой, но поскольку поддержки таких операций на большинстве систем нет, придется эмулировать их через операции удаленной записи программно.

Использовалась довольно простая схема: в первой фазе каждый процессор выполняет все атомарные операции сложения с плавающей точкой, причем вместо ячеек находящихся на других узлах использует соответствующие ячейки массива `morloc`, предварительно обнуленные. После окончания всех итераций, содержимое массива `morloc` посылается с помощью операций `shmem_put` в специальный вспомогательный массив. После приема всех сообщений от других узлов, во второй фазе каждый узел выполняет локальные операции сложения полученных значений от различных узлов с соответствующими ячейками tmor-массива. Отметим, что на первой фазе используется прямая карта доступа, в то время как на второй итерации используется обратная карта. Также отметим, что количество коммуникаций сокращено в то же количество раз, что и для операции `gather`. Более того, объем коммуникаций в операциях `scatter` и `gather` абсолютно одинаковый. Различным является только направление посылок.

На Blue Gene/P можно было использовать возможность передачи активных сообщений с помощью функции `DCMF_Send`, однако это сделало бы программу непереносимой на другие системы, где такой функциональности нет. В дальнейшем планируются исследования эффективности применения `DCMF_Send` для выполнения произвольных атомарных операций.

2.8 Коллективные операции

Единственными коллективными операциями, которые потребовались при реализации бенчмарка UA, оказались `shmem_allsum_double` - суммирования по одному числу с плавающей точкой двойной точности от каждого узла, доставляя результат на каждый узел и `shmem_all2all` - для сборки вектора температуры на всех узлах перед выполнением адаптации сетки. Обе эти операции на Blue Gene/P были реализованы через специализированную сеть для коллективных операций. На других системах эти операции были реализованы через функцию `shmem_put` программно.

2.9 Гибридный режим PGAS/OpenMP

Заявленное в названии статьи расширение OpenMP заключается в том, что предложенная к использованию библиотека и парадигма программирования SHMEM подразумевает обмен ме-

жду различными узлами системы, а параллелизм же внутри одного узла может быть использован более эффективно с помощью парадигмы общей памяти OpenMP.

С точки зрения программы это означает, что распараллеливалась изначально не последовательная версия, а OpenMP версия бенчмарка, таким образом, все циклы будут распараллелены с помощью OpenMP автоматически. При этом требуется не вызывать функций shmem из параллельных секций (можно вызывать функции `shmem_put` и `shmem_get`, если обеспечить их выполнение в критических секциях).

К сожалению, даже оригинальная версия NPB-OMP дает на одном узле Blue Gene/P ускорение в 1.22 раза на четырех ядрах одного узла относительно производительности одного ядра, что свидетельствует о не самой лучшей реализации критических секций и семафоров OpenMP на данной архитектуре. В связи с этим, гибридная версия давала всего лишь 10 процентный прирост относительно чистой SHMEM-версии, использовавшей одно ядро на узел.

В то же время, например, при использовании 128 узлов и SHMEM-версии, VN режим дает ускорение 2.26 раза относительно SMP-режима. Единственной системой, где гибридный параллелизм дал ощутимый прирост, оказалась система МВС-Экспресс.

3. Результаты

На графике приводятся результаты, полученные на стратегическом суперкомпьютере IBM Blue Gene/P в зависимости от суммарно использованного количества ядер. Исследовались три режима работы комплекса Blue Gene/P: режим SMP при котором используется одно ядро на узле, режим DUAL - два ядра на узел, режим VN - четыре ядра в узле. Для запуска были доступны размеры от 128 до 1024 узлов выделяемых под задачу. Меньшее число узлов запустить на установленном в МГУ суперкомпьютере Blue Gene/P нельзя из-за малого количества IO-узлов системы.

Запускался класс C задачи, который описывается следующими параметрами: число элементов 33500 (число точек коллокации 4 млн.), число точек согласования 1.26 млн., максимальная глубина измельчения каждого элемента - 8 шагов. Объем памяти, занимаемой последовательной версией - 508 МБ. Параллельная версия требует 166 МБ на статические массивы (целочисленная информация о сетке) и 150 МБ PGAS-памяти на каждом узле. Основной потребитель памяти - массивы для программной реализации атомарных операций.

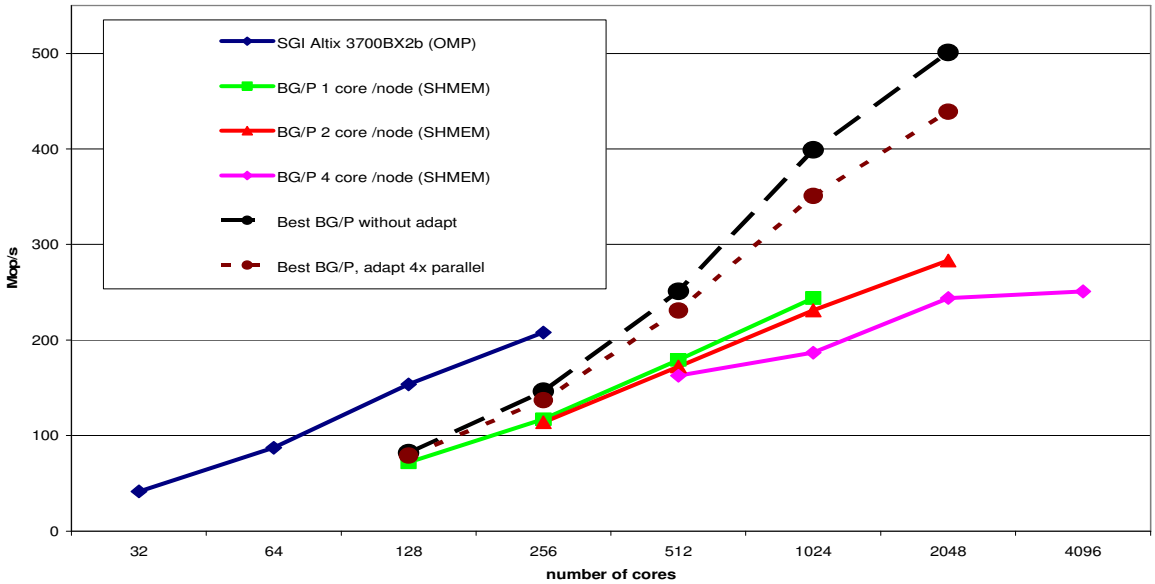


Рис.4 Абсолютная производительность NPB-PGAS UA (class C) на суперкомпьютерах IBM Blue Gene/P и SGI Altix 3700bx2.

Для сравнения приведены максимальные цифры производительности, полученные от NAS Advanced Supercomputing Division, полученные на OpenMP-версии на сравнительно старой ма-

шине с общей памятью SGI Altix 3700BX2b, 512 Itanium2, 1.6GHZ, 9MB L3, NUMALink 4. Максимальная производительность на 256 ядрах Itanium2 достигает 207 млн. операций в секунду. Под условной операцией в этом тесте понимается одна итерация метода сопряженных градиентов или интегрирование методом Рунге-Кутта. Такая единица измерения выбрана из-за неудобства оценки производительности операциями с плавающей точкой, так как ни scatter ни gather таких операций не содержат, и сравнение производительности запусков в FLOP/s при различном числе итераций метода CG было бы бессмысленным. В будущем, было бы интересно сравниться с результатами, полученными на новом поколении машин SGI Altix UV.

Из графика видно, что SMP и DUAL режимы близки по производительности при одинаковом количестве используемых ядер, в то время как VN режим несколько "проседает". Это согласуется с информацией о масштабировании SHMEM приведенной в разделе 2.1.

Максимальная производительность в 283 млн. операций в секунду достигается на 1024 узлах в режиме DUAL. На 4096 производительность падает из-за уже упоминавшегося проседания VN-режима, и из-за ограниченности задачи: в этом случае на один процесс приходится всего менее 8 элементов сетки и 315 точек согласования, а это слишком мелкий уровень гранулярности даже для такой машины как Blue Gene/P.

Следует отметить, что на большом количестве ядер значительную часть времени начинает занимать процедура адаптации сетки, которая автором в данное время распараллелена не была. Например, на 128 ядрах адаптация занимает 14 секунд из 122, а на 2048 ядрах процедура адаптации занимает 14 секунд из 31 общего времени счета задачи. На 1024 ядрах абсолютное ускорение достигает 100 раз, без учета времени адаптации получаемое ускорение составляет 171.

В дальнейшем планируется распараллелить адаптацию или на все узлы системы, или что было бы гораздо проще на 4 ядра одного узла. То есть каждый узел выполнял бы адаптацию независимо, но несколько ядер одного треда работали бы совместно. Поэтому на рисунке пунктиром обозначены две кривые, одна показывающая результат при нулевом времени адаптации, другая при 4х кратном ускорении времени адаптации, что показывает достижимость уровня производительности в 450-500 млн. операций в секунду на классе C.

3.2 Другие архитектуры, на которых была запущена NPB-PGAS UA

Кроме суперкомпьютера IBM Blue Gene/P, PGAS-версия бенчмарка была запущена на экспериментальном кластере из 6 узлов, на базе разработанного в НИЦЭВТ макета Ангара-М2 отечественной коммуникационной сети на базе ПЛИС, на экспериментальной системе МВС-Экспресс. На обычных SMP узлах и на vSMP системе запускалась OpenMP версия бенчмарка.

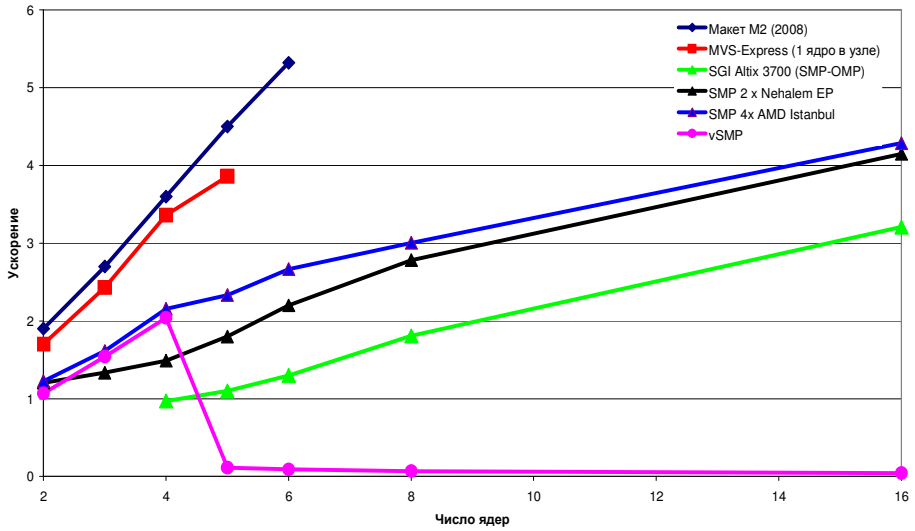


Рис.5 Ускорение на тесте NPB UA от числа используемых ядер.

Резкое падение производительности на vSMP-системе связано с тем, что до 4-х ядер задача работала на одном узле (с четырёхядерным процессором Intel Xeon QC), при большем же числе ядер были задействованы несколько узлов, связанных коммуникационной сетью Infiniband. При использовании нескольких узлов появляются частые промахи по виртуальным адресам, вызывающие исключения (page fault). В результате промаха на чтение одного слова (8 Б) в локальную физическую память подкачивается целая страница (4 КБ); таким образом, при нерегулярных обращениях к памяти накладные расходы на передачи по сети существенно превышают долю вычислений. В целом видно полное превосходство SHMEM-версии на системах, которые разрабатывались специально под SHMEM относительно OpenMP-версии.

4. Заключение

В заключение можно отметить, что применение парадигмы PGAS показало как высокую продуктивность, так и высокую эффективность даже на не имеющих аппаратной поддержки общей памяти стратегических машинах текущего поколения, таких как IBM Blue Gene/P. Будущие поколения суперкомпьютеров разработанных в рамках программы DARPA HPCS будут иметь еще большую производительность на данном классе PGAS-приложений.

Автору удалось на 2048 ядрах системы Blue Gene/P показать на 37 процентов лучший результат, чем максимально известный результат, который показывает OpenMP-версия на 256 ядрах SGI Altix 3700BX2B. Ожидаемый результат следующей версии, включающей распараллеленную версию адаптации сетки, превысит на 2048 ядрах результат Altix в более чем два раза.

Планируются следующие направления работ в дальнейшем: распараллелить адаптацию сетки, как в рамках одного узла, так и между всеми узлами, за счет этого и другой реализации атомарных с плавающей точкой радикально уменьшить потребление памяти и запустить класс D на Blue Gene/P, для чего потребуется расширить библиотеку SHMEM активными операциями. Также рассматриваются варианты ускорения реализации SHMEM на BG/P за счет экономии на подтверждениях отправки DCMF, а также реализовать агрегацию сообщений прозрачную для прикладного программиста. Также с помощью системы визуализации [4] планируется визуализировать работу метода на BG/P. Хорошо было бы провести измерения на стратегических суперкомпьютерах Cray серии XT5/XT6 (Baker в 2010 году) и на SGI Altix UV.

Автор выражает благодарности А.О. Лацису за полемически заостренную форму постановки вопросов и предоставленный доступ к коммуникационной системе МВС-Экспресс, факультету ВМиК МГУ за предоставленный доступ к единственному в России стратегическому суперкомпьютеру, изготовленному одной из двух суперкомпьютерных в мире компаний, H. Jin Haoqiang из NASA Advanced Supercomputing Division за предоставленные данные по производительности NPB-OMP UA на SGI Altix, компании Cray Inc. за разработку в 1991 гениальной и простой библиотеки Cray SHMEM.

Литература

1. Kumar, S., et al, The deep computing messaging framework: generalized scalable message passing on the blue gene/P supercomputer. // 22nd Annual international Conference on Supercomputing, June 07 - 12, 2008, Island of Kos, Greece. Proceedings, ACM. 2008. P. 94-103.
2. Moffett Field, H. Feng, R.F. Van der Wijngaart, R. Biswas Unstructured adaptive (UA) NAS Parallel Benchmark. NASA Ames Research Center, CA, 2004.
3. Лацис А.О. Вычислительная система МВС-Экспресс
URL: http://www.kiam.ru/MVS/research/mvs_express.html
4. Dzhosan O.V., Popova N.N., Korzh A.A. Hierarchical Visualization System for High Performance Computing // 14th International conference on Parallel Computing, September 1-4, 2009, Lyon, France, Proceedings.

Характеристики типовых алгоритмических структур

Вад.В. Воеводин

Существует предположение, что во многих предметных областях значительное множество задач построено на основе небольшого числа алгоритмических структур. Примерами таких структур могут служить перемножение матриц, скалярное произведение, БПФ или задача N тел. Поскольку типовые структуры лежат в основе многих задач, то, поняв, какими свойствами обладают эти структуры, можно будет сделать вывод о свойствах и характеристиках самих задач. В данной работе выделяются основные этапы процесса отображения задачи на некоторую аппаратную платформу и на каждом этапе описываются характеристики, которые определяют эффективность отображения. На примере типовой структуры «триада» показано влияние ее особенностей на получаемую эффективность отображения.

1. Введение

Существует предположение, основанное как на нашей собственной практике, так и на опыте других, что во многих предметных областях значительное множество задач построено на основе небольшого числа вычислительных структур [1-3]. В этих структурах сосредоточена основная вычислительная часть задачи. Подобные структуры мы будем называть типовыми алгоритмическими структурами. Примерами таких структур могут служить перемножение матриц, скалярное произведение, суммирование элементов массива, БПФ или задача N тел. Причем каждая из этих структур является типовой для некоторой определенной предметной области (или набора областей), например, БПФ является типовой структурой в области обработки сигналов.

Раз многие задачи построены на основе типовых структур, значит, поняв, как устроены типовые структуры, можно понять, как устроены и эти задачи. А это понимание чрезвычайно важно при построении эффективной реализации задачи на некоторой компьютерной платформе. Понимать, как устроена некоторая структура – это значит понимать, какими важными свойствами данные структуры обладают, какие характеристики их описывают, на какие параметры надо обращать внимание.

Задача состоит в нахождении подхода к формализации понятия типовой алгоритмической структуры, и после этого – в нахождении подхода к описанию алгоритмической структуры. Необходимо разработать методы исследования и эффективного отображения этих структур на конкретные классы параллельных систем и на конкретные архитектуры. В настоящий момент можно выделить следующие классы архитектур: реконфигурируемые процессоры – FPGA, графические процессоры – GPU, SMP-системы, кластеры и векторные машины. Эти параллельные платформы наиболее распространены на данный момент, однако нет желания ограничиваться только этими архитектурами, а хочется предусмотреть возможность добавления новых платформ, если таковые понадобятся.

2. Задача отображения

На рис. 1 приведена общая схема отображения алгоритма на некоторую платформу. Сверху расположены типовые структуры, снизу конкретные платформы. Хочется отработать путь сверху вниз, или, если более конкретно, хочется научиться решать, например, такие задачи:

1. Выявление причин неэффективного отображения типовых структур на некоторую платформу.
2. Упрощение процесса отображения в будущем при добавлении новых алгоритмических структур или новых платформ.
3. Оценка верхней границы степени эффективности отображения (в некоторой метрике) определенной структуры на определенную платформу.
4. Определение наиболее подходящей платформы для реализации конкретной структуры.

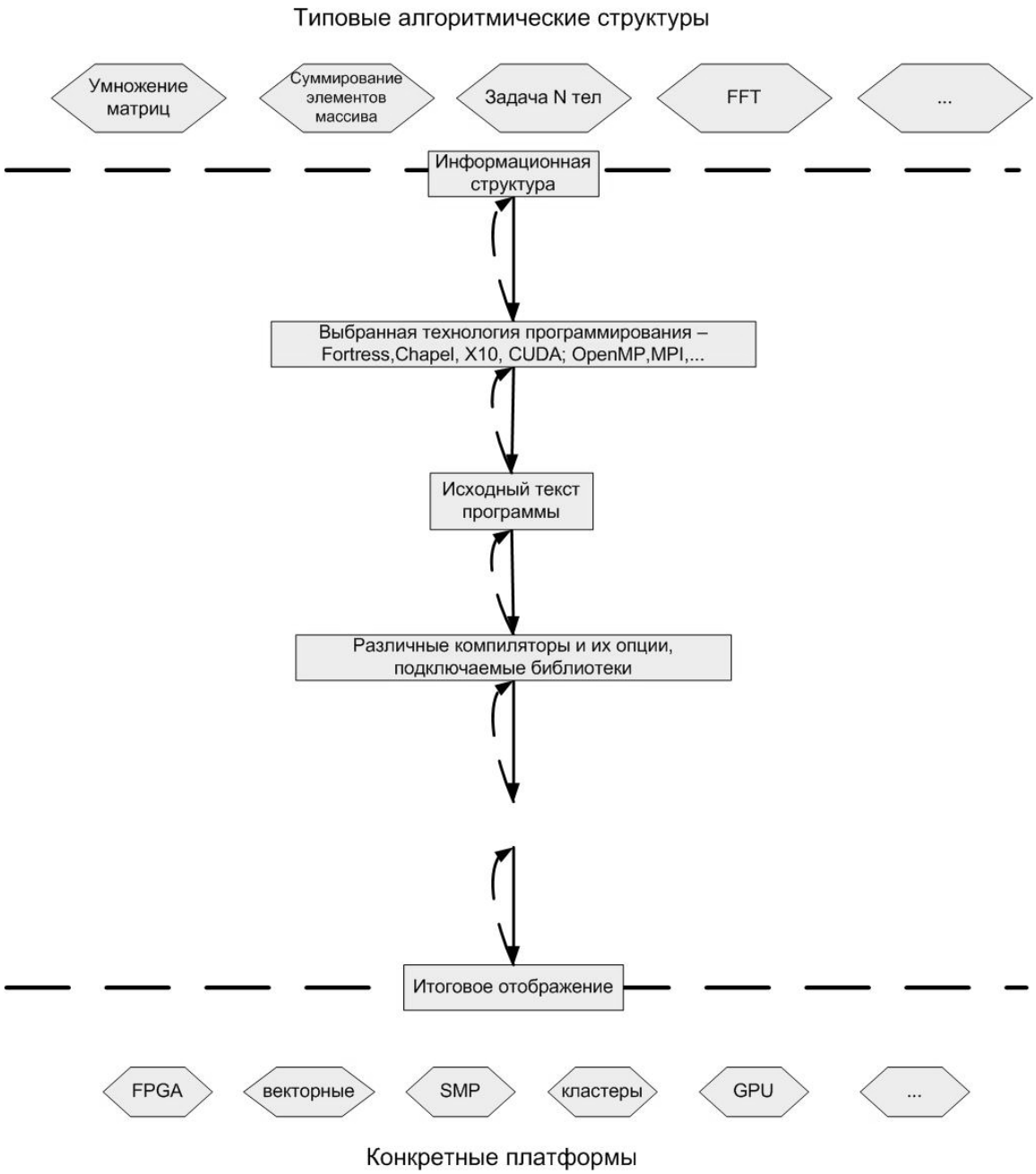


Рис. 1. Общая схема отображения типовых алгоритмических структур на вычислительные системы

Изучение процесса отображения структур на платформы позволит постепенно определять и фиксировать алгоритм отображения, поэтому в будущем некоторые шаги будут уже заранее определены и, возможно даже, будут выполняться автоматически.

Этапы процесса отображения типовой структуры на платформу, в общем, совпадают с этапами процесса написания обычной программы. Однако в данной работе важна не сама схема, отражающая последовательность шагов в процессе отображения алгоритма на архитектуру. Эффективность отображения – это является ключевым понятием. Мы хотим исследовать качество данного процесса, для чего нужно знать свойства и характеристики используемых в ней объектов: алгоритмов, программ и компьютерных платформ. Какие особенности архитектуры компьютера влияют на эффективность выполнения программы? По каким свойствам алгоритма можно судить о возможности его эффективного отображения на конкретную компьютерную

платформу? На эти вопросы нам поможет ответить исследование характеристик объектов, присутствующих на схеме отображения.

Исследуемые свойства и характеристики можно разделить на две группы: те, которые мы можем менять, и те, которые мы менять не можем. К первой группе можно отнести такие характеристики как число процессов или наличие точек синхронизации в программе, а примером из второй может служить характеристика, указывающая объем входных данных. Характеристики из первой группы определяют свободу при выборе отображения, и изменение значений этих характеристик приводит к различным вариантам реализации алгоритма. Характеристики из второй группы четко фиксированы и определяют потенциал эффективного отображения, который является показателем того, насколько хорошо в принципе можно отобразить алгоритм на выбранную платформу.

Указанную выше схему в более общем виде можно описать так: изначально задан алгоритм, который затем ложится в основу программы, а написанная и готовая к запуску программа затем исполняется на аппаратуре. Таким образом, в схеме можно выделить три основных составляющих – Алгоритм, Программа и Аппаратура. Каждая из этих составляющих обладает набором характеристик, которые могут влиять на эффективность реализации. Очень важным является то, что характеристики Алгоритма и Аппаратуры принадлежат ко второй группе, то есть не могут быть изменены, в то время как характеристики Программы могут меняться. Это означает, что эффективность отображения зависит только от того, насколько удачно будет сделан выбор значений характеристик Программы, который должен быть произведен с учетом характеристик двух других составляющих.

На каждом этапе изначально заложенный в алгоритм «потенциал отображения» (насколько хорошо он может быть распараллелен) может только уменьшаться, поэтому нужно выделить те характеристики алгоритма, которые влияют на эффективность отображения, а также научиться их оценивать и изменять для достижения лучших результатов. Невозможность повысить эффективность отображения при переходе от одного этапа к другому объясняется тем, что с каждым этапом происходит уточнение реализации алгоритма, все больше характеристик и параметров алгоритма четко фиксируется, и это накладывает все больше ограничений. Если рассматривать схему отображения, то на верхнем этапе схемы отображения задан только алгоритм и ничего более, поэтому и присутствуют только ограничения самого алгоритма. Затем задается технология программирования, в рамках которой необходимо писать программу, однако сама программа пока не зафиксирована. После написания программы становится гораздо меньше возможностей влиять на способ реализации начального алгоритма, поскольку он фиксируется в коде, и так далее. При этом очень часто нет возможности перейти к следующему этапу без потерь эффективности: например, при написании программы приходится заводить временные переменные и работать с ними, что естественно ведет к таким потерям. Поэтому чрезвычайно важно научиться понимать, какие свойства алгоритма на каждом этапе влияют на эффективность его реализации, и какие значения этих характеристик позволяют достигнуть наибольшей эффективности. Общей чертой всех этапов является то, что на каждом из них необходимо учитывать характеристики, которые получены не только на данном этапе, но и на всех предыдущих.

3. Описание характеристик различных этапов отображения

Перейдем к рассмотрению этапов отображения и выделенных в каждом из них характеристик. Общая схема приведена на рис. 2. Стоит отметить сразу, что данный набор характеристик не является окончательным и будет в дальнейшем дополняться.

Анализ информационной структуры алгоритма. Рассмотрим самый верхний уровень, когда известен только алгоритм, а, значит, известна и его информационная структура [4]. Для этого будем использовать граф алгоритма, который полностью отражает информационную структуру. Графом алгоритма называется ориентированный граф, в котором вершины – это множество всех операций алгоритма, и из одной вершины идет дуга в другую вершину тогда и только тогда, когда результат операции, вычисляемой в первой вершине, является аргументом операции, вычисляемой во второй вершине. Данный граф позволяет оценить целое множество полезных свойств, в частности, ресурс параллелизма алгоритма, его параллельную и последо-

вательную сложность, длину критического пути ярусно-параллельной формы графа, ширину и однородность ярусов, общее число операций [4]; объем входных/выходных данных, отношение числа операций к объему входных/выходных данных и другие.

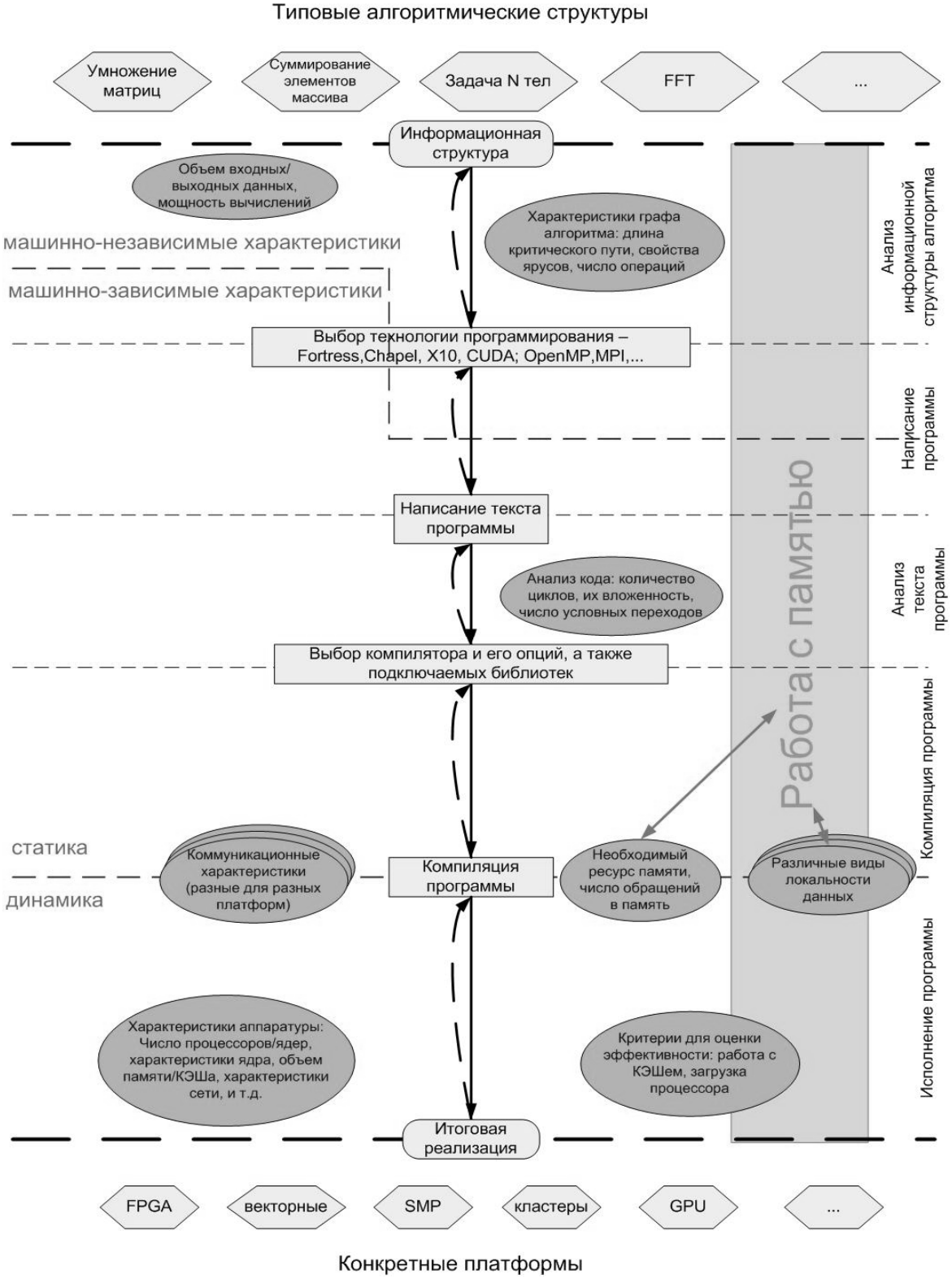


Рис. 2. Основные характеристики различных этапов отображения типовых алгоритмических структур

Помимо характеристик графа алгоритма, на данном этапе будут учитываться некоторые другие важные характеристики, такие как тип и разрядность данных, с которыми работает ал-

горитм. Разрядность важна, например, при работе с графическими процессорами, поскольку использование вещественных чисел с двойной точностью вместо одинарной часто существенно снижает скорость выполнения.

Стоит отметить, что большинство характеристик данного этапа относится к первой группе, то есть нельзя менять их значения, поскольку они являются свойствами исходного алгоритма. Однако характеристики графа алгоритма определяют потенциально возможный параллелизм данного алгоритма, верхнюю границу степени эффективности отображения. Они же могут помочь в принятии решения о том, по какому пути двигаться вниз по схеме отображения. К примеру, если мощность вычислений, то есть отношение общего числа операций к объему входных данных, слишком мала, то можно сделать вывод, что данный алгоритм вряд ли получится эффективно реализовать на графических процессорах, поскольку передача данных извне на графический процессор осуществляется медленно.

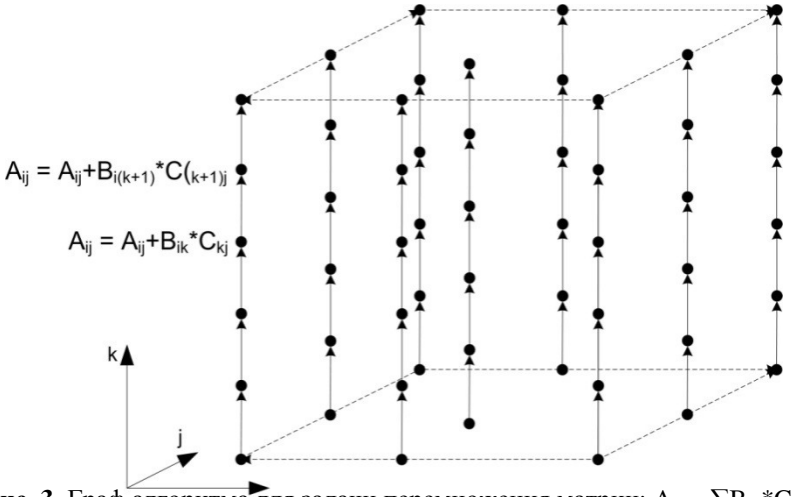


Рис. 3. Граф алгоритма для задачи перемножения матриц: $A_{ij} = \sum B_{ik}*C_{kj}$

На рис. 3 показан граф алгоритма умножения матриц размера NxN, который представляет собой куб со стороной N. Изучив граф, можно увидеть, что длина критического пути равна N-1 (каждая вершина графа – это одна операция сложения и одна операция умножения), ширина яруса постоянна и равна N^2, а общее число операций равно $2N^3$. Из полученных значений можно сделать вывод о том, что данный алгоритм можно разбить на не более чем N^2 параллельных ветвей вычислений, причем каждая из них будет состоять из 2N операций. Важной информацией является и то, что между разными ветвями отсутствует какая-либо передача данных.

После получения и анализа информационной структуры алгоритма желательно определить, на какую архитектуру исходный алгоритм будет отображаться, и на основании этой информации выбрать технологию программирования, поскольку не все технологии могут хорошо подходить для конкретной архитектуры.

Написание и анализ текста программы. После анализа графа алгоритма и последующего выбора технологии программирования наступает этап написания программы. Здесь накладываются новые ограничения на возможности отображения, поскольку необходимо программировать в рамках выбранной технологии.

Следующий этап – анализ текста программы. На данном этапе представляют интерес такие характеристики как: количество условных переходов и циклов в программе, уровень вложенности циклов; число объявленных переменных, и особенно массивов; характеристики параллельности программы – число нитей/процессов, каким образом они порождаются и т.д.; наличие в коде повторяющихся фрагментов программ (это может быть важно для реконфигурируемых процессоров); количество точек барьерной синхронизации. Такие характеристики как число циклов и уровень их вложенности, являются важными потому, что цикл является одним из основных источников параллелизма, и, следовательно, эти характеристики позволяют оценить потенциал распараллеливания в рамках написанной программы. В конце данного этапа определяются некоторые технические аспекты отображения, а именно происходит выбор наиболее

подходящего компилятора и его опций, а также определение подключаемых библиотек, если таковые требуются.

Компиляция и исполнение. Далее идет уровень, связанный с компиляцией программы и ее выполнением. Здесь анализируются основные характеристики, касающиеся работы с памятью (локальность данных, необходимый объем памяти, общее число обращений в память), а также коммуникационные характеристики, такие как коммуникационный профиль программы, объем передаваемых данных, неоднородность пересылок между процессами и т.д.

Одной из самых важных и интересных характеристик для рассмотрения является локальность данных. Локальность данных определяет, насколько близки последующие обращения в память. В зависимости от типа рассматриваемых объектов, выделяют локальность команд и локальность использования данных. Также различают пространственную и временную локальность. Пространственная локальность отражает среднее расстояние между несколькими последовательными обращениями в память; временная локальность показывает среднее число обращений по одному адресу в память за время исполнения всей программы. Анализ данных характеристик будет проводиться как минимум на двух уровнях, поскольку ясно, что большая часть информации о работе с памятью получается при выполнении программы, но часть информации можно извлечь и на этапе компиляции (можно оценить долю и профиль использования статических массивов).

Рассмотрим пример, показывающий влияние локальности обращений в память на эффективность программы. Возьмем обычную программу перемножения матриц:

```
for (i=0; i<n; i++)
    for (j=0; j<n; j++) {
        A[i,j] = 0.0;
        for (k=0; k<n; k++)
              A[i,j] = A[i,j] + B[i,k]*C[k,j];
        }
```

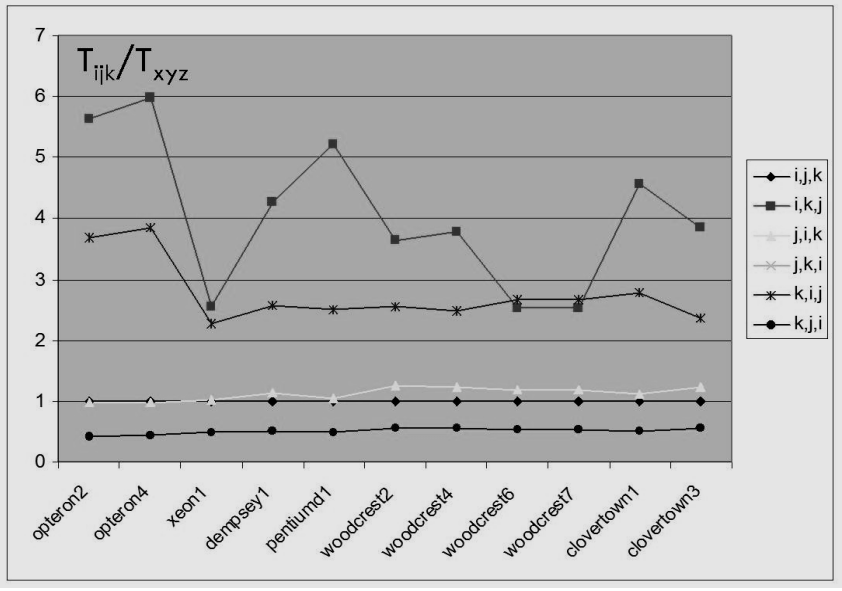


Рис. 4. Влияние локальности данных на эффективность реализации

Это самый очевидный способ реализации. Теперь единственное, что будем менять, – это порядок циклов. На рис. 4 приведены графики, которые соответствуют разным порядкам циклов в приведенном примере. По вертикали показано, во сколько раз время счета задачи при порядке циклов i,j,k больше (или меньше) времени счета задачи при других порядках циклов. Разница получается существенная, в самом лучшем варианте время счета уменьшается в 6 раз, и это только из-за перестановки циклов. Ускорение работы программы происходит именно из-

за улучшения локальности данных, и как следствие, более эффективного использования кэш-памяти.

Локальность использования данных является частью более общей характеристики – работы с памятью. Важным ее свойством, как и свойством некоторых других характеристик, является то, что они оказывают влияние почти на всех этапах отображения. Соответственно, оценивать их значение и влияние необходимо комплексно и, возможно, с применением отдельного, особого подхода.

Описание аппаратуры и оценка эффективности. Рассмотрим последний этап – выполнение программы. На этапе компиляции заканчивается часть процесса отображения, связанная со статикой, то есть с исследованием статических свойств алгоритма или программы, и на данном этапе начинается динамика – изучение программы во время ее исполнения. Здесь для нас важны аппаратные характеристики вычислительных платформ, для которых мы хотим найти эффективное отображение типовой алгоритмической структуры. Сюда входят такие характеристики как число процессоров/ядер, тактовая частота процессора, объем и структура ОЗУ и кэш-памяти, суперскалярность, характеристики коммуникационной сети и т.д. Важной особенностью подобных характеристик является то, что мы не можем изменять их значения, они только накладывают ограничения на потенциал эффективного отображения, и это приходится учитывать.

С указанными характеристиками тесно связаны критерии для оценки эффективности отображения. На их основе мы будем оценивать, насколько эффективно написана программа для данной платформы. Это такие критерии как загрузка процессора, работа с кэш-памятью, загрузка сети и т.д. Они будут сигнализировать о том, что эффективность где-то теряется, а для того, чтобы определить, в каком именно месте теряется эффективность, необходимо «подниматься» по схеме отображения по уровням вверх, чтобы понять, какую характеристику надо изменить, чтобы добиться большей эффективности. Например, некоторый критерий показывает, что происходит много кэш-промахов. Это значит, что в получившейся программе плохая локальность использования данных. Можно ли это исправить и как? Сначала исследуются характеристики на нижнем уровне, связанные с локальностью данных, однако причина не здесь, значит надо подниматься вверх; на этапе компиляции тоже все нормально; еще на уровень выше, на этапе написания программы видно, что надо было циклы написать в другом порядке (или использовать другие структуры данных). После внесения изменений в программе необходимо снова провести анализ характеристик эффективности.

4. Исследование эффективности реализации типовой структуры «триада»

Рассмотрим на конкретном примере влияние характеристик последнего уровня на эффективность реализации. Возьмем несколько вариаций типовой алгоритмической структуры «триада»:

A[i] = B[i]*X+C (1)
A[i] = B[i]*X[i]+C (2)
A[i] = B[i]*X+C[i] (3)
A[i] = B[i]*X[i]+C[i] (4)

Соответственно, переменные X и C в одних вариантах являются скалярами, в других – массивами. Также, будем рассматривать различные способы адресации массивов в этих вариантах – прямая и два варианта косвенной, с использованием дополнительного массива индексов ind. В первом варианте ind1[i] = i, во втором ind2[i] = (i*K)/N+(i*K)%N, где N – длина массива. Первый вариант помогает оценить накладные расходы, привносимые косвенной адресацией, а второй моделирует случай с плохой локальностью данных: константа K подбирается так, чтобы обеспечить максимальный процент кэш-промахов. При этом будем рассматривать самый простой, последовательный вариант решения этой задачи, в котором все исполняется только на одном ядре. При кажущейся простоте поставленной задачи ее анализ позволяет увидеть много интересного. Для реализации был выбран язык C (на ассемблере ничего не писалось), однако выбор языка не так важен в данном эксперименте, поскольку эффективность результатов рас-

сматривалась относительно друг друга. В качестве критерия эффективности использовалось отношение реальной производительности к пиковой.

Для изучения влияния аппаратных характеристик и получения реальных значений эффективности был проведен ряд экспериментов на различных системах. Информация о системах приведена в таблице 1.

Таблица 1. Описание систем, использованных в экспериментах с «триадой»

Название	Тип процессора	Частота процессора, GHz	Степень суперскалярности ядра	Частота FSB, MHz	Пиковая производительность, GFlops
Clovertown1	Xeon 5310	1,6	4	1066	6,4
Clovertown2	Xeon 5345	2,33	4	1333	9,32
Clovertown3	Xeon 5355	2,66	4	1333	10,64
Woodcrest1	Xeon 5150	2,66	4	1333	10,64
Woodcrest2	Xeon 5160	3,0	4	1333	12
Opteron1	Opteron 280	2.4	2	1000	4,8
Opteron2	Opteron 265	1.8	2	1000	3,6
Harpertown	Xeon E5472	3,02	4	1600	12

Основные аппаратные характеристики, которые могут оказывать влияние на эффективной последовательной реализации, следующие:

- Тактовая частота ядер
- Скорость шины FSB (системной шины)
- Частота оперативной памяти
- Объем ОЗУ на процессор
- Размер кэш-памяти разных уровней
- Суперскалярность
- Наличие векторных операций

Первые две характеристики сами по себе напрямую не определяют эффективность, однако их отношение является очень важным параметром, на который, как мы дальше увидим, нужно обращать внимание при рассмотрении эффективности реализаций. И вот почему: эффективность задачи определяется тем, насколько полно загружен процессор, однако скорость подачи данных по системной шине практически всегда меньше скорости обработки эти данных процессором, и значит, данные просто не будут успевать передаваться по шине процессору. Таким образом, узким местом является пропускная способность системной шины.

Взаимосвязь частоты процессора и частоты системной шины. Будем рассматривать следующую характеристику: $L=(F_{cpu}*s)/F_{fsb}$, где F_{cpu} – частота процессора, s – степень суперскалярности, F_{fsb} – частота системной шины. Рассмотрим вариант (1) типовой структуры «триада». В этом варианте идеальным является значение L=1, поскольку в таком случае системная шина перестает быть узким местом (можно и меньше 1, но к лучшим результатам это не приведет, поскольку узким место станет процессор, и эффективность остановится на уровне эффективности самого процессора). Однако для многих систем данное значение далеко от идеального. Рассмотрим связь характеристики L с эффективностью реализации нашей задачи. Для этого сначала посчитаем эффективность для каждой из систем, а затем посмотрим, как соотносятся значения эффективностей для различных систем и их значения характеристики L. Для этого мы считаем отношение эффективностей, деленное на обратное отношение характеристик L. Соответственно, чем ближе полученное значение к 1, тем ближе зависимость рассмотренных двух систем от параметра L друг к другу. Назовем этот отношение R, а эффективность будем обозначать EFF_i, где i – название системы. Приведенные отношения показаны на рис. 5: для каждой системы указано значение отношения R относительно системы clovertown1. Можно видеть, что для всех рассмотренных систем это значение близко к 1, и значит, для этих систем имеет место практически прямая зависимость эффективности от значения L. Это говорит о том, что измене-

ние эффективности на рассматриваемых процессорах определяется практически исключительно изменением характеристики L, что показывает ее важную роль в вопросах эффективности. Конечно, в других задачах эта зависимость может прослеживаться не столь явно, если обращения в память в них происходят не столь часто.

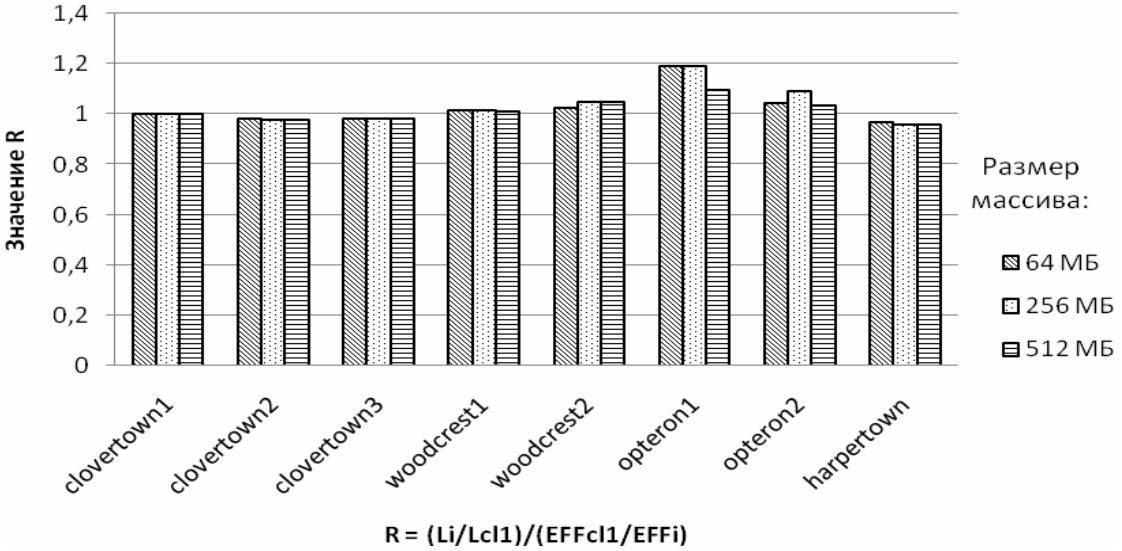


Рис. 5. Влияние характеристики L на эффективность реализации структуры «триада»

Из полученного вывода можно сделать интересные следствия. Рассмотрим эффективность для систем с процессорами из одной линейки – Xeon (рис. 6). У процессоров clovertown2 и clovertown3, а также у обоих процессоров woodcrest одинаковая частота системной шины. При этом увеличение тактовой частоты при переходе от одного к другому не приводит к какому-либо реальному ускорению работы программы, однако увеличивает значение пиковой производительности. Это в свою очередь приводит к обратно пропорциональному уменьшению эффективности программы. То есть, используя более мощный процессор, мы получаем уменьшение эффективности! Похожий эффект можно наблюдать и у процессора harpertown – у него и тактовая частота выше, и системная шина быстрее, однако отношение L не в его пользу: у harpertown это значение равно 7,55, а у clovertown2 – 6,99. Наилучшее значение характеристики L из рассматриваемых систем у clovertown1 – 6, что и выражается в наибольшей эффективности. При этом отношение эффективностей с точностью до сотых долей совпадает с отношениями их характеристик L.

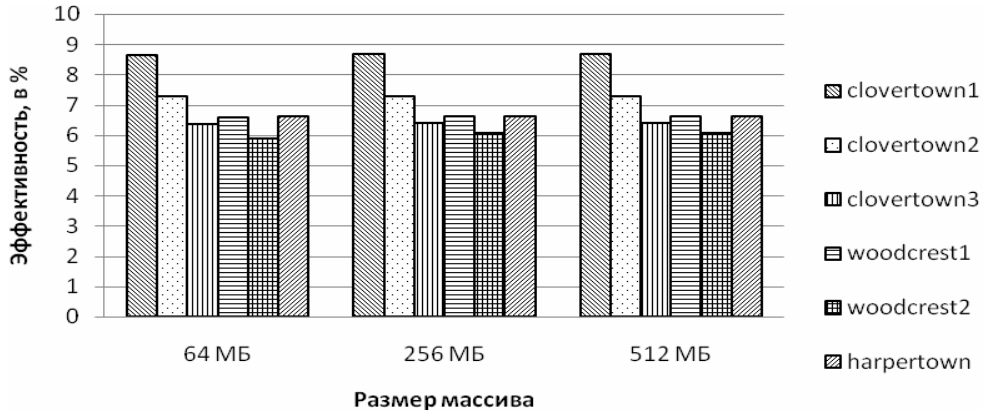


Рис. 6. Эффективность задачи для процессоров линейки Xeon

Еще одним интересным наблюдением является то, что эффективность данной задачи практически не зависит от объема входных данных – при увеличении размера массива эффективность остается практически на том же уровне. И это логично, поскольку работа с памятью происходит по одному сценарию, и кэш-память работает с одинаковой эффективностью – при занесении новой строки в кэш все ее элементы последовательно используются в программе. Важным замечанием является то, что поскольку пропускная способность системной шины у всех рассмотренных систем ниже, чем скорость, с которой оперативная память способна обрабатывать запросы на чтение/память, характеристики ОЗУ не влияют на эффективность реализаций. Все это приводит к тому, что такие обычно важные параметры как размер кэш-памяти разных уровней, размер и частота ОЗУ, в данной задаче практически не важны.

Кэш-промахи. Теперь рассмотрим примеры, когда работа с кэш-памятью происходит по разным сценариям. Для этого прогоним все 4 варианта задачи со всеми 3-мя способами адресации, и посмотрим изменения к эффективности. Соответственно, для каждой системы у нас будет 3 группы значений, по 4 в каждой. Динамика изменений для всех систем похожа, поэтому мы будем говорить безотносительно к какой-либо конкретной системе.

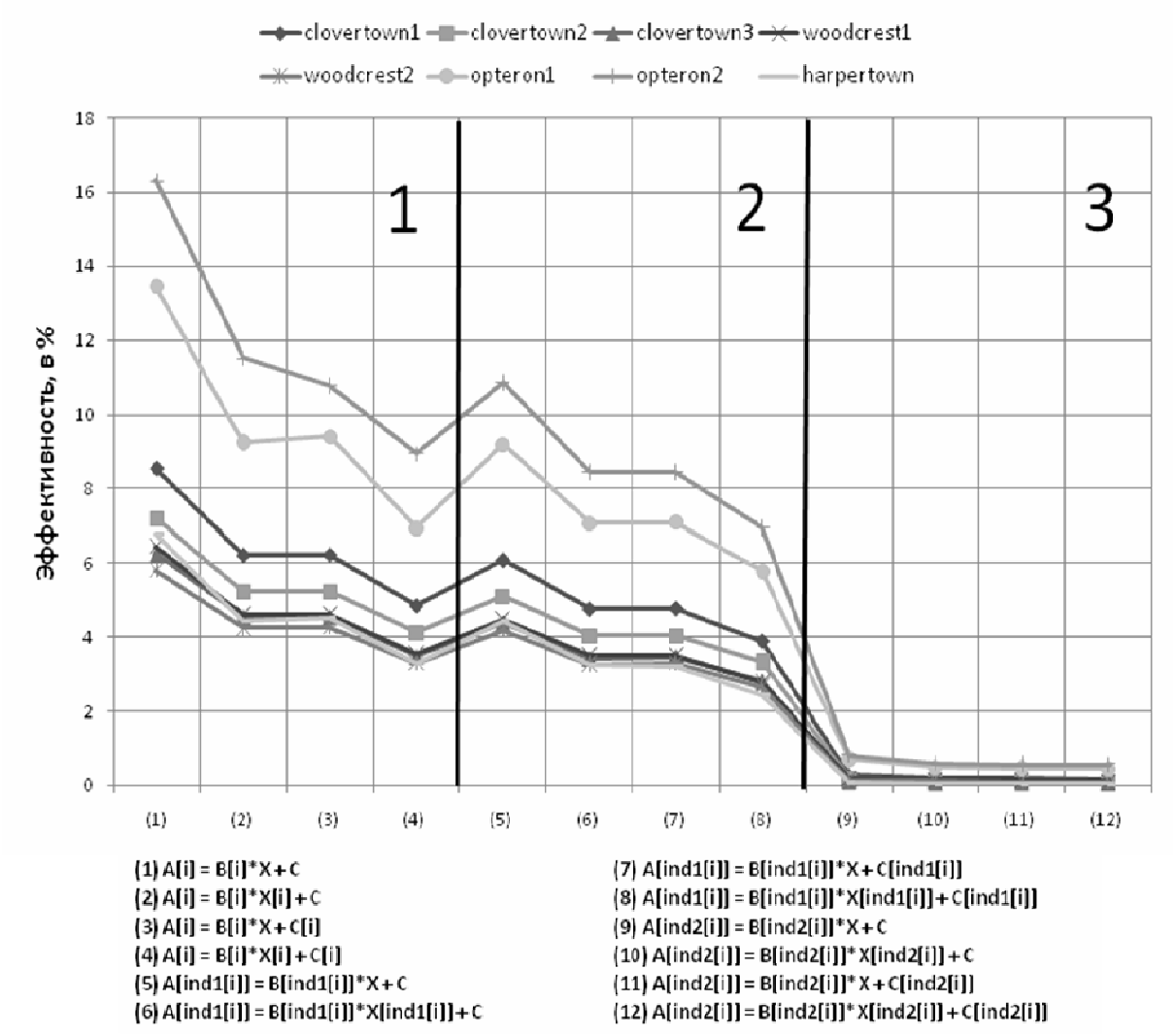


Рис. 7. Влияние степени локальности данных на эффективность реализации «триады»

В первой и второй группе результатов кэш-попадание происходило часто, поскольку выборка данных идет последовательная. Основным их отличием между собой является наличие дополнительного массива индексов ind1 – эффективность экспериментов (2), (3) и (5) практически равны, как и экспериментов (4), (6), и (7). Однако в третьей группе экспериментов эф-

фективность падает катастрофически, достигая значений менее 0.2% – кэш-промахи образуются практически при каждом обращении в память. Данный пример говорит о том, насколько важной характеристикой является локальность использования данных для оценки эффективности реализации.

5. Заключение

В данной статье рассмотрены некоторые теоретические и практические аспекты процесса выделения характеристик типовой алгоритмической структуры и оценки с их помощью эффективности реализации. На простом примере вычислительного ядра «триада» показано влияние этих характеристик. Этот пример также показывает, что эффективность и простой задачи может зависеть от довольно большого числа различных характеристик, и эта зависимость будет становиться все только сложнее с рассмотрением реальных больших задач.

Литература

1. В.В. Воеводин «Вычислительная математика и структура алгоритмов». – М: Изд-во МГУ, 2006. 112 с.
2. Asanovic, K. et al. The Landscape of Parallel Computing Research: A View from Berkeley. UCB/EECS-2006-183, University of California, Berkeley, Dec. 18, 2006.
3. Asanovic, K. et al. A View of the Parallel Computing Landscape. Communications of the ACM 52, 10 (Nov. 2009), 56-67.
4. В.В. Воеводин, Вл.В. Воеводин «Параллельные вычисления» - СПб: БХВ-Петербург, 2002. 608 с.

Исследование алгоритмов планирования параллельных задач для кластерных вычислительных систем с помощью симулятора*

П.Н. Полежаев

В данной работе описываются результаты экспериментального исследования различных алгоритмов планирования задач для вычислительного кластера, полученные с помощью программного симулятора вычислительного кластера и его управляющей системы. Для проведения исследования предложена модель вычислительной загрузки кластера, разработана его имитационная схема, а также построены критерии и метрики сравнения алгоритмов планирования.

1. Введение

В настоящее время благодаря дешевизне серийно выпускаемого аппаратного обеспечения и возможности его гибкого конфигурирования кластерные вычислительные системы получили большое распространение. Они используются для решения вычислительно емких задач в различных отраслях человеческой деятельности. Однако, даже не очень производительные кластерные системы весьма дороги, и, чтобы окупить затраты на их покупку, необходимо максимально рационально использовать доступные вычислительные мощности. Это может быть сделано за счет применения эффективных алгоритмов планирования параллельных задач, поступающих в управляющую систему вычислительного кластера.

Алгоритмы планирования, используемые в существующих управляющих системах, обладают целым рядом недостатков. Прежде всего, они не в состоянии обеспечить высокую вычислительную загрузку узлов кластера. Часто возникает ситуация, когда есть свободные вычислительные узлы, в то время как в очереди ожидает своего исполнения достаточно большое количество задач. Более того, они не всегда рассматривают случай, когда вычислительные узлы имеют неоднородный состав, т.е. когда они отличаются объемами доступной оперативной и дисковой памяти, производительностью процессоров. Также не во всех управляющих системах учитывается возможность наличия локальной загрузки узлов, которая характерна для кластеров рабочих станций.

Современные исследования [1–4], рассматривающие различные списочные алгоритмы планирования, не достаточно полные, т.к. не затрагивают всех основных существующих алгоритмов или рассматривают достаточно ограниченное количество анализируемых метрик. Чаще всего это зависимости среднего времени ожидания в очереди и использования процессорных ресурсов от загрузки системы потоком работ. В источниках практически не затрагиваются не менее важные метрики сбалансированности, гарантированности и честности, а также не достаточно полно изучается поведение алгоритмов планирования на кластерах рабочих станций, на кластерах со значительной степенью гетерогенности аппаратной платформы.

Данная работа призвана попытаться частично исправить сложившуюся ситуацию. В нашем исследовании проводится достаточно полный сравнительный анализ всех основных списочных алгоритмов планирования с помощью широкой системы критериев и метрик. Она охватывает все аспекты эффективности составляемых расписаний запуска задач: производительность, используемость памяти узлов, сбалансированность их загрузки, гарантированность обслуживания задач, а также честность алгоритма планирования по отношению к задачам.

В настоящем исследовании рассматриваются четыре сценария: гомогенный кластер при наличии или отсутствии локальной загрузки узлов и гетерогенный кластер при наличии или отсутствии локальной загрузки вычислительных узлов.

* Исследования выполнены при поддержке Федерального агентства по образованию в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг. (государственный контракт №П2039).

Для целей данной работы разработана модель вычислительной загрузки кластера, построена его имитационная схема, которая была реализована в виде программного симулятора работы вычислительного кластера. Он использовался для количественной оценки эффективности работы алгоритмов планирования на реалистичных рабочих загрузках кластера потоком параллельных задач и имитации локальной загрузки вычислительных узлов.

Использование симулятора по сравнению с реальным кластером дает финансовую выгоду, уменьшает дополнительные трудозатраты (например, нет необходимости обеспечивать отказоустойчивость, безопасность или составлять набор реальных задач для эксперимента), снижает время исследования, а также обеспечивает гибкую (возможность изменения аппаратной конфигурации) и контролируемую (отсутствие аппаратных и программных сбоев) среду вычислений. Кроме того, симулятор обеспечивает большую реалистичность результатов по сравнению с использованием метода аналитического анализа моделей, для которого характерно наличие значительного числа упрощений и предположений.

2. Имитационная схема и модель кластера

На рисунке 1 приведена имитационная схема управляющей системы кластера, включающая два источника: $И_1$ формирует поток параллельных задач, отправляемых пользователями в управляющую систему вычислительного кластера, $И_2$ генерирует локальную загрузку узлов. Не ограниченный по длине, накопитель $Н_1$ представляет собой очередь и используется для хранения заявок на выполнения задач, поступающих от источника $И_1$. Канал $К_0$ представляет собой выделенный управляющий узел, извлекающий из $Н_1$ подходящую задачу и распределяющий ее, в соответствии с заложенным алгоритмом, на требуемое количество доступных вычислительных узлов – подмножество свободных каналов $К_1$, $К_2$, …, $К_m$.

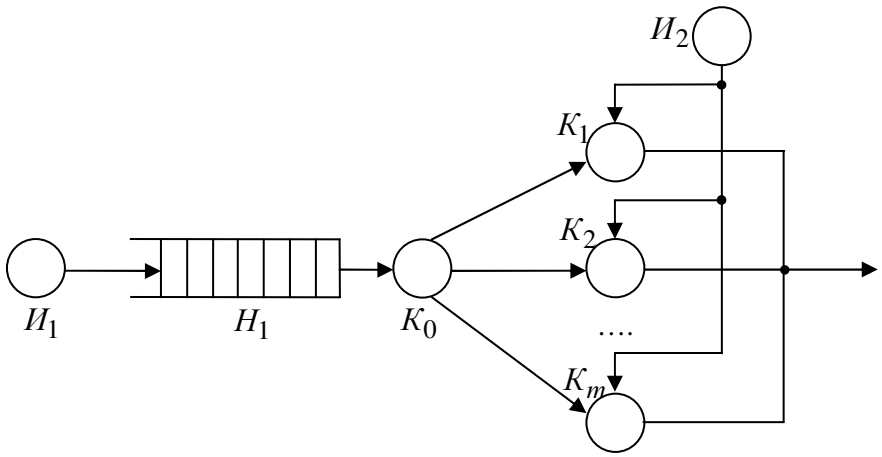


Рис. 1. Имитационная модель СПО кластера

Каждый вычислительный узел K_i имеет M_i Кбайт оперативной памяти и D_i Кбайт дисковой памяти за вычетом операционной системы и предустановленных программ, он также характеризуется относительной вычислительной скоростью S_i, которая показывает во сколько раз данный узел быстрее самого медленного в кластере. Кроме того, узел K_i описывается следующими динамическими параметрами: $m_i(t)$, $d_i(t)$ и $u_i(t)$ - соответственно значения доступной оперативной памяти, дисковой памяти и загруженности процессора i-го узла в момент времени $t \in [0;\infty)$.

В рамках данной модели не учитывается топология вычислительного кластера, многопроцессорность вычислительных узлов, а коммуникационные задержки, возникающие при передаче данных между процессами, исполняющейся параллельной программы, учитываются неявно и закладываются в ее время выполнения.

Данная имитационная схема и модель легли в основу симулятора вычислительного кластера и его управляющей системы, используемого для исследования алгоритмов планирования задач.

3. Модель вычислительной загрузки кластера

Вычислительная загрузка кластера формируется из потока задач, отправляемых пользователями в его управляющую систему, и локальной вычислительной загрузки узлов, формируемой локальными приложениями, выполняемыми на узлах.

Пусть $J = (J_1,...,J_n)$ - поток (последовательность) задач, упорядоченная по возрастанию времени их прибытия $a_j \in [0;\infty)$. Данный поток включает в себя все задачи, поступающие от пользователей в управляющую систему вычислительного кластера. Отдельно заметим, что алгоритм планирования, используемый в управляющей системе вычислительного кластера, знает только о тех задачах, которые попали в очередь до текущего момента времени.

Пользователь для задачи J_j указывает следующие требования к ресурсам кластера: n_j - количество узлов, mr_j - объем оперативной и dr_j - объем дисковой памяти в килобайтах на каждом узле. Также каждая задача характеризуется, выполняемой пользователем, оценкой времени выполнения $\tilde{p}_j$, осуществляемой, при условии, что все n_j узлов будут иметь единичные вычислительные скорости $S_i = 1$. Пусть N_j - множество узлов, выделенных планировщиком j-й задаче, тогда оценка времени ее выполнения с учетом различной скорости выделенных узлов будет равна $p_j = \dfrac{\tilde{p}_j}{\min\limits_{i \in N_j} S_i}$.

Кроме описанных выше характеристик, каждая задача также имеет параметр $\tilde{\tau}_j \in (0; p_j]$ – ее реальное время выполнения на узлах единичной скорости. Аналогично определяется реальное время выполнения задачи с учетом различных скоростей назначенных ей планировщиком узлов: $\tau_j = \dfrac{\tilde{\tau}_j}{\min\limits_{i \in N_j} S_i}$.

При планировании с использованием приоритетов для j-й задачи приоритет w_j либо задается пользователем, либо определяется непосредственно алгоритмом планирования.

Симуляция работы вычислительного кластера и его управляющей системы предполагает генерацию потока задач и локальной загрузки вычислительных узлов. Параметры задач и локальной вычислительной загрузки представляют собой случайные величины, имеющие определенные законы распределений, подбираемые на основе анализа реальных трасс, собираемых на кластерных вычислительных системах.

Анализ существующих моделей потоков задач [5–9] показывает, что они рассматривают небольшое количество параметров (чаще все только a_j, n_j, t_j и p_j) и требуют усовершенствования за счет учета требований к оперативной и дисковой памяти вычислительных узлов.

Далее коротко опишем используемые в настоящем исследовании законы распределений для каждого параметра, характеризующего задачу.

Требуемое количество процессоров n_j. Все задачи потока можно сгруппировать в три класса [9]: последовательные задачи, 2^k-задачи (параллельные задачи с n_j равным степени числа 2) и остальные. Генерация значений n_j (см. рисунок 2) осуществляется следующим образом. С вероятностью q_1 задача будет последовательной ($n_j = 1$), иначе для параллельной задачи выбирается согласно заданному закону распределения число x, являющееся логарифмом n_j. С вероятностью q_2 оно будет округлено до целого (в этом случае получим 2^k-задачу),

иначе получим параллельную задачу с произвольным числом требуемых процессоров. Для симуляции использовались следующие значения данных параметров [9]: $q_1 = 0.2$ и $q_2 = 0.325$. За основу для генерации значения n_j было взято двухэтапное равномерное распределение с параметрами [9]: $a = 1$ (наименьшее возможное значение), $b = \log_2 m$ (максимальное возможное значение), $c = \log_2 m - 1.5$ и $p = 0.3$.

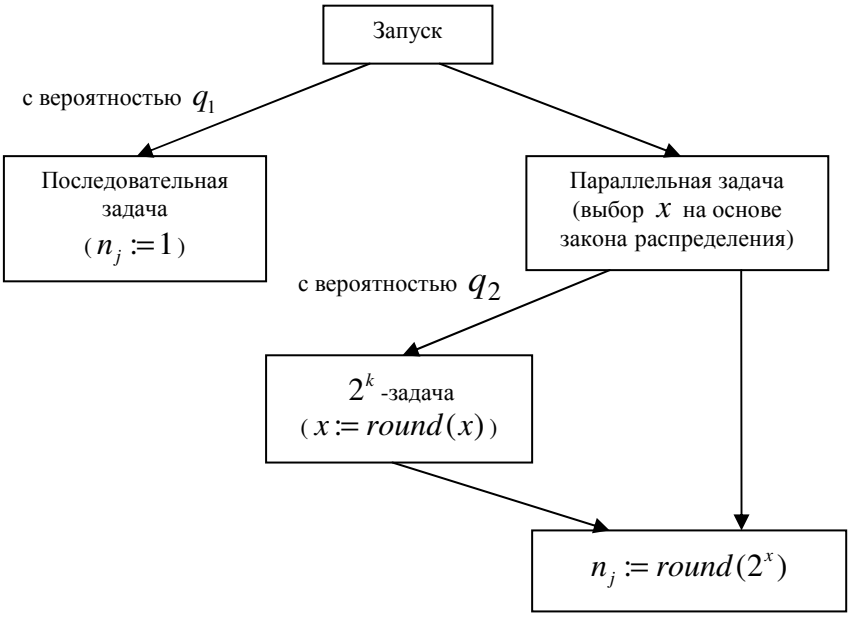


Рис. 2. Алгоритм генерации требуемого количества процессоров

Реальное время выполнения задачи τ_j. В данной работе используется подход, описанный в [9]. Логарифм реального времени выполнения задачи имеет гипер-гамма распределение, параметр p которого линейно зависит от n_j. При исследовании использовались следующие усредненные значение параметров распределения: $\alpha_1 = 6.57$, $\beta_1 = 0.823$, $\alpha_2 = 639.1$, $\beta_2 = 0.0156$, а линейная зависимость описывалась с помощью формулы $p = -0.003 n_j + 0.6986$.

Оценка пользователя времени выполнения задачи $\tilde{p}_j$. В данной модели будет использован подход, описанный в работе [3], согласно которому $\tilde{p}_j$ – равномерно распределенная величина в интервале $[\tau_j; f\tau_j]$, где $f \in [1;4]$ – константный показатель, отражающий неточность оценок пользователей. Для исследования было принято значение $f = 2$.

Время поступления a_j задачи в управляющую систему. Времена прибытия задач имеют распределение Пуассона [2, 5, 9], следовательно, интервалы времени между прибытиями подчиняются экспоненциальному закону распределения. В качестве параметра интенсивности может было взято значение $\lambda = \frac{1}{15}$ [7].

Требования к оперативной и дисковой памяти узлов. Данный способ генерации предложен нами и опирается на равномерное распределение. Пусть $Q_1 < Q_2 < ... < Q_k$ – отсортированный по возрастанию список значений объемов оперативной памяти на узлах, в котором исключены повторяющиеся значения, $W_1 < W_2 < ... < W_s$ – аналогичный список для объемов дисковой памяти узлов. Пусть g_{ij} - количество узлов кластера, у которых $M_l \geq Q_i$ и $D_l \geq W_j$ ($i = \overline{1,k}$ и $j = \overline{1,s}$). Тогда алгоритм генерации требований l-й работы к оперативной и дисковой памяти узлов будет следующим:

1. Если $n_l < \left\lceil \frac{m}{2} \right\rceil$, то $n_l := \left\lceil \frac{m}{2} \right\rceil$.

2. $I := \{\}$ – множество пар индексов вначале пусто.

3. Для каждого $i = \overline{1,k}$ найти такое наибольшее j, чтобы $g_{ij} \geq n_j$. Если для данного i такое j найти нельзя, то переходим к следующему значению для i, иначе сохраняем пару найденных индексов: $I := I \cup \{(i, j)\}$.

4. Выбрать случайно и равновероятно упорядоченную пару индексов (i, j) из I.

5. Выбрать произвольное значение mr_l (равномерное распределение) из интервала $[mr_{\min}; Q_i \cdot \alpha_m]$, где $mr_{\min}$ – минимальный объем требуемой оперативной памяти, $\alpha_m \in [0;1]$ – доля оперативной памяти узла, которая может быть занята работой.

6. Выбрать произвольное значение dr_l (равномерное распределение) из интервала $[dr_{\min}; W_j \cdot \alpha_d]$, где $dr_{\min}$ – минимальный объем требуемой дисковой памяти, $\alpha_d \in [0;1]$ – доля дисковой памяти узла, которая может быть занята работой.

Шаг 1 необходим для того, чтобы гарантировать, что любая задача может быть потенциальна запущена хотя бы на половине узлов кластера. Для симуляции использовались следующие значения параметров: $\alpha_m = \alpha_d = 0.70$ и $mr_{\min} = md_{\min} = 1024$ Кб.

Приоритет задачи w_j. Для генерации приоритета w_j задачи можно выбрать различные распределения. В данной модели будет использоваться равномерное распределение в интервале $[1;100]$.

Локальная вычислительная загрузка узлов (рабочих станций) формируется за счет программ, запускаемых их локальными пользователями. Непосредственное моделирование запуска программ, их диспетчеризации планировщиком локальной ОС узла чрезмерно бы усложнило разрабатываемую модель, при этом нет необходимости в такой точности получаемых результатов. Поэтому нами был предложен описанный далее подход.

Пусть $\psi_{mi}(t)$, $\psi_{di}(t)$ и $\psi_{ui}(t)$ – величины локальной загруженности соответственно оперативной памяти, дисковой памяти и процессора i-го узла в момент времени t. Причем, для любого узла i в любой момент t времени данные значения лежат в диапазоне от 0 до 1.

Величины $\psi_{mi}(t)$, $\psi_{di}(t)$ и $\psi_{ui}(t)$ имеют гамма распределения с соответствующими параметрами (α_m, β_m), (α_d, β_d) и (α_u, β_u).

Пусть ΔT_m, ΔT_d и ΔT_u – случайные величины, характеризующие интервалы времени, через которые происходит изменение соответствующих величин $\psi_{mi}(t)$, $\psi_{di}(t)$ и $\psi_{ui}(t)$. Они имеют экспоненциальный закон распределения с соответствующими параметрами λ_{m}, λ_{d} и λ_{u} (интенсивностями). Причем, логично предположить, что значения локальной загрузки процессора меняются интенсивнее значения доступности оперативной памяти, а оно, в свою очередь, интенсивнее значения доступности жесткого диска.

Значение загрузки процессора, а также доступной оперативной и дисковой памяти i-го узла в момент времени t определяется по формулам (1)-(3):

$$u_i(t) = \begin{cases} 1, & \text{если на } P_i \text{ выполняется } j\text{-ая работа в момент } t, \\ \psi_{ui}(t), & \text{иначе,} \end{cases} \quad (1)$$

$$m_i(t) = \begin{cases} M_i - (M_i - mr_j)\psi_{mi}(t) - mr_j, & \text{если на } P_i \text{ выполн. } j\text{-ая работа ,} \\ M_i - M_i\psi_{mi}(t), & \text{иначе,} \end{cases} \quad (2)$$

$$d_i(t) = \begin{cases} D_i - (D_i - dr_j)\psi_{di}(t) - dr_j, & \text{если на } P_i \text{ выполн. } j \text{ - ая работа}, \\ D_i - D_i\psi_{di}(t), & \text{иначе.} \end{cases} \quad (3)$$

Для целей симуляции были выбраны следующие значения параметров: $\alpha_m = 0.2$, $\beta_m = 0.3$, $\alpha_d = 0.2$, $\beta_d = 0.2$, $\alpha_u = 0.5$, $\beta_u = 0.5$, $\lambda_{\text{u}} = 0.08$, $\lambda_{\text{m}} = 0.04$ и $\lambda_{\text{d}} = 0.02$.

В данном разделе была описана модель вычислительной загрузки кластера, используемая в процессе симуляции работы вычислительного кластера и его управляющей системы.

4. Исследуемые алгоритмы планирования задач

Составление оптимальных расписаний запуска работ для кластерной вычислительной системы относится в общем случае к классу NP-полных задач [10–13], требующих экспоненциального времени решения, что неприемлемо для планировщика, т.к. он должен инициировать цикл планирования каждый раз при наступлении в системе некоторого события – поступления в очередь новой задачи, завершении исполняющейся задачи или изменении локальной загрузки вычислительного узла. В связи с этим для планирования задач на кластере используются различные эвристические алгоритмы планирования, строящие субоптимальные расписания.

В данной работе исследуются наиболее известные списочные эвристические алгоритмы планирования задач [1–3, 13, 14], ориентированные на разделение пространства вычислительных ресурсов. Среди прочих алгоритмов, оставшихся за рамками исследования, можно отметить алгоритм планирования групп задач (gang scheduling) [6], основанный на принципе разделения времени, когда на одних и тех же узлах поочередно выполняются различные наборы задач.

В рамках данного исследования в структуре алгоритма планирования выделяются следующие части: алгоритм выбора задачи для назначения на узлы, метод назначения задачи на узлы и критерий доступности узлов для планирования.

Алгоритм выбора задачи для назначения – алгоритм, который на основе имеющейся информации о доступных узлах кластера и ожидающих исполнения задачах, выбирает из очереди очередную задачу для назначения ей ресурсов.

Метод назначения определяет наиболее подходящие узлы из доступных для выбранной задачи. Варианты методов назначения хорошо освещены в источниках [10–14].

Критерий доступности узла представляет собой логическое выражение, завязанное на статических и динамических параметрах узла. Узел доступен тогда и только тогда, когда соответствующее логическое выражение принимает истинное значение. Критерий доступности узла особенно актуален для кластера рабочих станций. В рамках данного исследования при наличии локальной загрузки вычислительных узлов применялся критерий, представляющий собой требование, чтобы свертка загруженности оперативной памяти, дисковой памяти и процессора была не больше некоторого заданного порогового значения.

В данной работе рассматриваются следующие эвристические алгоритмы выбора задач для назначения: First Come First Served (FCFS, первым пришел – первым обслужен), First Come First Served Scan (FCFS Scan, первым пришел – первым вышел со сканированием очереди), Priority Queue (PQ, очередь с приоритетами), Priority Queue Scan (PQ Scan, очередь с приоритетами со сканированием), Shortest Job First (SJF, кратчайшая задача первая), Shortest Job First Scan (SJF Scan, кратчайшая задача первая со сканированием), Longest Job First (LJF, длиннейшая задача первая), Longest Job First Scan (LJF Scan, длиннейшая задача первая со сканированием), Most Processors First Served (MPFS, задача с наибольшим количеством процессоров первая), Most Processors First Served Scan (MPFS Scan, задача с наибольшим количеством процессоров первая со сканированием), Least Processors First Served (LPFS, задача с наименьшим количеством процессоров первая), Smallest Work Job First (SWJF, задача с наименьшей работой первая), Smallest Work Job First Scan (SWJF Scan, задача с наименьшей работой первая со сканированием), Largest Work Job First (LWJF, задача с наибольшей работой первая), Largest Work Job First Scan (LWJF Scan, задача с наибольшей работой первая со сканированием), Random First Served (RFS,

случайная задача обслуживается первой), агрессивный вариант алгоритма Backfill (обратного заполнения). Их описание можно найти в литературе [1–4, 10–14].

В рамках данного исследования рассматриваются следующие методы назначения задач на доступные вычислительные узлы кластера: First Fit (FF, первый подходящий) – выбираются первые попавшиеся подходящие узлы из первого подходящего окна расписания; Best Fit (BF, наилучший подходящий) – выбираются узлы из такого окна расписания, которое при закрытии его задачей даст наименьшую оставшуюся площадь; Fastest Node First (FNF, самый быстрый узел первым) – выбираются подходящие узлы подходящего окна, на которых данная задача может быть выполнена максимально быстро; Least Utilized Node First (LUNF, наименее загруженный узел первым) – выбираются наименее загруженные подходящие узлы подходящего окна; Random First (RF, случайный узел первым) – случайным образом (равновероятно) выбираются подходящие узлы случайно выбранного подходящего окна.

5. Система критериев и метрик сравнения алгоритмов планирования задач

С целью охвата всех аспектов эффективности составляемых алгоритмами планирования расписаний запуска задач была построена, описываемая в данном разделе, система критериев и метрик их сравнения.

Анализ алгоритмов планирования по определенному критерию представляет собой сравнение для различных алгоритмов зависимостей метрик этого критерия от величины системной загрузки и выбор лучшего варианта для того или иного случая. Сравнительный анализ удобнее всего представлять визуально с помощью графиков зависимостей метрик от системной загрузки L, определяемой как отношение суммарного объема вычислительной работы всех задач к объему работы, который кластер может потенциально выполнить (при полной загруженности) за время до появления в управляющей системе последней задачи.

Критерии сравнения алгоритмов планирования и метрики, к ним относящиеся:

1. Производительность расписаний. Включает следующие метрики: средняя загруженность процессора узла кластера $U_{проц.}$, потеря производительности кластера CL, максимальное время завершения задачи $C_{\max}$, среднее время ожидания задачи в очереди $\bar{t}_{ож.}$ и среднее ограниченное замедление задачи $\bar{s}_{огр.}$. Метрики $U_{проц.}$, CL и $C_{\max}$ характеризуют непосредственно производительность расписания, поэтому они более приоритетны для исследования, чем $\bar{t}_{ож.}$ и $\bar{s}_{огр.}$, которые имеют косвенный характер.

2. Используемость оперативной и дисковой памяти. Включает метрики $\overline{m}$ и $\overline{d}$, определяющие соответственно средние загруженности оперативной и дисковой памяти вычислительных узлов. Позволяет выявить процент неиспользованных ресурсов. Их, например, можно отдать под локальные приложения.

3. Сбалансированность загрузки узлов. Включает метрики Du, Dm и Dd, обозначающие соответственно дисперсии средние загруженности процессора, оперативной и дисковой памяти вычислительного узла. Демонстрирует честность алгоритма планирования по отношению к узлам.

4. Гарантированность обслуживания задач. Включает метрики максимального времени ожидания задачи в очереди $t_{ож.\max}$ и максимальное ограниченное замедление задачи $s_{огр.\max}$. Данный критерий имеет особую важность в случае, если кластер используется в рамках системы реального времени.

5. Честность по отношению к задачам. Содержит метрику $Dt_{ож.}$ – дисперсию времени ожидания задач в очереди. Позволяет оценить степень равноправности задач с точки зрения алгоритма планирования.

При исследовании различных алгоритмов планирования, нас, прежде всего, интересует производительность составляемых ими расписаний. Поэтому при анализе алгоритмов в первую очередь будет рассматриваться первый критерий сравнения. Остальные – имеют вторичный характер.

6. Описание технологии экспериментального исследования алгоритмов планирования задач с помощью симулятора вычислительного кластера и его управляющей системы

Опишем технологию экспериментального исследования алгоритмов планирования с помощью симулятора кластера и его управляющей системы, которая используется в данной работе.

Исследование состоит из нескольких сценариев, каждый из которых определяет вид используемого кластера. Например, они могут отличаться однородность аппаратной конфигурации, наличием локальной загрузки узлов. Каждый сценарий предполагает сбор значений метрик по каждой из $N_{тр.}$ трасс, которые затем усредняются для каждого алгоритма планирования. Трасса состоит из фиксированного числа задач n, параметры которых формируются согласно модели вычислительной загрузки.

Предполагается, что в первую очередь алгоритмы планирования (сочетание алгоритма выбора задачи, метода назначения и критерия доступности узлов) анализируются по критерию производительности.

Исследование с помощью симулятора кластера для одного сценария включает следующие шаги:

1. Выбрать и фиксировать критерий доступности узлов.
2. Выполнить симуляцию по всем трассам сценария для всех планировщиков. Получить усредненные значения метрик для каждого планировщика и каждого значения системной загрузки $L \in \{10\%, 20\%, ..., 100\%\}$.
3. Выбрать очередную метрику критерия производительности расписания ($U_{проц.}$, CL, $C_{\max}$, $\bar{t}_{ож.}$ или $\bar{s}_{огр.}$).
4. Для данной метрики и для каждого алгоритма выбора задач проанализировать его всевозможные варианты сочетания с методами назначения задач на узлы и в каждом случае выбрать лучший вариант. Удобно изображать в одной системе координат графики зависимости метрики от системной загрузки для всевозможных сочетаний алгоритма выбора с методами назначения.
5. Сопоставить графики лучших вариантов по данной метрике в одной системе координат. Сделать выводы о наиболее эффективных по данной метрике сочетаниях.
6. Если есть нерассмотренная метрика, то перейти на шаг 3.
7. Проанализировать наиболее эффективные по метрикам критерия производительности сочетания и на предмет использования одинаковых методов назначения и алгоритмов выбора задач и, учитывая, что метрики $U_{проц.}$, CL и $C_{\max}$ более приоритетны, выбрать лучшие алгоритмы планирования.
8. Для лучших алгоритмов планирования произвести сравнительный анализ по метрикам остальных вторичных критериев. При построении их графиков по некоторой вторичной метрике возможно рассмотрение также графиков сочетаний, которые являются лучшими среди комбинаций алгоритма выбора задач с методами назначения на узлы по этой метрике. Это позволит выявить сочетания, которые не входят в число лучших алгоритмов, но являются лучшими по отдельным метрикам.

Данная последовательность шагов исследования выполняется для каждого сценария, затем выявляются общие закономерности и делаются выводы. Заметим, что нас в первую очередь будут интересовать значения метрик, когда $L \geq 50\%$, т.к. при меньших значениях L интенсивность потока работ будет низкой и в этих условиях нельзя говорить о преимуществе того или иного алгоритма планирования.

7. Результаты экспериментального исследования алгоритмов планирования задач

Сценарии исследования алгоритмов планирования представлены в таблице 1. Они отличаются однородностью аппаратной конфигурации кластера и наличием локальной загрузки узлов.

В данной таблице также отражены составляющие части алгоритма планирования, которые имеет смысл варьировать для конкретного сценария. При наличии локальной загрузки вычислительных узлов применялся критерий в виде свертки Convolution of Utilizations (CU), а при отсутствии – Local Utilization Free (LUF).

Таблица 1. Сценарии исследования алгоритмов планирования

№ сцен.	Характеристики сценария		Алгоритмы планирования		
	Гомогенный кластер	Локальная загрузка узлов	Алгоритмы выбора работ для назначения	Методы назначения работ на узлы	Критерий доступности узлов
1	+	-	все	FF, RF	LUF
2	+	+	все	FF, LUNF, RF	CU
3	-	-	все	FF, FNF, LUNF, RF	LUF
4	-	+	все	все	CU

Для симуляции за основу была выбрана аппаратная конфигурация кластера Оренбургского государственного университета. Для формирования гетерогенного кластера она была видоизменена. В первом и втором сценарии кластер состоит из 14 узлов, каждый из которых обладает 1Гб оперативной памяти и 40Гб дисковой, все узлы имеют единичную относительную вычислительную скорость. В двух оставшихся сценариях применяется гетерогенный кластер, аппаратная конфигурация которого приведена в таблице 2.

Таблица 2. Аппаратная конфигурация гетерогенного кластера

Объем оперативной памяти, Мб	Объем дисковой памяти, Гб	Относительная вычислительная скорость	Количество узлов
1024	40	1	2
1024	40	1	2
512	40	1	2
512	20	3	2
2048	40	1	2
2048	40	4	2
256	20	4	2

Во втором и четвертом сценарии критерий доступности узлов имеет вид:

$$(M_i - m_i(t))/M_i \cdot 0.3 + (D_i - d_i(t))/M_i \cdot 0.1 + u_i(t) \cdot 0.6 \le 0.5. \qquad (4)$$

Коэффициенты выбраны исходя из соображений балансировки локальной загрузки узла и загрузки от управляющей системы.

В каждом сценарии исследуется 100 различных трасс по 500 задач в каждой.

Для простоты изложения договоримся название алгоритма планирования формировать из сокращений имен алгоритма выбора задачи и метода назначения. Например, алгоритм Most Processors First Served с методом Random First будет именоваться MPFS RF.

Подробно рассмотрим результаты исследования алгоритмов планирования задач на примере критерия производительности для первого сценария – гомогенного кластера без локальной загрузки вычислительных узлов. Согласно таблице 1 были составлены сочетания каждого алгоритма выбора задач с возможными методами назначения First Fit и Random Node First при фиксированном критерии доступности узлов Local Utilization Free. На основе результатов симуляции для каждой такой пары сочетаний в одной системе координат строились графики зависимостей конкретной метрики производительности от L, выбирался лучший из них вариант.

Графики зависимости метрик критерия производительности для всех лучших вариантов сочетаний приведены на рисунке 3.

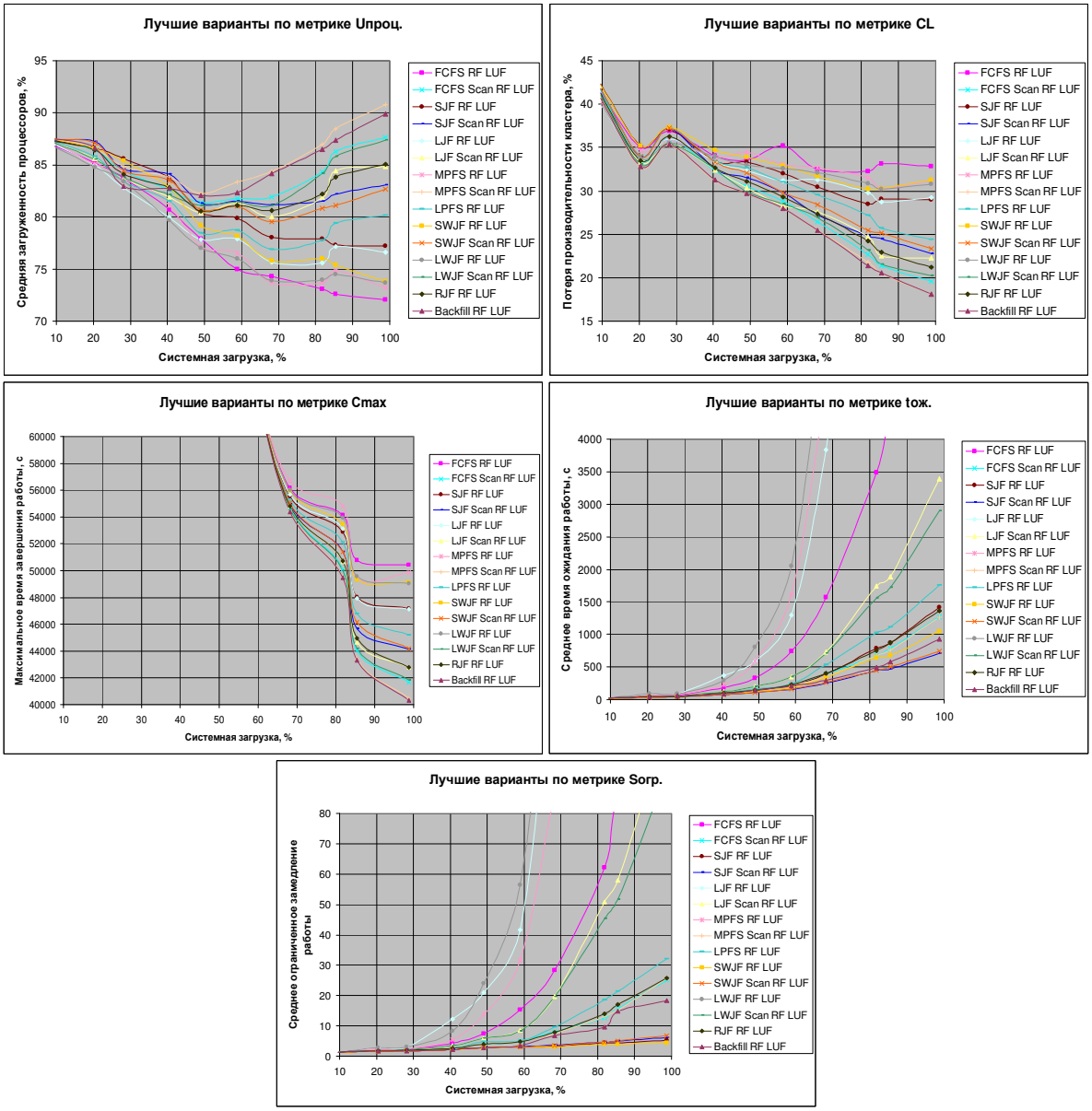


Рис. 3. Графики зависимости метрик критерия производительности от системной загрузки для различных алгоритмов планирования.

Согласно метрике средней загруженности процессора узла кластера $U_{проц.}$ лучшими сочетаниями являются комбинации алгоритмов выбора задач с методом назначения Random First. Наиболее эффективными алгоритмами планирования по данной метрике являются MPFS Scan RF и с незначительным отрывом – Backfill RF. Несколько хуже ведут себя алгоритмы FCFS Scan RF и LWJF Scan RF. Самые плохие результаты показывает алгоритм FCFS RF, что и следовало ожидать.

Метрика потери производительности кластера CL показывает, что Backfill RF и MPFS Scan RF являются наиболее эффективными алгоритмами, для них значения метрики CL практически неразличимы. Чуть хуже – FCFS Scan RF и LWJF Scan RF. Худшую потерю производительности показывает алгоритм FCFS RF.

Лучше всех минимизирует максимальное время завершения работы $C_{\max}$ алгоритмы Backfill RF и MPFS Scan RF, несколько хуже – FCFS Scan RF и LWJF Scan RF. Худшие результаты у FCFS RF и MPFS RF.

Наиболее эффективными по метрике среднего времени ожидания задачи в очереди $\bar{t}_{ож.}$ являются алгоритмы – SJF Scan RF и SWJF Scan RF, т.к. они обрабатывают большое число не-

больших задач быстрее, чем другие алгоритмы обрабатывают небольшое число больших и средних задач. Несколько хуже показывает результаты алгоритм Backfill RF. Результаты алгоритмов MPFS Scan и FCFS Scan практические не отличаются и находятся в середине. Хуже всех – LWJF RF.

Согласно метрике среднего ограниченного замедления задачи $\bar{s}_{огр.}$ очень близкие друг к другу наилучшие результаты демонстрируют алгоритмы SJF RF и SWJF RF, чуть хуже – SJF Scan RF и SWJF Scan RF. Далее идет алгоритм Backfill RF, несколько хуже показывают результаты MPFS Scan и FCFS Scan. Хуже всех – LWJF RF.

Значения всех метрик, кроме $U_{проц.}$, для сочетаний алгоритмов выбора задач с методами назначения First Fit и Random First совпадают. Это связано с тем, что кластер в данном сценарии имеет гомогенную структуру.

Результаты исследования по критерию производительности расписания показывают, что наиболее эффективными по метрикам $U_{проц.}$, CL и $C_{\max}$ являются алгоритмы Backfill RF и MPFS Scan RF, которые имеют очень близкие результаты. По метрикам $\bar{t}_{ож.}$ – SJF Scan RF и SWJF Scan RF, $\bar{s}_{огр.}$ – SJF RF и SWJF RF, в то же время данные алгоритмы демонстрируют плохие результаты по остальным метрикам. По двум последним метрикам Backfill RF и MPFS Scan RF также показывают неплохие результаты, поэтому они являются лучшими алгоритмами планирования по критерию производительности.

Аналогичным образом для первого сценария алгоритмы планирования задач были исследованы по оставшимся критериям.

Во всех четырех сценариях алгоритмы планирования, включающие алгоритмы выбора задач Backfill или MPFS Scan, являются лучшими по основному критерию производительности расписания, причем в первых двух (гомогенный кластер) они также являются лучшими по большинству вторичных критериев, а в оставшихся (гетерогенный кластер) – демонстрируют неплохие результаты.

Алгоритм с Backfill по основному критерию превосходит алгоритм с MPFS Scan во всех сценариях. При гомогенном кластере (первые два сценария) разница между ними небольшая, а при гетерогенном (последние два сценария) – является существенной. Также заметим, что в первых двух сценариях алгоритм MPFS Scan превосходит Backfill по большинству вторичных критериев, а в оставшихся сценариях наоборот. С другой стороны алгоритм Backfill по сравнению с MPFS Scan более сложен в реализации и требует указания пользователем оценок времени выполнения для задач.

В каждом сценарии лучшие результаты в сочетании с алгоритмом выбора демонстрирует свой метод назначения задач на узлы: гомогенный кластер без локальной загрузки узлов – Random First, гомогенный кластер с локальной загрузкой – Least Utilized Node First, гетерогенный кластер при наличии или отсутствии локальной загрузки – Fastest Node First.

По критерию используемости оперативной и дисковой памяти наиболее эффективными являются сочетания, использующие алгоритмы выбора Backfill или MPFS Scan.

Исследование алгоритмов планирования по критерию сбалансированности показывает следующие результаты: лучшие алгоритмы выбора по метрике Du – SJF Scan (первый сценарий при L не больше 50-70%), MPFS Scan (первый при L не меньше 50-70% и второй), Backfill (второй при L не больше 50-70%), FCFS Scan (второй при L не больше 50-70% и четвертый), FCFS (третий и четвертый); по метрикам Dm и Dd – SJF Scan (первый сценарий при L не больше 50-70%), MPFS Scan (первый при L не меньше 50-70%), Backfill (второй), FCFS Scan (второй), SJF (второй), FCFS (третий и четвертый).

Наилучшую гарантированность демонстрируют алгоритмы SWJF и SJF, которые могут быть использованы для построения систем реального времени.

Наилучшую честность по отношению к задачам обеспечивают следующие алгоритмы выбора (метрика $Dt_{ож.}$): MPFS Scan и Backfill в случае первого и второго сценария (гомогенный кластер); FCFS Scan – четвертый сценарий при L не больше 50-60%; FCFS –третий сценарий при L не больше 50-60%; SJF Scan – четвертый и третий сценарии при L не меньше 50-60%.

8. Заключение

В рамках данной работы была предложена имитационная схема кластера, разработана модель его вычислительной загрузки, а также построена система критериев и метрик сравнения различных алгоритмов планирования. Все они легли в основу симулятора вычислительного кластера и его управляющей системы – основного инструмента настоящего исследования.

Лучшим является алгоритм планирования, использующий Backfill, несколько хуже показывает себя MPFS Scan. Однако, Backfill более сложен в реализации и требует наличия достаточно точных оценок времени выполнения задач. Также для каждого сценария может быть рекомендован свой метод назначения задач и критерий доступности узлов для планирования.

Данное исследование будет продолжено в будущем за счет учета топологии вычислительного кластера, коммуникационных задержек при передаче данных, а также многопроцессорности вычислительных узлов.

Литература

1. Jones W.M., Pang L.W. Beowulf Mini-grid Scheduling. [Электронный ресурс] http://www.parl.clemson.edu/beosim.
2. Aida K., Kasahara H., Narita S. Job Scheduling Scheme for Pure Space Sharing among Rigid Jobs. //Lecture Notes In Computer Science, Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Vol. 1459. – London: Springer-Verlag, 1998. – p. 98–121.
3. Feitelson G. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. //12th International Parallel Processing Symposium / 9th Symposium on Parallel and Distributed Processing. – Orlando: Springer, 1998. – p. 542–546.
4. Feitelson G., Rudolph L., Metrics and Benchmarking for Parallel Job Scheduling. //Job Scheduling Strategies for Parallel Processing. – Orlando: Springer, 1998. – p. 1–24.
5. Feitelson G. Workload Modeling for Computer Systems Performance Evaluation. [Электронный ресурс] http://www.cs.huji.ac.il/~feit/wlmod/
6. Feitelson G. Packing Schemes for Gang Scheduling. //Lecture Notes In Computer Science, Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, Vol. 1162. – London: Springer-Verlag, 1996. – p. 89 - 110.
7. Downey A.B. A parallel workload model and its implications for processor allocation. //Cluster Computing, Volume 1 , Issue 1. – Hingham: Kluwer Academic Publishers, 1997. – p. 133 - 145.
8. Jann J. Modeling of Workload in MPPs. //Lecture Notes In Computer Science, Vol. 1291. – London: Springer-Verlag, 1997. – p. 95 - 116.
9. Lublin U. Feitelson G. The workload on parallel supercomputers: modeling the characteristics of rigid job. //Journal of Parallel and Distributed Computing archive, Volume 63 , Issue 11. – Orlando: Academic Press, 2003. – p. 542-546.
10. Blazewicz J., Ecker K., Pesch E., Schmidt G., Weglarz J. Handbook on Scheduling. From Theory to Applications. – Berlin: Springer, 2007. – 647 p.
11. Leung J. Y. Handbook of Scheduling. Algorithms, Models and Performance Analysis. – Boca Raton: CRC Press, 2004. – 622 p.
12. Коффман Э.Г. Теория расписаний и вычислительные машины. – М.: Наука, 1984. – 336 с.
13. Топорков В.В. Модели распределенных вычислений. – М.: ФИЗМАТЛИТ, 2004. – 320 с.
14. В.Н. Коваленко, Е.И. Коваленко, Д.А. Корягин, Д.А. Семячкин. Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами. [Электронный ресурс] http://www.keldysh.ru/papers/2007/source/prep2007_63.doc

Реализация процедур прогнозирования трудоемкости параллельного решения SAT-задач

О.С. Заикин

Разработана и реализована в виде MPI-программы крупноблочная параллельная технология решения SAT-задач (задач поиска решений уравнений вида «КНФ = 1», где КНФ – конъюнктивная нормальная форма) в распределенных вычислительных средах. В рамках данной технологии осуществляется декомпозиция исходной SAT-задачи на семейство подзадач. Используется процедура статистического прогнозирования трудоемкости параллельного решения SAT-задач, которая позволяет определить оптимальные (по прогнозу) параметры декомпозиции. Использование параметров декомпозиции, найденных с помощью процедур прогнозирования, позволяет успешно решать SAT-задачи, кодирующие задачи обращения ряда криптографических дискретных функций.

1. Введение

Многие значимые в практическом отношении комбинаторные проблемы допускают эффективные сводимости к задачам поиска решений булевых уравнений вида «КНФ=1» (КНФ – конъюнктивная нормальная форма) (см. [1]). Задачи поиска решений таких уравнений называются SAT-задачами, для их решения используются специальные программные комплексы, называемые SAT-решателями (см. [2]). В последнее время высокими темпами развиваются параллельные SAT-решатели (см., например, [3] и [4]). Как правило, в таких SAT-решателях используется концепция мелкозернистого параллелизма (см. [5]). Такой поход вполне оправдывает себя на многих классах тестов, например для КНФ, кодирующих задачи верификации в микроэлектронике (см. [6]). Применительно же к задачам обращения дискретных функций (например, к задачам криптоанализа поточных систем шифрования) высокую эффективность показала представленная в серии работ [7–11] крупноблочная параллельная технология решения SAT-задач в распределенных вычислительных средах (РВС). В рамках данной технологии осуществляется декомпозиция исходной SAT-задачи на семейство подзадач. Наилучшие результаты показали различные варианты декомпозиции по переменным, кодирующим вход дискретной функции, задачу обращения которой требуется решить. Для определения наилучших (по прогнозу) параметров декомпозиции используется процедура прогнозирования трудоемкости параллельного решения SAT-задач. Каждому варианту значений параметров декомпозиции соответствует случайная выборка SAT-задач. Прогноз заведомо быстро вычисляется для одной из выборок, затем он итеративно улучшается при обработке остальных выборок. Решение некоторых SAT-задач может быть прервано при превышении порогового значения.

Изначально данная технология была реализована в виде пакета прикладных программ (ППП) D-SAT [12], который функционирует под управлением инструментального комплекса DISCOMP (см. [10]). Функциональное наполнение ППП D-SAT включает процедуры решения SAT-задач и процедуры прогнозирования трудоемкости решения SAT-задач. Дальнейшим развитием данной параллельной технологии стала ее реализация в виде MPI-программы. Подробности данной реализации рассматриваются в настоящей статье.

В MPI-программе в режиме прогнозирования все SAT-задачи по всем случайным выборкам объединяются в единый параллельный список. В результате достигается равномерная загрузка РВС, но усложняется обработка данных. Для своевременного прерывания решений SAT-задач используются неблокирующие обмены (см. [13]), что позволяют каждому процессу эффективно использовать свое рабочее время: управляющий процесс занимается отправкой заданий и обработкой решений, вычислительные процессы решают SAT-задачи.

В данной работе впервые приведены результаты параллельного логического криптоанализа суммирующего генератора на основе четырех регистров сдвига с линейной обратной связью. Также приведены улучшенные результаты логического криптоанализа ряда других генераторов.

2. Крупноблочная параллельная технология решения SAT-задач

Далее приведено краткое описание крупноблочной параллельной технологии решения SAT-задач, представленной в работах [7–9].

Под распределенной вычислительной средой (РВС) понимается совокупность вычислительных единиц, объединенных коммуникационной сетью. В качестве вычислительной единицы РВС выступает программно-аппаратный ресурс, требуемый для решения некоторой вычислительной задачи. В качестве вычислительной единицы далее рассматривается одно ядро процессора, часть общей оперативной памяти и памяти жесткого диска, а также системное программное обеспечение.

Рассматривается произвольная конъюнктивная нормальная форма (КНФ) C над множеством булевых переменных $X = \{x_1,...,x_n\}$. В множестве X выбирается некоторое подмножество $\{x_{i_1},...,x_{i_d}\}, \{i_1,...,i_d\} \subseteq \{1,...,n\}, d \in \{1,...,n\}$. Множество $X' = \{x_{i_1},...,x_{i_d}\}$ называется *декомпозиционным множеством*, а d *– размерностью декомпозиционного множества*. Дополнительно полагается, что при $d = 0$ декомпозиционное множество пусто. Декомпозиционному множеству $X' : |X'| = d, d > 0$ ставится в соответствие множество $Y(X') = \{Y_1,...,Y_k\}$, состоящее из $k = 2^d$ различных двоичных векторов длины d, каждый из которых является набором значений переменных из множества X'. *Декомпозиционным семейством*, порожденным из КНФ C множеством X', называется множество $\Delta_{X'}(C)$ КНФ, полученных подстановками в C векторов $Y_j, j \in \{1,...,k\}$: $\Delta_{X'}(C) = \{C_1 = C|_{Y_1},...,C_k = C|_{Y_k}\}$, $\Delta_{\varnothing}(C) = \{C\}$. КНФ, полученная подстановкой в C вектора Y_j, обозначается через $C_j = C|_{Y_j}$.

Пусть $\Delta_{X'}(C) = \{C_1,...,C_k\}$ – декомпозиционное семейство КНФ, порожденное из КНФ C некоторым декомпозиционным множеством X' мощности d. Всякому набору, выполняющему исходную КНФ C, соответствует набор, выполняющий некоторую КНФ из семейства $\Delta_{X'}(C)$. Наоборот, произвольному набору, выполняющему некоторую КНФ из $\Delta_{X'}(C)$, соответствует единственный набор, выполняющий КНФ C. Следовательно, исходная КНФ C выполнима тогда и только тогда, когда выполнима хотя бы одна КНФ семейства $\Delta_{X'}(C)$. Таким образом, решение исходной SAT-задачи для КНФ C сводится к решению, вообще говоря, $k = 2^d$ SAT-задач для КНФ $C_1,...,C_k$ соответственно. Если исходная КНФ C выполнима, то по набору, выполняющему некоторую КНФ семейства $\Delta_{X'}(C)$, можно эффективно перейти к набору, выполняющему исходную КНФ C.

Пусть имеется РВС, состоящая из $r \in \mathrm{N}$ вычислительных единиц. Возможны следующие два случая.

1) $k \le r$ – число КНФ в семействе $\Delta_{X'}(C)$ не превосходит числа вычислительных единиц РВС. В этом случае для каждой КНФ из семейства $\Delta_{X'}(C)$ SAT-задача решается на отдельной вычислительной единице РВС.

2) $k > r$ – число КНФ в семействе $\Delta_{X'}(C)$ больше числа вычислительных единиц РВС.

Крупноблочное распараллеливание SAT-задач для случая $k \le r$ рассматривается, например, в работе [14]. Для случая $k > r$ предлагается следующая процедура.

Процедура 1. Каждому вектору $Y_j, j \in \{1,...,k\}$ ставится в соответствие натуральное число N_j, двоичным представлением которого является вектор Y_j. Данное число назовем натуральным индексом КНФ C_j. Семейство КНФ $\Delta_{X'}(C)$ упорядочивается некоторым образом (например, по возрастанию натуральных индексов соответствующих векторов). Произвольная КНФ из $\Delta_{X'}(C)$ называется *связанной*, если в рассматриваемый момент времени SAT-задача для нее либо уже решена, либо решается на некоторой вычислительной единице РВС. Остальные КНФ называются *свободными*. Выбираются первые r КНФ $C_1,...,C_r$ из семейства $\Delta_{X'}(C)$. Для каждой из выбранных КНФ $C_1,...,C_r$ решается SAT-задача на отдельной вычислительной единице

РВС. Как только освобождается некоторая из r вычислительных единиц РВС, на ней запускается процедура решения SAT-задачи для первой (в смысле введенного выше порядка) свободной КНФ семейства $\Delta_{X'}(C)$. Данный процесс продолжается до тех пор, пока не будет найден выполняющий набор некоторой КНФ из $\Delta_{X'}(C)$, либо пока не будет доказана невыполнимость всех КНФ из $\Delta_{X'}(C)$. Описанная процедура решает SAT-задачу для произвольной КНФ C корректно.

Пусть выбрано некоторое декомпозиционное множество X'. Представляет интерес построение такого $X^{\tilde{}} \subset X'$, использование которого в качестве декомпозиционного множества делает декомпозицию более эффективной, чем на основе X'. Данная проблема весьма нетривиальна. Если мощность $X^{\tilde{}}$ мала, то SAT-задачи, получаемые при декомпозиции КНФ, как правило, весьма сложны. Если мощность $X^{\tilde{}}$ велика, то велика и мощность декомпозиционного семейства $\Delta_{X^{\tilde{}}}(C)$, и в этом случае простота SAT-задач КНФ данного семейства мало что дает. Для решения данной задачи предлагается следующая процедура статистического прогнозирования (см. [7]).

Процедура 2. Используется натуральное число R_0, от значения которого зависит, имеется необходимость формирования случайной выборки или нет. Например, за R_0 можно принять число вычислительных единиц в РВС. Если при некотором $X^{\tilde{}} \subseteq X', |X^{\tilde{}}| = d$ мощность семейства $\Delta_{X^{\tilde{}}}(C)$ слишком велика, то представление о времени соответствующего параллельного вычисления можно составить на основе знания среднего времени решения SAT-задач для серии КНФ, выбранных случайным образом из $\Delta_{X^{\tilde{}}}(C)$. Через q_d обозначаем объем такой выборки. Через Y^d обозначается множество, образованное всеми различными векторами значений переменных из $X^{\tilde{}} : |X^{\tilde{}}| = d$. Каждому значению параметра $d \in \{0,1,...,|X'|\}$ такому, что $2^d > R_0$, ставится в соответствие множество векторов $\{Y_{j_1},...,Y_{j_{q_d}}\}$, выбираемых из $Y(X')$ в соответствии с равномерным распределением, а также выборка КНФ $\Theta_d = \{C_{j_1} = C|_{Y_{j_1}},...,C_{j_{q_d}} = C|_{Y_{j_{q_d}}}\}$. Каждому значению параметра $d \in \{0,1,...,|X'|\}$ такому, что $2^d \le R_0$, ставится в соответствие множество $Y(X')$ и множество КНФ $\Theta_d = \Delta_{X'}(C)$. Множество выборок $\{\Theta_d\}_{d\in\{0,1,...,|X'|\}}$ обозначается через Θ. Фиксируется SAT-решатель S. Обозначим через $t(C')$ время работы SAT-решателя S на произвольном входе C'. Вводится в рассмотрение функция

$$\tau_S : \Theta \to \mathrm{N}, \tau_S(\Theta_d) = \sum_{C'\in\Theta_d} t(C'),$$

значением которой при каждом фиксированном $d \in \{0,1,...,|X'|\}$ является суммарное время работы SAT-решателя S по всем КНФ из Θ_d. При некоторых значениях параметра d (например, при $d = 0$) КНФ из Θ_d могут оказаться очень сложными для SAT-решателя, и в этом случае время подсчета соответствующего значения прогнозной функции может превысить разумные границы. Для учета данного факта вводится в рассмотрение специальная функция $g(C) = p(m \cdot n)$, здесь m – число дизъюнктов в КНФ C, а $p(\cdot)$ – некоторый полином, степень которого больше 1.

Допустим, что в соответствии с перечисленными правилами построено семейство выборок $\Theta = \{\Theta_d\}_{d\in\{0,1,...,|X'|\}}$ (при фиксированном R_0). Прогнозная функция определяется следующим образом.

$$T(\Theta_d)=\begin{cases}\dfrac{2^d}{q_d}\cdot\tau_S(\Theta_d),\ 2^d>R_0,\ \tau_S(\Theta_d)<g(C);\\ \tau_S(\Theta_d),\ 2^d\le R_0,\ \tau_S(\Theta_d)<g(C);\\ \\ \infty,\ \tau_S(\Theta_d)\ge g(C).\end{cases}$$

Запись «$T(\Theta_d)=\infty$» означает, что функция не определена на выборке Θ_d. Рациональное число $T(\Theta_d)$ является прогнозом времени, требуемого для решения исходной SAT-задачи при декомпозиции КНФ C на семейство КНФ, порожденное множеством X^d. Тем самым задача прогнозного планирования оптимального по трудоемкости параллельного вычисления сводится к задаче минимизации функции T на множестве $domT\subseteq\Theta$. Идея оптимизации функции T состоит в том, что значение $T(\Theta_{|X'|})$ вычисляется заведомо эффективно. Затем значение T итеративно улучшается при обработке остальных выборок из $domT$. Если время обработки выборки превышает пороговое значение, обработка данной выборки прерывается. Результатом работы описанной процедуры является наилучшее (по прогнозу) значение $d_*\in domT$ мощности декомпозиционного множества X', а также соответствующее прогнозное время $T(\Theta_{d_*})$.

Эффективность процедуры 1 существенным образом зависит от структуры декомпозиционного множества. Выбор декомпозиционного множества – это отдельная нетривиальная проблема. Некоторые общие стратегии построения декомпозиционных множеств с ориентацией на задачи криптоанализа генераторов ключевого потока были рассмотрены в [10].

3. Описание MPI-программы PD-SAT

Приведенная в разделе 2 технология была реализована в виде MPI-программы PD-SAT, которую можно также назвать параллельным SAT-решателем. PD-SAT может функционировать в режиме решения SAT-задачи (см. раздел 3.1) и в режиме прогнозирования трудоемкости решения SAT-задачи (см. раздел 3.2). Следует особо отметить принципиальные различия данных, обрабатываемых в указанных режимах. Если в режиме решения заданиями являются списки SAT-задач, то в режиме прогнозирования заданиями являются конкретные SAT-задачи.

В режиме решения SAT-задачи декомпозиционное семейство разбивается на непересекающиеся подсемейства КНФ. Каждое такое подсемейство образует задание, которое обрабатывается на фиксированной вычислительной единице РВС. Под решением задания понимается решение SAT-задач для всех КНФ из соответствующего подсемейства: ответ на задание «UNSAT», если все КНФ из подсемейства невыполнимы; ответ «SAT», если хотя бы одна КНФ из подсемейства выполнима.

Режим прогнозирования реализует описанную в разделе 2 процедуру статистического прогнозирования трудоемкости решения SAT-задачи. В данном режиме заданиями являются SAT-задачи для КНФ, образующих обрабатываемую случайную выборку. Тем самым каждой такой выборке сопоставляется *выборка заданий*.

Все сказанное позволяет выделить следующие классы заданий:

- *Свободные задания* – задания, процесс решения которых на текущий момент не был запущен;
- *Связанные незавершенные задания* – задания, которые решаются на текущий момент;
- *Связанные завершенные задания* – задания, которые на текущий момент уже решены.

3.1 Реализация режима решения SAT-задачи

Данный режим основан на процедуре 1, приведенной в разделе 2. Вычисления разделены на три этапа.

Этап 1. PD-SAT запущен на n процессах: процесс номер 1 управляющий, процессы с номерами 2,…,n – вычислительные. Управляющий процесс по входным данным формирует список заданий, вычислительные процессы при этом простаивают. Число заданий D равно ближайшей справа степени двойки от числа $(n-1)\cdot C$. Здесь C – константа, влияющая на загрузку вычислительных процессов. Данная константа определяется эмпирически; в вычислительных экспериментах, описанных в разделе 4, использовалась $C = 4$.

Пусть, например, дана РВС, состоящая из четырех вычислительных единиц. В MPI-программе один управляющий процесс и три вычислительных. Пусть $C = 2$. В соответствии со сказанным выше управляющий процесс сгенерирует $D = 8$ заданий.

Этап 2. С управляющего процесса отсылаются первые n-1 свободных заданий из списка: i-ое задание (i=1,…,n-1) отсылается на вычислительный процесс с номером i+1 (каждое такое задание становится связанным незавершенным). Каждый вычислительный процесс приступает к обработке полученного задания.

Пример: 1 управляющий процесс, 3 вычислительных процесса, 8 заданий. Схема выполнения второго этапа для данного примера приведена на Рис. 1, управляющий процесс обозначен «УП», вычислительные – «ВП».

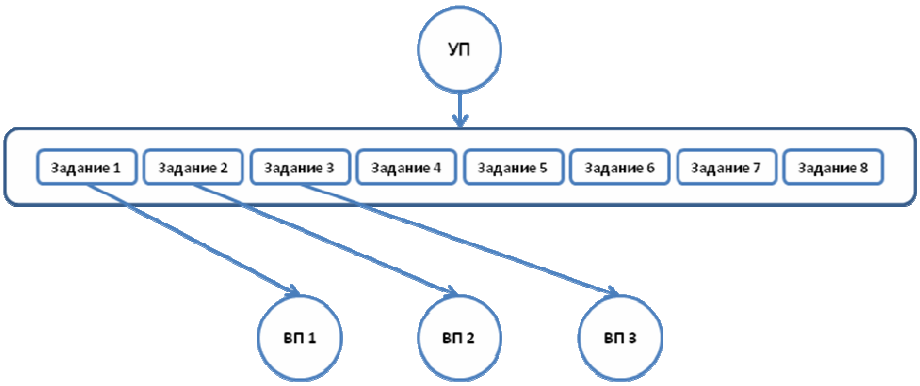


Рис. 1. Пример выполнения этапа 2 режима решения SAT-задачи

Первые два этапа являются подготовительными, выполняются очень быстро, и время их выполнения не вносит значительного вклада в общее время работы.

Этап 3. После выполнения этапов 1–2 управляющий процесс переходит в состояние ожидания решений заданий с вычислительных процессов. Если на управляющий процесс приходит ответ «UNSAT», то задание, ответ на которое был прислан, становится связанным завершенным. На приславший данный ответ вычислительный процесс отправляется очередное свободное задание из списка. Программа завершают свою работу, если на управляющий процесс приходит ответ «SAT» (в этом случае исходная КНФ выполнима) или если получены ответы «UNSAT» на все задания (в этом случае исходная КНФ невыполнима).

В этапах 1–3 используются только блокирующие функции обмена MPI_Send и MPI_Recv (см. [15]). Безусловно, приведенная схема решения проста и присуща многим задачам, допускающим крупноблочное распараллеливание, но ее описание полезно для понимания работы PD-SAT в режиме прогнозирования (см раздел 3.2).

3.2 Реализация режима прогнозирования трудоемкости параллельного решения SAT-задачи

Данный режим основан на процедуре 2, приведенной в разделе 2. Как и в режиме решения SAT-задачи, вычисления в режиме прогнозирования разделены на три этапа. Основные отличия режимов – в функционировании на третьем этапе.

Еще раз отметим, что в режиме прогнозирования заданием является конкретная SAT-задача (в отличие от режима решения, где заданием является список SAT-задач).

Далее используются обозначения, введенные в работе [11]. Через $d^{\min}$ и $d^{\max}$ обозначаются натуральные числа, определяющие соответственно нижнюю и верхнюю границы интервала,

в котором изменяются значения d , где d – размерность декомпозиционного множества (см. раздел 2). Через q обозначается жестко заданное значение, определяющее число КНФ в произвольной случайной выборке, а через r – число вычислительных единиц РВС, относительно которого строится прогноз. В режиме прогнозирования на вход PD-SAT подаются параметры $d^{\min}$, $d^{\max}$, q, r.

Этап 1. Как и в режиме решения SAT-задачи, PD-SAT использует *n* процессов. На управляющем процессе формируется список заданий. Для каждого значения $d \in \{d^{\min},...,d^{\max}\}$ строится отдельная выборка заданий. В случае $2^d > q$ в выборку заданий включаются SAT-задачи для q случайным образом выбранных КНФ из декомпозиционного семейства. Если $2^d \le q$, то в выборку включаются SAT-задачи для всех КНФ из декомпозиционного семейства. Задания из всех выборок объединяются в единый параллельный список, притом первыми в списке располагаются задания из выборки, полученной при $d = d^{\max}$. Далее задания располагаются по убыванию значения d, последними в списке расположены задания из выборки, полученной при $d = d^{\min}$.

Этап 2. Подобно режиму решения SAT-задачи, на данном этапе с управляющего процесса отсылаются первые *n*-1 свободных заданий из списка: *i*-ое задание (*i*=1,…,*n*-1) отсылается на вычислительный процесс с номером *i*+1.

Этап 3. На данном этапе осуществляется параллельная обработка различных выборок заданий с целью построения прогнозов трудоемкости решения исходной SAT-задачи при использовании соответствующей декомпозиции. Основной на данном этапе является процедура прогнозирования GetPredict, работающая на управляющем процессе. Данная процедура, во-первых, определяет параметры лучшего на текущий момент прогноза трудоемкости решения исходной SAT-задачи. Во-вторых, она определяет, обработка каких выборок заданий должна быть прервана, ввиду превышения соответствующими процессами текущих ограничений на время работы. Процедура GetPredict запускается через малые временные интервалы (на практике использовался двухсекундный интервал). На входе GetPredict получает следующие массивы:

- cnf_real_time_arr (в данном массиве содержится получаемая от вычислительных процессов информация о времени обработки связанных завершенных заданий);
- cnf_appr_time_arr (данный массив строится на управляющем процессе и содержит информацию о времени обработки связанных незавершенных заданий);
- cnf_status_arr – массив статусов заданий;
- set_status_arr – массив статусов выборок заданий.

Статус задания и статус выборки заданий – это динамически изменяющиеся параметры, которые в различные моменты вызова GetPredict могут принимать различные значения.

В текущий момент статус задания может принимать следующие значения:

- WAIT, если задание свободное или связанное незавершенное;
- STOP, если задание находится в выборке, обработка которой прерывается. Задания, получившие статус STOP (в том числе и свободные на текущий момент), в дальнейшем не обрабатываются;
- UNSAT, если задание связанное завершенное и соответствующая ему КНФ оказалась невыполнимой;
- SAT, если задание связанное завершенное и соответствующая ему КНФ оказалась выполнимой.

В текущий момент статус выборки заданий может принимать следующие значения:

- WAIT, если в выборке имеются задания со статусом WAIT, но нет ни одного задания со статусом SAT;
- SAT, если хотя бы одно задание из выборки получило статус SAT;
- STOP, если счет для выборки прерван;
- UNSAT, если все задания из выборки имеют статус UNSAT.

На выходе GetPredict выдает измененные массивы cnf_appr_time_arr, cnf_status_arr, set_status_arr, а также массив cnf_to_stop_arr, содержащий номера вычислительных процессов, на которых должна быть прервана обработка текущих заданий (данный массив может быть пустым).

Прерывание обработки заданий достигается за счет отправки с управляющего процесса неблокирующих сообщений (используются функции MPI_Isend) о прерывании на вычислительные процессы с номерами из массива cnf_to_stop_arr (если данный массив не пуст).

От вычислительных процессов требуется не только получать и решать задания, но и периодически проверять наличие сообщений о прерываниях. В используемые SAT-решатели были внесены изменения, позволяющие осуществлять такую проверку за счет применения неблокирующих функций MPI_Iprobe. Если сообщение о прерывании есть, то работа SAT-решателя досрочно завершается, выдается ответ «UNSAT». Даже если КНФ была на самом деле выполнимой, для прогнозирования это не важно. Сообщение с ответом «UNSAT» отправляется на управляющий процесс, после чего принимается следующее задание.

Между периодическими запусками процедуры прогнозирования управляющий процесс переходит в состояние ожидания ответов от вычислительных. Если от вычислительного процесса присылается ответ «UNSAT», на приславший этот ответ вычислительный процесс отправляется очередное свободное задание из списка, время решения SAT-задачи заносится в массив cnf_real_time_arr. Процедура прогнозирования завершает свою работу, если управляющий процесс получил ответ «SAT» (в этом случае SAT-задача для исходной КНФ решена в режиме прогнозирования, исходная КНФ выполнима) или если для всех заданий получены ответы «UNSAT».

Применение неблокирующих обменов позволяют каждому процессу эффективно использовать свое рабочее время: управляющий процесс занимается отправкой заданий и обработкой решений, вычислительные процессы решают SAT-задачи.

Дополнительно отметим, что в PD-SAT предусмотрена процедура отслеживания «опоздавших» сообщений о прерывании: такие сообщения могут возникать вследствие того, что за время обработки данных управляющим процессом на некотором вычислительном процессе было решено задание из выборки, обработку которой необходимо было прервать (но этого не было сделано из-за загруженности управляющего процесса). В этом случае сообщение о прерывании от управляющего процесса может быть некорректно интерпретировано. Такого рода ситуации исключаются за счет дополнительной проверки статусов сообщений, поступающих на вычислительные процессы от управляющего.

В PD-SAT используются следующие SAT-решатели, основой которых является известный решатель Minisat (см. [16]):

- dminisat, основанный на MiniSat-C_v1.14.1. (версия на языке C), оптимизирован для решения SAT-задач, кодирующих задачи обращения дискретных функций (см. [10]);
- minisat2, без существенных изменений;
- minisat2_mod, основан на minisat2, внесены изменения в ключевые параметры-константы, добавлено увеличение активности ядровых переменных.

Изначально SAT-решатели семейства Minisat предназначены только для работы под Unix-подобными операционными системами (ОС). В исходный код всех используемых в PD-SAT SAT-решателей были внесены изменения, обеспечивающие платформонезависимость (в смысле переносимости на уровне исходного кода, см. [17]). Тем самым, PD-SAT может функционировать как под управлением Unix-подобных ОС, так и под управлением ОС семейства Windows.

4. Вычислительные эксперименты

В данном разделе приведены результаты криптоанализа ряда генераторов ключевого потока, полученных, в том числе, с использованием программы PD-SAT.

Последовательный логический криптоанализ, реализованный на обычном персональном компьютере (ПК), оправдал себя применительно к генераторам Геффе и Вольфрама (см. [18], [19]). Применение комплекса TransAlg (см. [20]) позволило получить более экономные КНФ-

представления ряда криптографических алгоритмов в сравнении с полученными ранее посредством LC-комплекса (см. [21]). Данный факт, а также адаптация SAT-решателей к задачам обращения дискретных функций (см. [10]), позволили осуществить последовательный логический криптоанализ суммирующего генератора (см. [22], [23], [24]) на основе трех регистров сдвига с линейной обратной связью (РСЛОС), задаваемых следующими полиномами обратной связи: $X^{19}+X^{18}+X^{17}+X^{14}+1;\ X^{22}+X^{21}+1;\ X^{23}+X^{22}+X^{21}+X^{8}+1$. Длина инициализирующей последовательности составляет 66 бит (64 бита – начальное заполнение РСЛОС 1–3 и 2 бита – начальное заполнение регистров сумматора), анализировался фрагмент ключевого потока длиной 180 бит. В таблице 1 приведены результаты последовательного логического криптоанализа суммирующего генератора данной конфигурации для всех трех используемых в PD-SAT SAT-решателей (см. раздел 3).

Таблица 1. Результаты последовательного логического криптоанализа 66-битного суммирующего генератора на основе трех РСЛОС.

Время решения / SAT-решатель	Минимальное	Максимальное	Среднее
minisat2	1 мин.	4 ч. 40 мин.	1 ч. 17 мин.
minisat2_mod	4 мин.	1 ч. 30 мин.	58 мин.
dminisat	21 мин.	8 ч. 37 мин.	2 ч. 51 мин.

Несмотря на все сказанное, в отношении описанных в таблице 2 генераторов последовательный логический криптоанализ по-прежнему неэффективен.

Таблица 2. Описание генераторов.

Описание генератора	РСЛОС	Длина фрагмента ключевого потока	Размер КНФ	
			Переменных	Дизъюнктов
Пороговый 5 РСЛОС, 72 бита	$X^{11}+X^{9}+X^{4}+X^{2}+1;$ $X^{13}+X^{4}+X^{3}+X+1;$ $X^{15}+X^{5}+X^{4}+X^{2}+1;$ $X^{16}+X^{6}+X^{4}+X+1;$ $X^{17}+X^{6}+X^{4}+X^{2}+1$	150	972	15000
Пороговый 5 РСЛОС, 80 бит	$X^{13}+X^{10}+X^{8}+X^{5}+1;$ $X^{15}+X^{13}+X^{3}+X+1;$ $X^{16}+X^{13}+X^{8}+X^{2}+1;$ $X^{17}+X^{6}+X^{4}+X^{2}+1;$ $X^{19}+X^{18}+X^{17}+X^{14}+1$	150	980	15000
Суммирущий, 4 РСЛОС, 63 бита	$X^{13}+X^{4}+X^{3}+X+1;$ $X^{15}+X^{5}+X^{4}+X^{2}+1;$ $X^{16}+X^{6}+X^{4}+X+1;$ $X^{17}+X^{6}+X^{4}+X^{2}+1$	180	1683	19266

Для параллельного логического криптоанализа перечисленных генераторов была использована описанная в разделе 3 MPI-программа PD-SAT. Вычислительные эксперименты осуществлялись на кластере Blackford Multicore ИДСТУ СО РАН (см. [25]), который имеет следующие основные характеристики: 20 вычислительных узлов; 40 четырехъядерных процессоров Intel Xeon Quad-Core E5345 2.33 GHz; пиковая производительность – 1,493 TFlops; наивысшая производительность по Linpack – 924,4 GFlops; интерконнект 2 x Gigabit Ethernet; ОС Gentoo Linux.

При проведении вычислительных экспериментов кластер Blackford Multicore рассматривался как РВС (далее «РВС Blackford»), вычислительная единица которой состоит из одного ядра процессора Intel Xeon Quad-Core E5345 2.33 GHz, общей оперативной памяти и жесткого диска. В целях единообразного представления далее приводятся сравнительные результаты последовательного решения SAT-задачи на одной вычислительной единице РВС Blackford и результаты параллельного решения SAT-задачи при помощи программы PD-SAT в данной РВС.

При параллельном решении SAT-задач, кодирующих криптоанализ 63-битного суммирующего генератора на основе четырех РСЛОС, использовались 129 вычислительных единиц РВС Blackford. Для остальных генераторов использовались 72 вычислительные единицы.

В таблице 3 приведены результаты прогнозирования трудоемкости параллельного логического криптоанализа рассматриваемых генераторов и соответствующие прогнозу параметры декомпозиции.

Таблица 3. Результаты прогнозирования.

SAT-решатель / Генератор/ Число используемых единиц РВС	minisat2	minisat2_mod	dminisat
Пороговый, 5 РСЛОС, 72 бита / 72 единицы РВС	7 мин. 38 сек. 13 переменных	8 мин. 17 сек. 16 переменных	6 мин. 55 сек. 15 переменных
Пороговый, 5 РСЛОС, 80 бит / 72 единицы РВС	4 ч. 20 мин. 14 переменных	4 ч. 9 мин. 24 переменных	4 ч. 4 мин. 26 переменных
Суммирующий, 4 РСЛОС, 63 бита / 129 единиц РВС	42 ч. 4 мин. 28 переменных	14 ч. 18 мин. 25 переменных	13 ч. 9 мин. 24 переменных

Для каждого генератора представлены примеры оптимизации прогнозной функции (см. Рис. 2–4), полученные при использовании в PD-SAT SAT-решателя dminisat (он оказался лучшим по прогнозу среди SAT-решателей). Заштрихованные сеткой столбцы означают, что соответствующие вычисления прогнозной функции были прерваны из-за превышения текущего порогового значение (см. раздел 2). С использованием параметров декомпозиции, найденных при помощи процедур прогнозирования (см. таблицу 3), осуществлен криптоанализ перечисленных генераторов, результаты приведены в таблице 4.

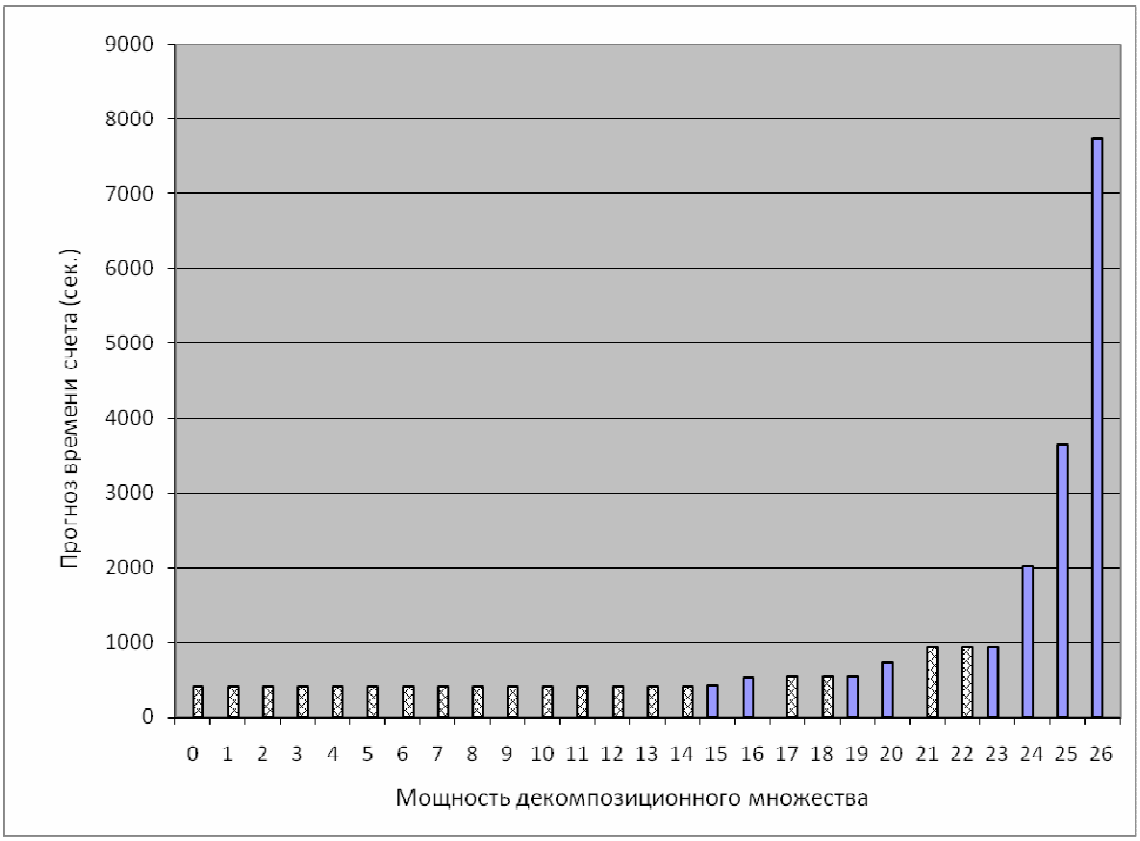


Рис. 2. Пример оптимизации прогнозной функции для 72-битного порогового генератора (один тест)

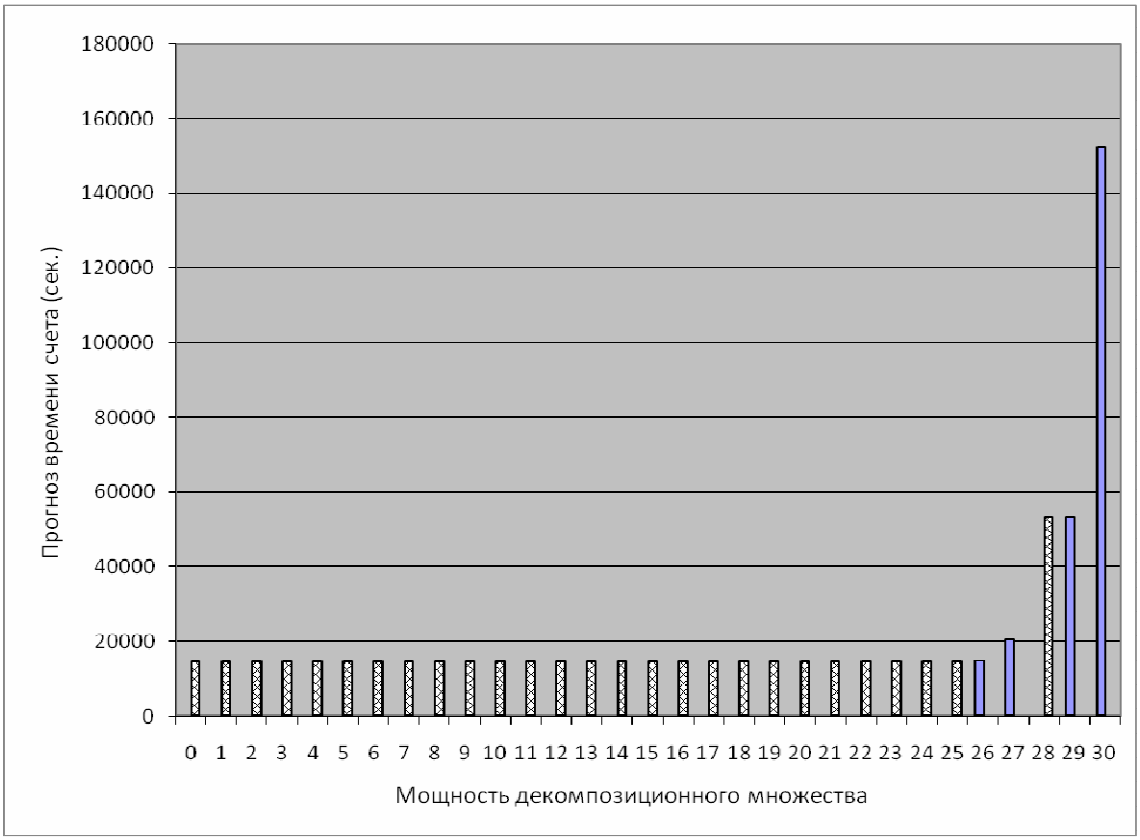


Рис. 3. Пример оптимизации прогнозной функции для 80-битного порогового генератора (один тест)

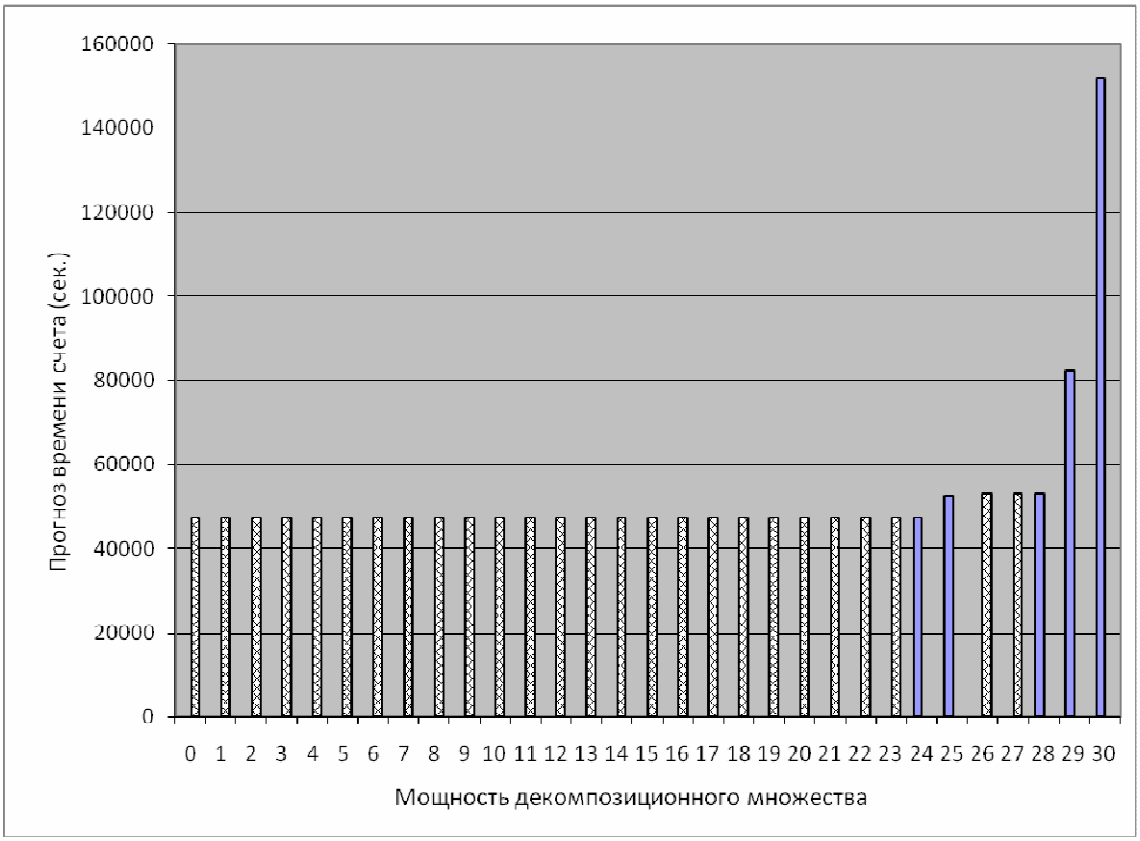


Рис. 4. Пример оптимизации прогнозной функции для 63-битного суммирующего генератора на основе четырех РСЛОС (один тест)

Таблица 4. Результаты криптоанализа некоторых генераторов (серия тестов).

Генератор / Число используемых единиц РВС	Время параллельного решения			Время последовательного решения
	Минимальное	Максимальное	Среднее	
Пороговый, 5 РСЛОС, 72 бита / 72 единицы РВС	3 мин. 57 сек.	13 мин. 4 сек.	6 мин. 29 сек.	> 1 суток (вычисление прервано)
Пороговый, 5 РСЛОС, 80 бит / 72 единицы РВС	34 мин.	3 ч. 17 мин.	1 ч. 34 мин.	> 1 суток (вычисление прервано)
Суммирующий, 4 РСЛОС, 63 бита / 129 единиц РВС	12 мин.	5 ч.	2 ч. 6 мин.	> 2 суток (вычисление прервано)

Заключение

В работе представлена реализация крупноблочной параллельной технологии решения SAT-задач в виде MPI-программы PD-SAT. Данная программа позволяет осуществлять прогнозирование трудоемкости решения SAT-задач и их непосредственное решение в рамках любой распределенной вычислительной среды с установленной коммуникационной MPI-средой.

На серии численных экспериментов продемонстрировано успешное использование PD-SAT в решении задач логического криптоанализа ряда поточных систем шифрования, последовательный логический криптоанализ в отношении которых не дал приемлемых результатов.

Предполагается дальнейшее развитие представленной в работе технологии и ее применение в параллельном логическом криптоанализе других систем шифрования.

Автор благодарит Семенова Александра Анатольевича за внимание к работе и участие в обсуждении основных результатов.

Литература

1. Семенов А.А. О сложности обращения дискретных функций из одного класса // Дискретный анализ и исследование операций. 2004. Т. 11. № 4. С. 44-55.
2. Семенов А.А., Беспалов Д.В. Технологии решения многомерных задач логического поиска // Вестник Томского гос. ун-та. – Приложение. – 2005. – № 14. – С. 61-73.
3. Luís Gil, Paulo Flores, Luís Miguel Silveira. PMSat: a parallel version of MiniSAT. Journal on Satisfiability, Boolean Modeling and Computation, 2008. –Volume 6. –71-98.
4. Tobias Schubert, Matthew Lewis, Bernd Becker. PaMiraXT: Parallel SAT Solving with Threads and Message Passing // Journal on Satisfiability, Boolean Modeling and Computation, 2009. – Volume 6. –P. 203-222.
5. Бандман О.Л. Мелкозернистый параллелизм в вычислительной математике // Программирование. –2001. –№ 4. –С. 5-20.
6. Miroslav N. Velev, Randal E. Bryant. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors // Journal of Symbolic Computation, 2003. –Volume 35, Issue 2. –P. 73-106.
7. Заикин О.С., Семенов А.А. Технология крупноблочного параллелизма в SAT-задачах // Проблемы управления. 2008. №1. С. 43-50.
8. Семенов А.А., Заикин О.С. Неполные алгоритмы в крупноблочном параллелизме комбинаторных задач // Вычислительные методы и программирование. – 2008. т. 9. – С. 108-118.
9. Заикин О.С., Семенов А.А., Сидоров И.А., Феоктистов А.Г. Параллельная технология решения SAT-задач с применением пакета прикладных программ D-SAT. // Вестник ТГУ. – Приложение. 2007. – № 23. – С. 83-95.

10. Семенов А.А., Заикин О.С., Беспалов Д.В., Буров П.С., Хмельнов А.Е. Решение задач обращения дискретных функций на многопроцессорных вычислительных системах // Труды Четвертой Международной конференции «Параллельные вычисления и задачи управления» PACO'2008 (Москва 26-29 октября 2008). – С. 152-176.

11. Заикин О.С. Декомпозиционные представления данных в крупноблочном параллелизме SAT-задач // Прикладные алгоритмы в дискретном анализе. Иркутск: ИГУ, 2008. – Серия: Дискретный анализ и информатика, вып. 2. – С. 49-69.

12. Заикин О.С. Пакет прикладных программ Distributed-SAT: Свидетельство об официальной регистрации программы для ЭВМ № 2008610423. – М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам, 2008.

13. Гришагин В.А., Свистунов А.Н. Параллельное программирование на основе MPI. Учебное пособие. – Нижний Новгород: издательство ННГУ им. Н.И. Лобачевского, 2005.

14. Опарин Г.А., Богданова В.Г., Сидоров И.А. Интеллектуальный решатель задач в булевых ограничениях в распределенной вычислительной среде // Информационные и математические технологии в науке и управлении. – Иркутск: ИСЭМ РАН, 2007. – С. 32-40.

15. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002. – 608 с.

16. SAT-решатель MiniSat: [http://minisat.se/MiniSat.html], 10.12.2009.

17. Олейников А.Я. Методика тестирования на соответствие стандартам, обеспечивающим переносимость прикладных программ (POSIX). – Москва, 1999. – 31 с.

18. Семенов А.А. Логико-эвристический подход в криптоанализе генераторов двоичных последовательностей // Труды международной научной конференции ПАВТ'07. Челябинск, ЮУрГУ, 2007. – Т. 1, С. 170-180.

19. Семенов А.А., Заикин О.С., Беспалов Д.В., Ушаков А.А. SAT-подход в криптоанализе некоторых систем поточного шифрования // Вычислительные технологии. 2008. – Т. 13, № 6. – С. 134-150.

20. Буров П.С., Игнатьев А.С., Отпущенников И.В. Программная трансляция алгоритмов в логические выражения в задачах диагностирования дискретных систем // Материалы IX школы-семинара «Математическое моделирование и информационные технологии», Иркутск, 2007. – С. 33-35.

21. Буранов Е.В. Программная трансляция процедур логического криптоанализа симметричных шифров // Вестник Томского гос. ун-та. – Приложение. – 2004. – № 9 (1). – С. 60-65.

22. Rueppel R.A., Correlation immunity and the summation combiner. In Lecture Notes in Computer Science 218; Advances in Cryptology: Proc. Crypto'85, H. C. Williams Ed., Santa Barbara, CA, Aug. 18-22, 1985, P. 260-272. Berlin: Springer-Verlag, 1986.

23. Menezes A., Van Oorschot P., Vanstone S. Handbook of Applied Cryptography. – CRC Press, 1996. – 657 с.

24. Поточные шифры. Результаты зарубежной открытой криптологии. – М.:Мир, 1997. – 389 с.

25. Суперкомпьютерный центр ИДСТУ СО РАН [http://www.mvs.icc.ru], 10.12.2009.

Параллельные алгоритмы построения изоповерхностей на больших сетках*

В.А. Киев, А.К. Кузин, С.Г. Орлов,
Б.Н. Четверушкин, Н.Н. Шабров, М.В. Якобовский

Рассматривается задача о построении редуцированной изоповерхности на большой (порядка 10^9 узлов) нерегулярной сетке тетраэдров. В каждом узле сетки задано значение непрерывного скалярного поля, изоповерхность которого требуется найти. Внутри каждого тетраэдра поле интерполируется линейно. Предложены реализации параллельных алгоритмов построения изоповерхностей, ориентированные на многоядерные вычислительные архитектуры; рассматривается также возможность использования GPU.

1. Введение

Исходная сетка разбита на домены, в каждом из которых — порядка 10^6 узлов (таким образом, имеется около 1000 доменов); имеется также соответствие между узлами границ, разделяющих соседние домены. Каждый домен может быть обработан отдельно, независимо от остальных. Результат обработки домена — кусок редуцированной изоповерхности на нём. Полученные на доменах куски изоповерхности попарно сшиваются и затем заново редуцируются.

Схема рассматриваемой программной реализации алгоритмов построения редуцированной изоповерхности представлена на рис. 1.

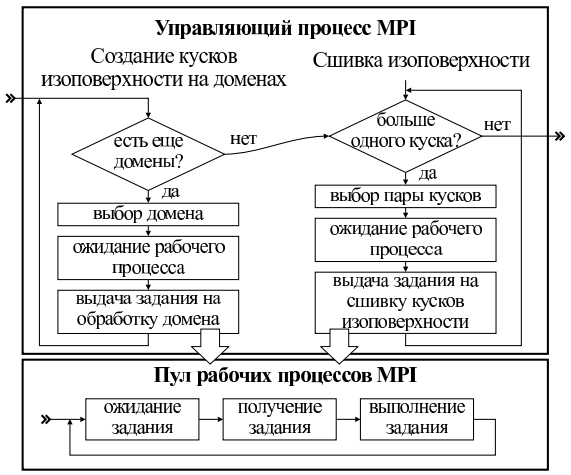


Рис. 1. Схема работы программы

Вначале создается пул процессов MPI, в одном из которых выполняется код планировщика заданий, а остальные ждут от него заданий. Планировщик выдает освободившимся рабочим процессам задания, сводящиеся к обработке отдельных доменов. Таким образом, обеспечивается динамическая балансировка загрузки CPU. Пока количество доменов превышает количество CPU, имеет место рост производительности с ростом количества CPU.

После того, как все домены обработаны, производится сшивка кусков и последующее редуцирование получающихся более крупных кусков. Процесс продолжается до тех пор, пока не останется единственный кусок. На этом этапе рабочим процессам выдаются задания, состоящие в сшивке и редуцировании пары кусков изоповерхности. Выбором пар

*Авторы работы благодарят РФФИ за поддержку исследований в рамках гранта № 09-07-12020-офи_м. Также авторы благодарны компании «Ниагара компьютерс» за предоставленное компьютерное оборудование.

занимается процесс планировщикаика заданий.

При размере домена порядка 10^6 узлов его обработка возможна как на CPU, так и на GPU, причем следует ожидать, что благодаря параллелизации внутри домена использование GPU позволит значительно сократить время обработки. Для эффективного использования GPU разработаны специализированные параллельные версии алгоритмов создания и редуцирования изоповерхности. Рабочие потоки MPI определяют, следует ли использовать CPU или GPU — в зависимости от того, имеется ли в системе свободный GPU. Для этого разработан распределитель ресурсов, отслеживающий занятость вычислительных ресурсов.

Таким образом, в программной реализации выделяются несколько отдельных частей: алгоритм генерации изоповерхности на одном домене (реализации для CPU и GPU); алгоритм редуцирования изоповерхности (реализации для CPU и GPU); алгоритм сшивки двух кусков изоповерхности (реализация для CPU); планировщик заданий; распределитель ресурсов.

2. Создание изоповерхности

Для алгоритма генерации изоповерхности существенно, что сетка состоит из тетраэдров, и скалярное поле линейно на каждом из них. Поэтому каждое ребро сетки пересекается с изоповерхностью не более, чем в одной точке. Пересечение изоповерхности с тетраэдром — либо треугольник, либо четырехугольник, так как часть изоповерхности внутри каждого тетраэдра — плоская.

Особо следует отметить случай, когда значение поля в некотором узле в точности равно его значению на изоповерхности. При этом возникает негрубая ситуация, сильно усложняющая весь алгоритм. Разработанные версии алгоритма избегают этой проблемы, добавляя малые слагаемые к тем узловым значениям поля, которые в точности равны значению на изоповерхности.

Алгоритм создания изоповерхности состоит из следующих шагов.

1. Определение диапазона $[f_{\min},\ f_{\max}]$ узловых значений поля f.

2. Добавление малых слагаемых к узловым значениям, совпадающим с заданным на изоповерхности f_0. Величина слагаемого выбирается равной εf_0, если $f_0 \neq 0$ и $\varepsilon\,(f_{\max} - f_{\min})$, если $f_0 = 0$. Величина ε принимается равной 10^{-7}, так как вычисления производятся в числах с плавающей запятой с одинарной точностью. Такое изменение поля не сказывается на видимой геометрии изоповерхности, но существенно упрощает алгоритм, устраняя негрубые ситуации.

3. В цикле по всем ребрам сетки домена выясняется, пересекается ли ребро с изоповерхностью. Если это так, ребру ставится в соответствие очередной номер узла сетки изоповерхности. Также вычисляется параметр от 0 до 1, определяющий положение этого узла на ребре.

4. Создание треугольников изоповерхности. На этом этапе необходимо обойти все тетраэдры, пересекающиеся с изоповерхностью, и сгенерировать для каждого из них обходы одного или двух треугольников, являющихся частью изоповерхности в данном тетраэдре. Чтобы обеспечить согласованность ориентации треугольников на соседних тетраэдрах, достаточно располагать узловыми значениями поля в каждом отдельном тетраэдре (предполагается, однако, что ориентации всех тетраэдров исходной сетки согласованы).

5. Создание узлов изоповерхности. На этом этапе вычисляются координаты узлов изоповерхности, фактически найденных на шаге 3.

6. Построение соответствия между узлами на краю изоповерхности и ребрами сетки домена на его границе. Этот шаг необходим для последующей сшивки кусков изоповерхностей на соседних доменах. Указанное соответствие позволяет при сшивке определить соответствующие друг другу узлы алгебраическим путем, не сравнивая их координаты.

Отметим, что реализация этого алгоритма на GPU нетривиальна, но может быть полностью сведена к последовательности стандартных алгоритмов `for_each`, `transform`, `partition`, `sort`, `scan`, `gather`, `scatter`, `unique`, `remove_if`, `copy` [1]. Разработанное программное обеспечение использует библиотеку Thrust [3], предоставляющую параллельные реализации этих алгоритмов для GPU.

3. Редуцирование изоповерхности и сшивка кусков

Предлагаемый алгоритм позволяет существенно уменьшить размер сетки, представляющей изоповерхность и полученной на предыдущем этапе. Основная операция, производимая над сеткой изоповерхности — *удаление ребра* (рис. 2), то есть стягивание ребра в узел. Кроме этого, некоторые пары соседних треугольников заменяются другими парами, в которых общее ребро проходит иначе (рис. 3); отметим, что каждая такая пара идентифицируется некоторым ребром сетки изоповерхности. Вторую операцию будем называть *заменой ребра*. Возможность выполнения одной из двух указанных операций на ребре сет-

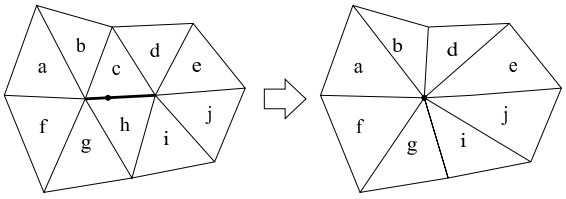


Рис. 2. Удаление ребра сетки

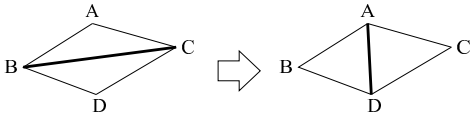


Рис. 3. Замена ребра сетки

ки определяется значением критерия Φ, вычисляемого на этом ребре (о нем речь пойдет ниже). Существенно, что выполнение операции на ребре влияет на значение критерия на близких ребрах. При разработке алгоритма предполагалась возможность его распараллеливания (с использованием GPU), и это наложило определенные требования на порядок выполняемых действий. Алгоритм выполняет последовательность итераций, на каждой из которых выполняются следующие шаги.

1. Вычисление значения критерия Φ_i на i-том ребре, для всех ребер сетки изоповерхности. Эта операция может быть выполнена параллельно (в частности, в реализации на GPU для каждого ребра используется отдельный поток выполнения). Ребра, на которых Φ_i превышает некоторое Φ^*, определяющее качество редуцированной изоповерхности, потенциально подходят для удаления (или замены, что для каждого ребра определяется отдельным флажком). Будем обозначать символом G^* множество ребер g_i, на которых $\Phi_i > Phi^*$. Отметим, что в случае удаления ребра также вычисляется параметр, определяющий положение узла, которое заменит это ребро.

2. Выбор подмножества G_1^* ребер, которые можно удалить или заменить одновременно, из G^*. Возможность одновременного удаления таких ребер подразумевает, что удаление любого из ребер, принадлежащих G_1^*, не влияет на значение критерия на любом другом ребре этого подмножества. Для выбранного нами критерия это, в свою очередь, означает, что в графе, образованном узлами и ребрами сетки изоповерхности, длина пути от любого узла, принадлежащего ребру из G_1^*, до любого узла, принадлежащего другому узлу из G_1^*, должна быть не менее двух. В настоящее время реализован лишь последовательный алгоритм выбора такого подмножества.

3. Выполнение операции удаления или замены для всех ребер из G_1^*. Эта операция может быть произведена параллельно.

Итерации продолжаются до тех пор, пока G_1^* не станет пустым, или же пока не возникнет двух идущих подряд итераций, на которых не удалено ни одно ребро.

Для вычисления критерия Φ_i на ребре g_i используется информация обо всех гранях изоповерхности, содержащих узлы этого ребра. Назовем *листом* с центром в k-том узле, L_k, множество всех граней изоповерхности, содержащих этот узел. Значение Φ_i определяется объединением листов $L_{g_{i,1}} \cup L_{g_{i,2}}$, где $g_{i,1}$ и $g_{i,2}$ — номера узлов на концах ребра g_i (рис. 4). Обозначим также через $e_{g_{i,1}}$ и $e_{g_{i,2}}$ номера граней, содержащих ребро g_i. Отметим, что алгоритм создания изоповерхности гарантирует, что каждое ребро принадлежит либо двум граням, либо (для ребер на краю сетки) одной.

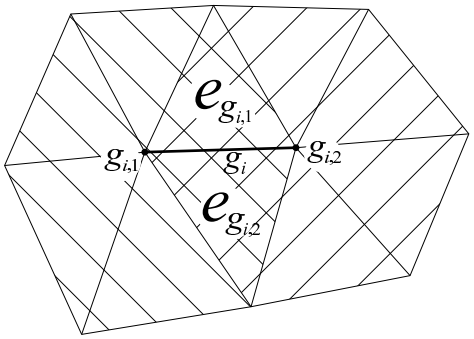


Рис. 4. Листы с центрами в узлах ребра; грани, содержащие ребро

Процедура вычисления критерия Φ_i на ребре g_i довольно разветвленная. Наиболее часто значение вычисляется на основании оценки локальной кривизны листов $L_{g_{i,1}}$ и $L_{g_{i,2}}$. Рассмотрим для примера внутреннее ребро — оно принадлежит двум граням. Единичные нормали к этим граням обозначим $\boldsymbol{n}_1$ и $\boldsymbol{n}_2$. Каждый из листов $L_{g_{i,j}}$ $(j = 1, 2)$ разделяется на две части $L_{g_{i,j},1}$ и $L_{g_{i,j},2}$ следующим образом. В первую часть попадает грань $e_{g_{i,1}}$. Далее обходятся соседние грани этого листа, причем в такую сторону, чтобы следующей за $e_{g_{i,1}}$ гранью была *не* грань $e_{g_{i,2}}$. В первую часть попадут все идущие подряд грани, нормали к которым ближе к $\boldsymbol{n}_1$, нежели к $\boldsymbol{n}_2$. Остальные грани попадут во вторую часть. Обозначим множества единичных нормалей к граням из первой и второй частей как $\boldsymbol{n}_{g_{i,j},1}$, $\boldsymbol{n}_{g_{i,j},2}$. Для каждого листа вычисляется

$$\Phi_{i,j} = \min_{s=1,2} \left\{ \min_{\boldsymbol{n} \in \boldsymbol{n}_{g_{i,j},s}} \{\boldsymbol{n}_s \cdot \boldsymbol{n}\} \right\}, \qquad j = 1, 2$$

Значение $\Phi_{i,j}$ окажется тем больше, чем ближе нормали граней из $L_{g_{i,j},1}$ к $\boldsymbol{n}_1$, а граней из $L_{g_{i,j},2}$ — к $\boldsymbol{n}_2$. Оно достигает единицы для плоских и для «согнутых» листов (рис. 5). Наконец, значение критерия Φ_i на ребре вычисляется по формуле

$$\Phi_i = \Phi_{i,1} t_i + \Phi_{i,2}(1 - t_i), \qquad t_i = \frac{2}{\pi} \operatorname{arctg} \frac{\Phi_{i,2}}{\Phi_{i,1}}.$$

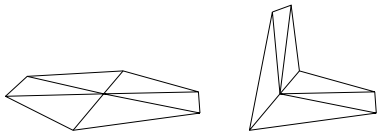


Рис. 5. Плоский и «согнутый» листы

Параметр t_i определяет положение узла, который заменит ребро при его удалении.

Использование указанных формул для вычисления Φ_i позволяет редуцировать поверхности, имеющие углы, удаляя ребра, лежащие на этих углах (а не только на «плоской» части сетки).

При вычислении значений критерия на ребрах иногда реализуются и другие ветки процедуры, позволяющие удалить очень короткие ребра; особо рассмотреть ребра с листами из трех граней; особо рассмотреть ребра, лежащие на границе; заменить длинные ребра между очень узкими треугольникам. В данной статье не представляется возможным описать все эти ветки, однако их наличие позволяет существенно повысить качество и уменьшить количество граней редуцированной поверхности. Кроме того, производится ряд тестов, отбраковывающих ребра, удаление которых привело бы к нарушению топологии сетки или появлению почти вырожденных граней. Благодаря этому гарантируется гомеоморфизм исходной и редуцированной поверхностей.

Отметим еще, что алгоритм редуцирования предусматривает возможность запретить удаление произвольного подмножества узлов на краю поверхности. Она требуется для обеспечения возможности сшивки кусков изоповерхности на соседних доменах.

Сшивка пары кусков изоповерхности сводится к созданию единой нумерации узлов и граней с учетом того, что часть узлов, принадлежащая интерфейсной границе между доменами принадлежит обоим кускам. Кроме того, после сшивки производится повторное редуцирование, цель которого — проредить сетку в месте, где проходит «шов».

4. Балансировка и планирование заданий

При разработке приложения для мультипроцессорной системы неминуемо возникает задача балансировки загрузки созданных процессов. В настоящей задаче это достигается простым и в то же время эффективным механизмом динамического распределения работы между MPI процессами. Один MPI процесс — управляющий, его задача состоит только в распределении работы между остальными процессами. Рабочий процесс оповещает о своей готовности управляющего и в ответном сообщении получает задание. По выполнении задания он отсылает главному процессу результат и опять ждет сообщение с новым заданием, либо команду завершения. Управляющий же процесс по запросу раздает работу из очереди заданий. В силу специфики рассматриваемой задачи такой подход весьма эффективен и позволяет свести практически к нулю время простоя вычислительных мощностей.

Однако использование гетерогенных вычислительных систем, таких как кластер с установленными на узлах GPU создает дополнительную проблему распределения заданий между процессорами для получения максимального быстродействия. Для достижения этой цели можно было бы создать менеджер ресурсов, позволяющий процессу на узле принять решение, какое устройство (GPU или CPU) следует использовать для минимизации времени выполнения текущей задачи; менеджер предсказывал бы время выполнения операции на основании накопленной статистики о ранее завершенных заданиях, а также вел бы учет загрузки устройств узла. В случае нехватки ресурсов рабочий процесс переводится в состояние ожидания до их освобождения.

Описанный вариант менеджера ресурсов был разработан, однако в рассматриваемой нами задаче от него пришлось отказаться в пользу более простой модификации, оказавшейся

в то же время более эффективной. Работа менеджера сводится к назначению фиксированного ресурса для каждого рабочего процесса, так что одни процессы используют только CPU, а другие — только GPU, причем количество последних не превышает количества GPU в системе. Более высокая эффективность упрощенного менеджера ресурсов связана с отсутствием необходимости перевода рабочих процессов в режим ожидания.

Менеджер ресурсов — один на узел, поэтому он обслуживает все рабочие процессы, запущенные на этом узле.

5. Тестирование программной реализации

Программная реализация алгоритмов построения редуцированной изоповерхности тестировалась на узле с 12 Гб оперативной памяти, 16 ядрами CPU и двумя GPU Tesla с 4 Гб памяти на каждом.

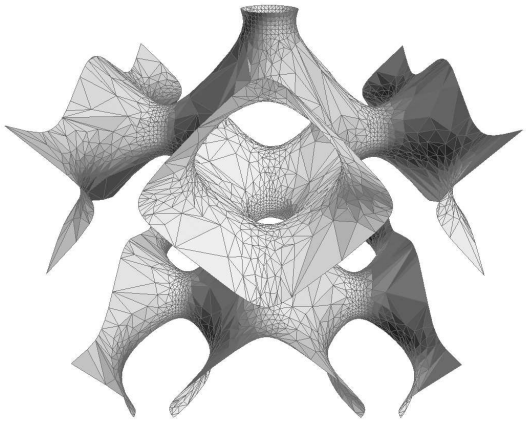


Рис. 6. Фрагмент тестовой изоповерхности

В качестве теста рассмотрена область в виде куба с размерами $5 \times 5 \times 5$ с заданной на ней сеткой $500 \times 500 \times 500$ узлов и полем

$$f(x, y, z) = 2\cos(10x) + 2\sin(10y) + \cos(10z);$$

изоповерхность строилась для уровня $f = 0,5$; её фрагмент представлен на рис. 6. Проводились три серии тестов, отличающихся размерами домена:

- 250 доменов по $5 \cdot 10^5$ узлов;
- 125 доменов по 10^6 узлов;
- 63 домена по $2 \cdot 10^6$ узлов.

Благодаря специальному выбору поля f в каждом из этих случаев в домены попали примерно одинаковые по размеру куски изоповерхности.

Приведенные далее графики относятся к серии 125 доменов по 10^6 узлов; отметим, что зависимость быстродействия программы от размера домена слабая; с ростом размера домена она незначительно увеличивается.

Были рассмотрены зависимости времени обработки всех доменов от количества рабочих MPI-процессов и от размера домена. Их графики представлены на рис. 7. Выигрыш при использовании GPU очевиден, особенно это заметно при малом количестве рабочих процессов. Отметим, что время сшивки изоповерхности во всех случаях невелико и не превышает 10% от времени обработки доменов.

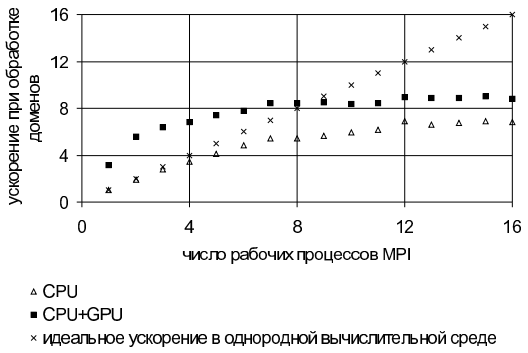


Рис. 7. Ускорение расчетов в зависимости от количества рабочих процессов MPI

6. Заключение

Тестирование разработанной реализации алгоритмов построения редуцированной изоповерхности показало её работоспособность и масштабируемость, пока число процессоров остается значительно меньше числа доменов.

Анализ результатов тестирования показал, что уменьшение прироста производительности с ростом числа используемых CPU скорее всего связано с пропускной способностью подсистемы оперативной памяти. Чем больше запущено процессов, тем больше нагрузка на нее.

Использование имеющихся в системе GPU позволяет значительно увеличить скорость расчета. Алгоритм редуцирования изоповерхности выполняется на GPU намного быстрее, чем на CPU, хотя и не в десятки раз. Отчасти это связано с необходимостью частых обменов данных с GPU, отчасти — с дополнительными операциями, требующимися в GPU-версии алгоритма.

Список литературы

1. S. Gorlatch, C. Lengauer. (De)Composition for Parallel Scan and Reduction. mppm, pp.23, Massively Parallel Programming Models, 1997.

2. Burkhard Wuensche. A Survey and Evaluation of Mesh Reduction Techniques. Proceedings of IVCNZ '98, Auckland University, Auckland, November 1998, pages 393–398.

3. Thrust — a CUDA library of parallel algorithms. http://code.google.com/p/thrust/

Моделирование ударных процессов в тканевых бронежилетах и теле человека на вычислительном кластере «СКИФ Урал»*

Н.Ю. Долганина, С.Б. Сапожников, А.А. Маричева

Работа посвящена численному моделированию ударных процессов в тканевых бронежилетах и теле человека. Построена модель скелета грудной клетки человека. Проведены численные эксперименты по исследованию масштабируемости задач динамического взаимодействия индентора с тканевыми преградами различных размеров и разным количеством слоев, а также динамического взаимодействия индентора со скелетом грудной клетки человека при помощи пакета программ LS-DYNA.

1. Введение

Основной задачей при проектировании бронежилетов является минимизация их массы при сохранении заданного уровня защиты. Проверка качества бронежилета не находящегося в контакте с защищаемым объектом проводится с определением баллистического предела [1]. А если бронежилет контактирует с защищаемым объектом (тело человека), то в этом случае существует критерий определения тупой травмы, который применяется для сравнения бронежилетов различных классов (рис. 1) [2,3].

1 - многослойная тканевая преграда (бронежилет); 2 - регистрирующая среда; W - фактический прогиб.
Рис. 1. Сертификационные испытания бронежилетов

В экспериментах в качестве тела человека используют либо технический пластилин (при этом довольно сложно оценить степень травмирования тела человека), либо дорогостоящие экспериментальные модели грудной клетки [4]. Экспериментально-аналитический путь оптимизации конструкции многослойных тканевых преград позволяет достаточно быстро определить оптимальное соотношение параметров для фиксированного воздействия (конкретных формы индентора и скорости нагружения), однако этот метод весьма затратный [5].

Чисто аналитических моделей, точно описывающих процесс динамического взаимодействия пули и бронежилета с учетом разрушения, на данный момент не существует и, очевидно, их получение невозможно из-за сложности физических явлений, происходящих в этом процессе: большие перемещения, скольжение, фрикционные контакты, повышение температуры. Для того чтобы учесть эти сложные физические явления, необходимо учитывать структуру баллистической ткани. В России при изготовлении бронежилетов используют баллистические ткани различных типов переплетения: полотняное, саржевое, сатиновое. В то же время нити в баллистической ткани состоят из множества волокон. В первых работах, посвященных численному исследованию взаимодействия бронежилета с пулей, баллистические ткани заменяли мембраной, затем использовали сетки со связанными узлами. Для исследования поведения конструкций имеющих тканую структуру в ряде работ из ткани выделяют повторяющийся элемент, моделируют его с высокой степенью точности, изучают его свойства при различных видах нагружения, затем полученные параметры используют для расчета таких конструкций, моделируя их

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проект 10-07-96007-р_урал_а).

как сплошную среду [6]. Такой подход применительно к расчету взаимодействия бронежилета с пулей не подходит, т.к. в этом случае происходит значительное вытягивание и разрыв нитей. С развитием вычислительных возможностей в настоящее время можно моделировать баллистическую ткань с точностью до нити [7, 8, 9]. В некоторых работах каждая нить моделируется очень точно с учетом формы поперечного сечения объемными элементами [7], но данные модели содержат большое количество конечных элементов, что не позволяет рассчитать пакеты больших размеров (30х30 см) с несколькими слоями ткани даже с использованием суперкомпьютеров.

При разработке бронежилетов необходимо иметь представление о механизме повреждений, которые возникают в теле человека при локальных ударных воздействиях. Поэтому учеными разных стран ведется работа по созданию теоретических и экспериментальных моделей тела человека, которые в точности повторяют форму человеческого тела и обладают такими же свойствами. Учеными Университета имени Джона Хопкинса в США (Вашингтон) были созданы конечно-элементная и экспериментальная модели грудной клетки человека, были построены ребра, грудина, хрящи, позвоночник, сердце, легкие, печень, желудок, мышцы и кожа [4]. Если значения ускорений, полученные экспериментально и с помощью расчета, близки, то отличие давлений существенно. Таким образом, разработки теоретических и экспериментальных моделей грудной клетки человека активно продолжаются, однако какие-либо достоверные данные пока получены не были. К тому же были созданы только модели деформирования тела человека без учета степени травмирования.

Для того чтобы использовать численную модель грудной клетки человека для проектирования бронежилетов необходимо знать механические свойства всех ее элементов. Идентификацию параметров грудной клетки можно провести, сопоставив экспериментальные и расчетные перемещения при статическом нагружении, и ускорений, спектра собственных частот колебаний при динамическом нагружении. При этом динамическое нагружение грудной клетки реального человека должно быть низкоскоростным, чтобы не нанести травм человеку.

Реальный бронежилет 2-го класса [2] состоит из 60-70 слоев баллистической ткани. Типичный размер бронепанели тканевого бронежилета составляет 30х30 см. Для ткани полотняного переплетения количество точек контакта нитей утка и основы составляет 400 тыс. в одной ткани. Для всего бронежилета (65 слоев) количество контактов возрастает до 25 млн. И это не считая контактов между слоями в процессе ударного взаимодействия. По различным оценкам общее число контактов может достичь 40 млн. Такие задачи не могут быть в принципе решены на персональных компьютерах. Даже на высокопроизводительных вычислительных кластерах придется вводить некоторые упрощения. Эти упрощения сводятся к замене группы одинаковых по конструкции слоев одним, эквивалентным по массе. Основная проблема состоит в определении максимального количества слоев в группе без потери качества результата расчета. Имеющиеся данные по прогнозу величины баллистического предела малослойных (до 10 слоев) тканевых пакетов показывают, что наличие трех эквивалентных слоев (вместо 10 оригинальных) позволяет получить ошибку расчета в пределах 2%, что приемлемо, учитывая высокую ответственность таких изделий [10]. Следует отметить, что бронепанели, имеющие в своем составе до 10 слоев можно моделировать и рассчитывать на вычислительных кластерах типа «СКИФ Урал» без введения упрощений типа объединения ряда слоев в один эквивалентный по массе.

В настоящей статье мы рассматриваем моделирование динамического взаимодействия индентора с тканевыми преградами различных размеров и разным количеством слоев, а также динамического взаимодействия индентора со скелетом грудной клетки человека на вычислительном кластере «СКИФ Урал». Статья организована следующим образом. В разделе 2 приведена постановка задачи. В разделе 3 описываются методы исследования, и приводится описание задачи. В разделе 4 обсуждаются результаты проведенных экспериментов на вычислительном кластере. В заключении суммируются основные результаты, полученные в данной работе.

2. Постановка задачи

Были рассмотрены тканевые пакеты полотняного переплетения (рис. 2) из одного и пяти слоев ткани размером 5х5 см, а также размером 30х30 см. В работе использована арамидная

ткань СВМ арт. 5601. В расчетной модели нити имеют относительную свободу перемещения с возможностью вытягивания с учетом сухого трения. Рассматривали нити, которые имеют прямоугольное поперечное сечение и были представлены одним оболочечным элементом по ширине с одной точкой интегрирования по толщине и выполнены из ортотропного материала с малыми поперечно-сдвиговыми свойствами. Нити в расчетной модели могли разрушаться. Края ткани не были закреплены. В расчете индентор имел форму цилиндра с полусферическим основанием диаметром 7 мм, массой 5,5 г, с начальной скоростью 450 м/с (имитация пули пистолета ТТ) и был выполнен из абсолютно жесткого материала.

При создании модели грудной клетки человека были введены следующие упрощения: ребро представляло собой вытянутое по определенной траектории тело с эллиптическим поперечным сечением; поперечный размер ребра плавно увеличивался от позвонка к реберному хрящу; каждый позвонок имел форму призмы с эллиптическим основанием. Размеры позвонков увеличивались от верхних к нижним; поперечные, суставные и остистые отростки были исключены; мечевидный отросток грудины был исключен. В расчетах предполагали, что каждое ребро жестко соединено с верхней частью одного позвонка; реберные хрящи истинных ребер жестко соединены с грудиной; реберный хрящ каждого ложного ребра жестко соединен с верхним реберным хрящом. В модели были запрещены все перемещения и повороты в плоскости самого нижнего поясничного позвонка, также были запрещены перемещения вдоль оси нагружения 5, 6, 7 позвонков сверху. Индентор имел форму цилиндра диаметром 3 см, массой 1 кг и был выполнен из абсолютно жесткого материала, его начальная скорость равнялась 5 м/с.

Туловище человека принято делить на три отдела: верхний, средний и нижний. В расчетах используется масса только верхнего и половины среднего отделов. Статистические исследования показывают, что масса верхнего отдела составляет 15,9% от массы тела, масса среднего отдела – 16,3%. Модель грудной клетки человека соответствует человеку ростом 150 см с обхватом грудной клетки 640 мм. Вес такого человека около 50 кг. Тогда масса двух отделов туловища составляет 12 кг. Это вес костей, мышц, кожи и внутренних органов. При расчете все внутренние органы, а также некоторые кости, которые расположены в верхнем отделе туловища (лопатки, ключицы и плечевые кости), не рассматривали. В итоге вес скелета грудной клетки, кожи и мышц оказался равен 5,0 кг. Чтобы в расчете учесть массу мягких тканей, плотность кости была увеличена.

3. Методы исследования

Для решения задачи динамического взаимодействия индентора с тканевыми преградами с помощью пакета программ LS-DYNA геометрия и сетка конечных элементов были созданы в пакете программ ANSYS. Повторяющийся элемент ткани показан на рис. 3, где отмечены номера ключевых точек, координаты которых были введены в ANSYS, после чего по ключевым точкам были заданы соответствующие поверхности. Далее набор поверхностей был размножен до получения необходимых размеров модели, после чего была построена сетка конечных элементов (рис. 4).

Количество элементов, получившееся при создании сетки, представлено в таблице 1.

Таблица 1. Количество элементов.

	5x5 см, 1слой	5x5 см, 5слоев	30x30см, 1слой	30x30см, 5слоев
Количество элементов	20 000	77 600	547 296	2 714 080

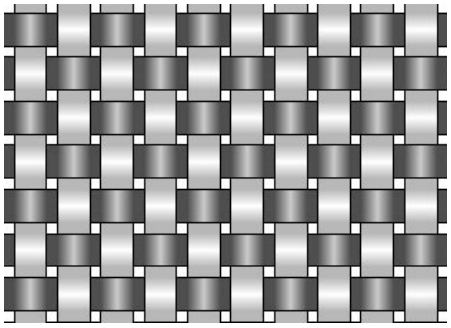


Рис. 2. Полотняное переплетение

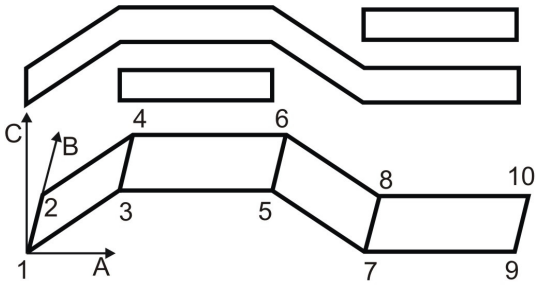


Рис. 3. Повторяющийся элемент

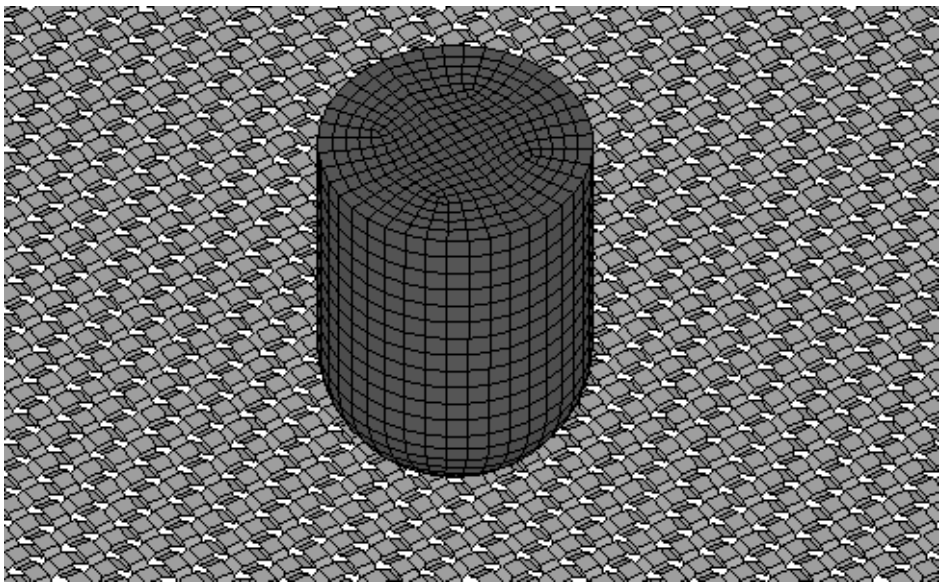


Рис. 4. Сетка конечных элементов (полотняное переплетение)

Из списка материалов, предложенных пакетом программ LS-DYNA [11], для индентора был выбран *MAT_RIGID для нитей – *MAT_ENHANCED_ COMPOSITE_DAMAGE. Характеристики нитей для ввода в программу LS-DYNA приведены в табл. 2. Принято, что разрушение нити происходит при достижении в ней заданной величины первого главного напряжения 3 ГПа (определено экспериментально). Механические свойства нитей и тканей определены на универсальной испытательной машине Instron 5882.

Таблица 2. Характеристики нитей.

Параметр	Обозначение	Величина
Толщина нити, мкм	T	100
Ширина нити, мкм	D	500
Модули упругости, МПа	EA	$1{,}3 \cdot 10^5$
	EB	$1 \cdot 10^3$
Плотность, кг/м^3	ρ	1 440
Коэффициент Пуассона	μ_{AB}	0,003
Модули сдвига, МПа	GAB	$1 \cdot 10^3$
	GBC	$1 \cdot 10^3$
	GCA	$1 \cdot 10^3$

Контакт объектов моделировался командой *CONTACT_AUTOMATIC_SURFACE_ TO_SURFACE с коэффициентом трения 0,4, характерным для типичных нитей СВМ, применяемых в бронежилетах. Расчет динамики был проведен от момента удара до времени t = 60 мкс.

Верификация модели была проведена ранее в работе [10], результаты численных расчетов отлично согласуются с экспериментальными исследованиями.

На примере пакета из 5 слоев ткани размером 5х5 см были рассмотрены три различных способа декомпозиции модели на шестнадцати процессорных ядрах. В первом случае модель была разбита на прямоугольные области, проходящие через всю толщину пакета (рис. 5); во втором – на полосы, расположенные вдоль одной из сторон пакета и также проходящие через всю толщину пакета (рис. 6); в третьем случае – декомпозиция была проведена также на полосы расположенные вдоль одной из сторон пакета, при этом каждая полоса разбивалась еще и по толщине пакета (рис. 7).

Было получено, что время расчета при использовании 1-го и 2-го способа декомпозиции одинаково, а для 3-го в 2,5 раза больше чем в первых двух. Это объясняется тем, что контакт между слоями обрабатывается на разных ядрах, что повышает межпроцессорные обмены и увеличивает время расчета. В результате все дальнейшие расчеты были проведены с использованием 1-го способа декомпозиции (рис. 5).

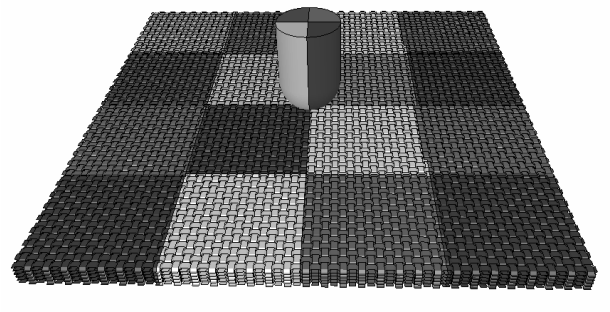


Рис. 5. Первый способ декомпозиции модели (16 ядер)

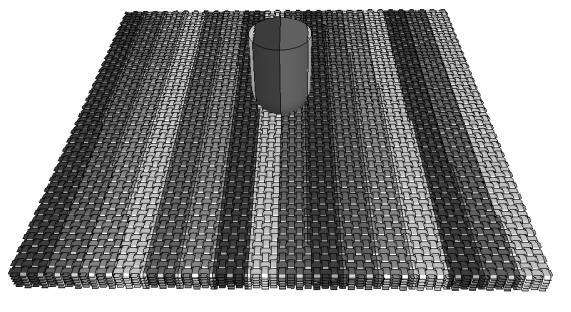
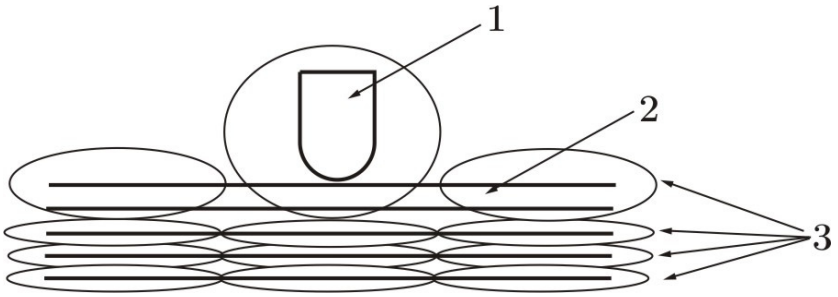


Рис. 6. Второй способ декомпозиции модели (16 ядер)



1 - пуля; 2 - пакет из 5 слоев ткани размером 5х5 см (вид сбоку); 3 - области декомпозиции.

Рис. 7. Третий способ декомпозиции модели (16 ядер)

С помощью пакета SolidWorks построена модель скелета грудной клетки. Смоделированы ребра (12 пар), грудина, грудные позвонки (12), поясничные позвонки (5), межпозвоночные диски и реберные хрящи (рис. 8).

Для решения задачи с помощью пакета программ LS-DYNA сетка конечных элементов была создана в пакете программ ANSYS (рис. 9). Количество элементов, получившееся при создании сетки равно 211 112.

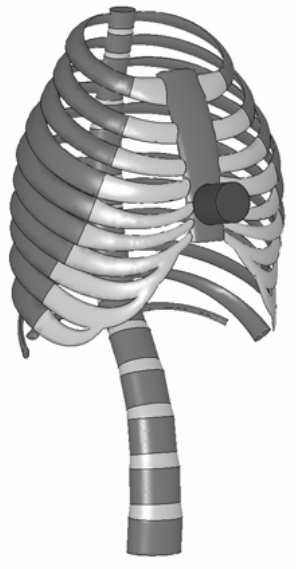


Рис. 8. Модель скелета грудной клетки человека

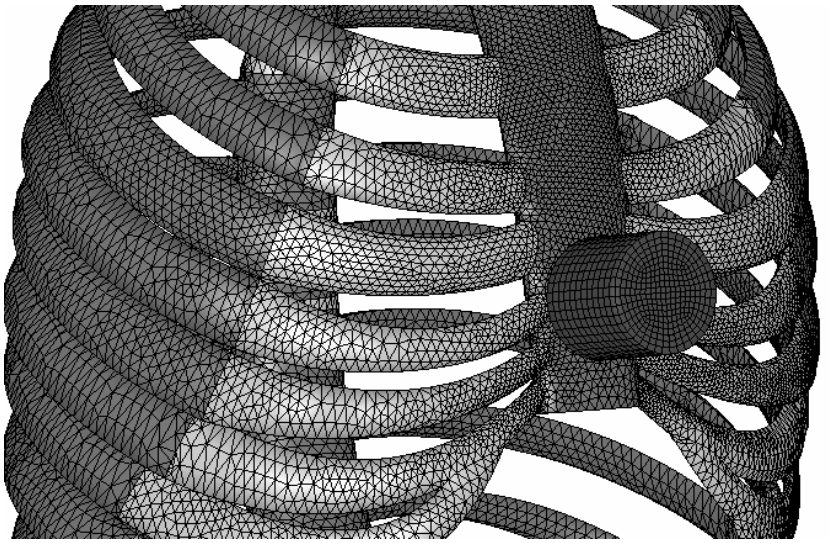


Рис. 9. Сетка конечных элементов

Модули упругости материалов костей и хрящей представлены в таблице 3 и были взяты из литературы [12], при этом в расчете плотность кости увеличили для того чтобы учесть массу мягких тканей. Из списка материалов, предложенных пакетом программ LS-DYNA [11], для индентора был выбран *MAT_RIGID для костей и хрящей *MAT_ELASTIC (упругий). Расчет динамики был проведен от момента удара до времени t = 30 мс.

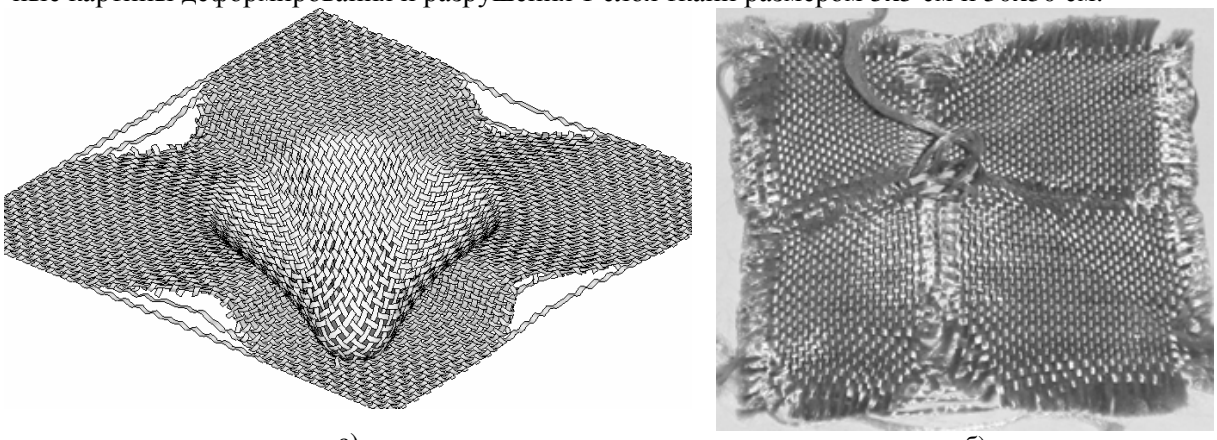
Таблица 3. Механические свойства материалов кости и хряща.

Материал	Плотность ρ, кг/м^3	Модуль упругости E, МПа	Коэффициент Пуассона
кость	15 000	8 000	0,3
хрящ	1 800	40	0,4

4. Результаты исследований и их анализ

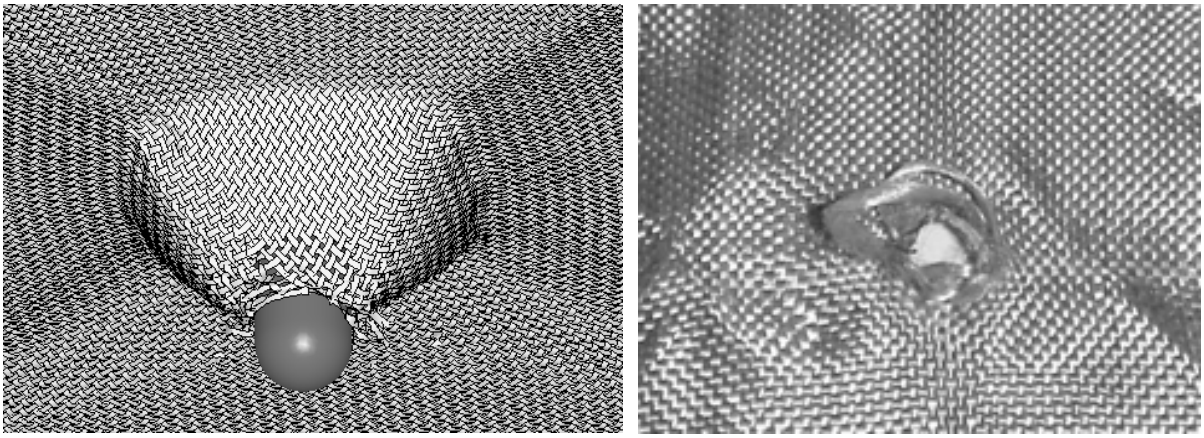
Расчеты были проведены на высокопроизводительном вычислительном кластере «СКИФ Урал» [13], оснащенном 166 вычислительными узлами с 2 процессорами Intel Xeon E5472 (4 ядра по 3.0 ГГц) и 8 ГБ оперативной памяти на каждом узле.

Расчеты были проведены с использованием одного, двух, четырех и восьми ядер с узла вычислительного кластера «СКИФ Урал». На рис. 10 и 11 показаны расчетные и экспериментальные картины деформирования и разрушения 1 слоя ткани размером 5х5 см и 30х30 см.



а) б)

Рис. 10. Сопоставление расчетных а) и экспериментальных б) данных по деформированию однослойных образцов тканей размером 5х5 см



а) б)

Рис. 11. Сопоставление расчетных а) и экспериментальных б) данных по пробою однослойных образцов тканей размером 30х30 см

Было получено, что тканевые пакеты размером 30х30 см пробиваются, а тканевые пакеты размером 5х5 см не пробиваются. Это связано с тем, что после контакта пули с тканевым пакетом первое главное напряжение в нитях постоянно возрастает, в небольших по размеру пакетах ударная волна быстрее доходит до края пакета и возвращается, разгружая место контакта.

На примере задачи с одним слоем ткани размером 30х30 см показано сравнение ускорений при расчете с использованием одного, двух, четырех и восьми ядер с узла вычислительного кластера «СКИФ Урал» (рис. 12). Ускорение вычислялось по формуле T1/Tn, где T1 – время, затраченное на решение теста на восьми процессорных ядрах, Tn – время, затраченное на решение этой же задачи на n процессорных ядрах.

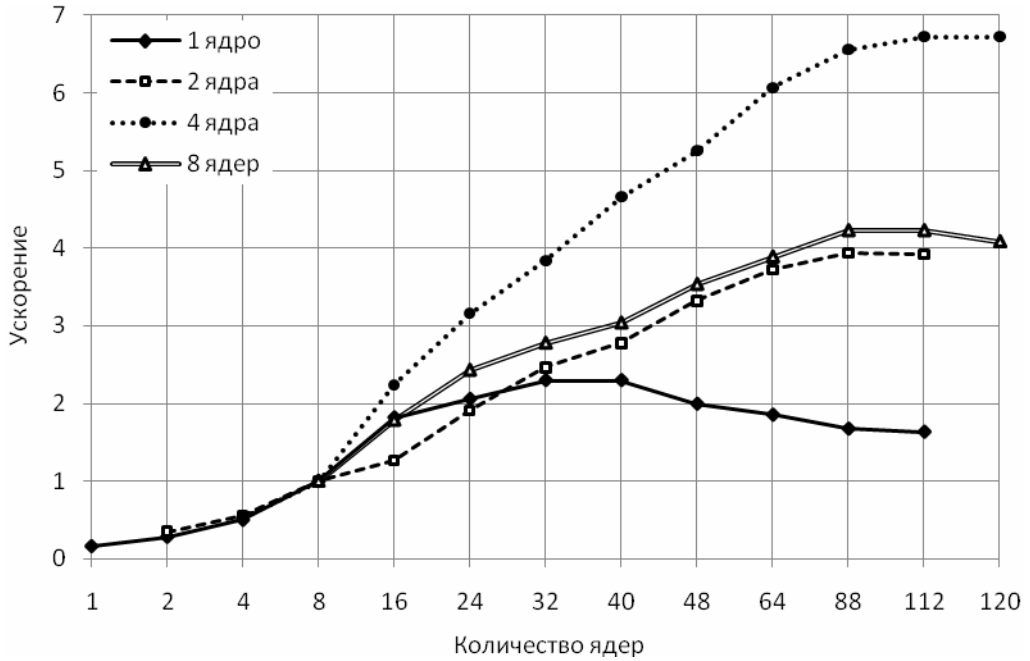


Рис. 12. Сравнение ускорений на вычислительном кластере «СКИФ Урал» при использовании 1, 2, 4 и 8 ядер с узла (1 слой ткани размером 30х30 см)

Было получено, что наилучшая масштабируемость получается при использовании четырех ядер с узла. Время расчета при использовании 4-х ядер с узла в среднем на 20% меньше по сравнению со временем расчета при использовании 1, 2, 8 ядер с узла.

Графики ускорений при использовании 1, 2, 4, 8 ядер с узла качественно похожи. Поэтому приведены результаты на примере использования 4-х ядер с узла (рис. 13 и 14). На рис. 13 приведены графики ускорений, где T1=4, а на рис. 14 T1=24. Пакет из 5 слоев ткани размером

30x30 см не считается меньше чем на 24 ядрах и больше чем на 92 ядрах. Это вязано с выделением оперативной памяти на каждое ядро для решения задачи, а также с тем, что декомпозиция задачи производится на одном узле, на котором максимум 8 Гб оперативной памяти.

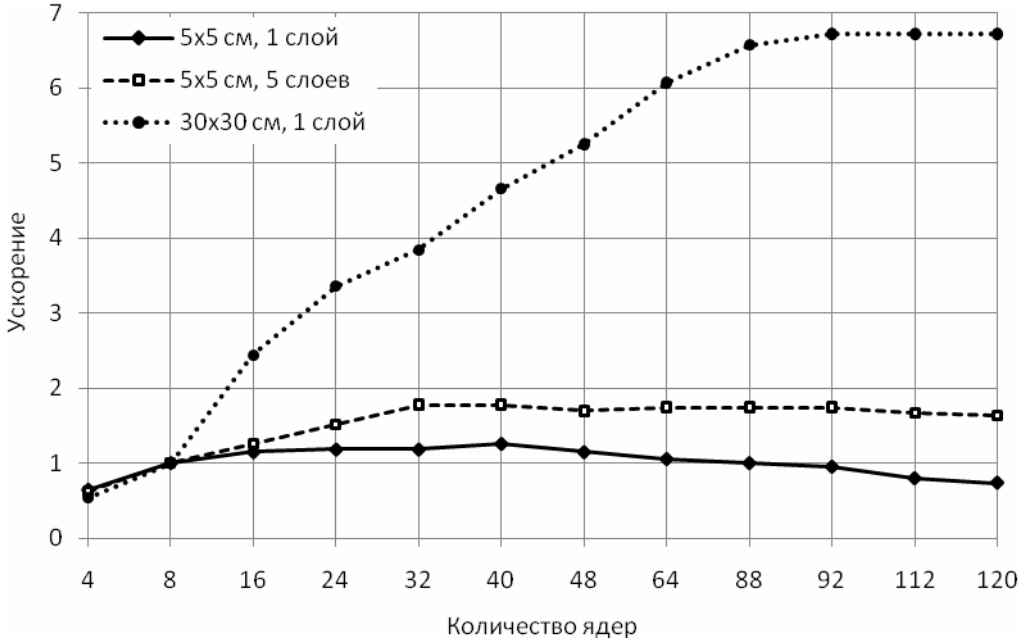


Рис. 13. Сравнение ускорений на вычислительном кластере «СКИФ Урал» (4 ядра с узла, T1=4)

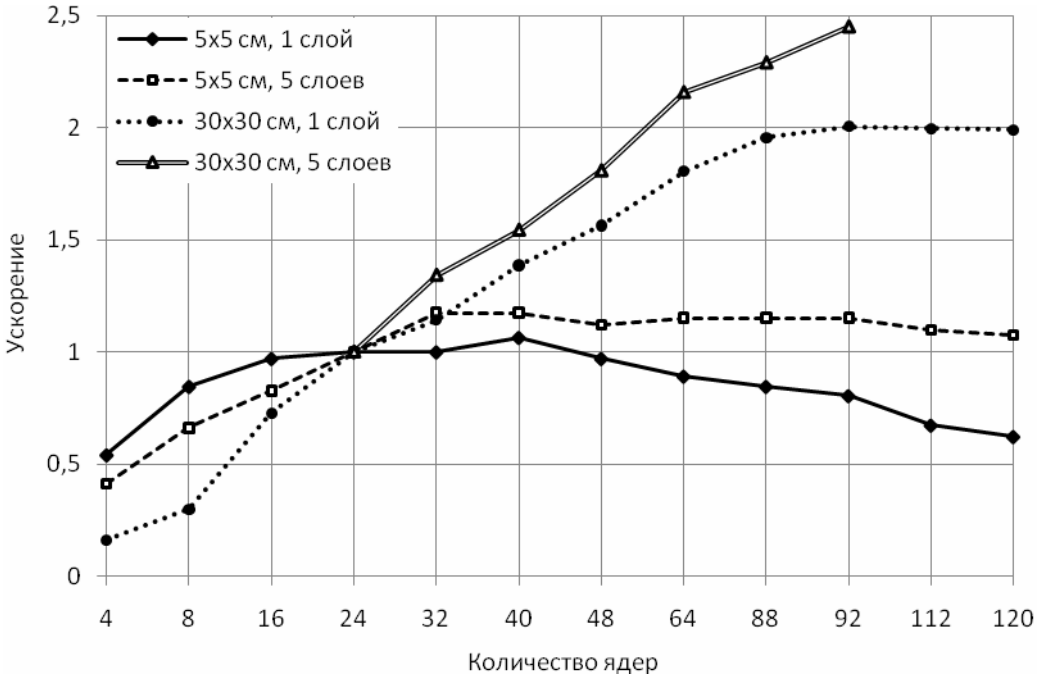


Рис. 14. Сравнение ускорений на вычислительном кластере «СКИФ Урал» (4 ядра с узла, T1=24)

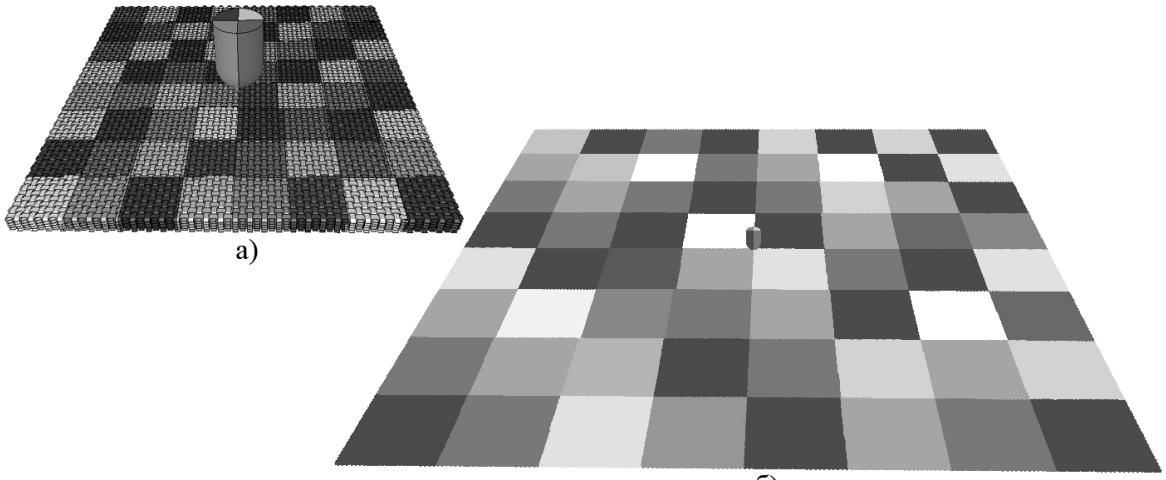
Время, за которое были произведены расчеты по динамическому взаимодействию индентора и различных тканевых пакетов с использованием 4-х ядер с узла, приведено в таблице 4.

Таблица 4. Время выполнения расчетов с использованием 4-х ядер с узла.

№ п.п.	Вычислительных ядер	Время выполнения, сек.			
		5x5 см, 1 слой	5x5 см, 5 слоев	30x30 см, 1 слой	30x30 см, 5 слоев
1	4	61	592	4 987	-

2	8	39	368	2 688	-
3	16	34	294	1 100	-
4	24	33	243	800	6 326
5	32	33	207	700	4 713
6	40	31	206	577	4 100
7	48	34	217	512	3 500
8	64	37	212	443	2 934
9	88	39	211	409	2 764
10	92	41	212	399	2 584
11	112	49	221	401	-
12	120	53	226	402	-

В расчетах рассматривалось одинаковое воздействие (пуля, ее скорость и время процесса) на различные объекты, отличающиеся размером и количеством слоев. В этих случаях результаты воздействия были принципиально разными: наличие или отсутствие сквозного пробоя, сохранение или исчезновение контакта с пулей, множественные контакты разорванных нитей с соседними нитями и даже с нитями из других слоев. Эти явления могут объяснить относительно хорошую масштабируемость для многослойных пакетов и сравнительно низкую для однослойных пакетов. Для пакетов больших размеров (30х30 см) относительно хорошая масштабируемость по сравнению с пакетами небольших размеров (5х5 см) объясняется тем, что на один процессор приходится больший объем материала и, соответственно, меньшие межпроцессорные обмены в зоне контакта индентора с тканевым пакетом, где происходит разрушение и вытягивание нитей (рис. 15).



а)

б)

а) пакет размером 5х5 см; б) пакет размером 30х30 см

Рис. 15. Декомпозиция модели (64 ядра).

Использование большого количества процессоров заставляет разбивать объект на небольшие зоны, в которых происходит разрушение нитей и множественные контакты между элементами из разных зон, что заставляет усиливать межпроцессорные обмены и снижает, в итоге, масштабируемость. В пакете программ LS-DYNA нет такой функции как сгущение при декомпозиции модели, с помощью которой можно было бы увеличить области декомпозиции возле контакта пули с тканевым пакетом.

Декомпозиция модели скелета грудной клетки человека на примере 16 ядер показана на рис. 16. Перемещения индентора и элементов грудной клетки показано на рис. 17.

Ускорение для задачи динамического взаимодействия индентора со скелетом грудной клетки человека вычислялось по формуле T1/Tn, где T1 – время, затраченное на решение теста на одном процессорном ядре, Tn – время, затраченное на решение этой же задачи на n процессорных ядрах (рис. 18).

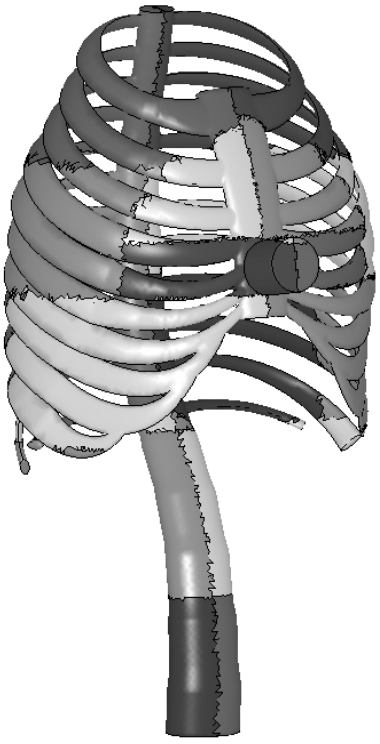


Рис. 16. Декомпозиция модели (16 ядер)

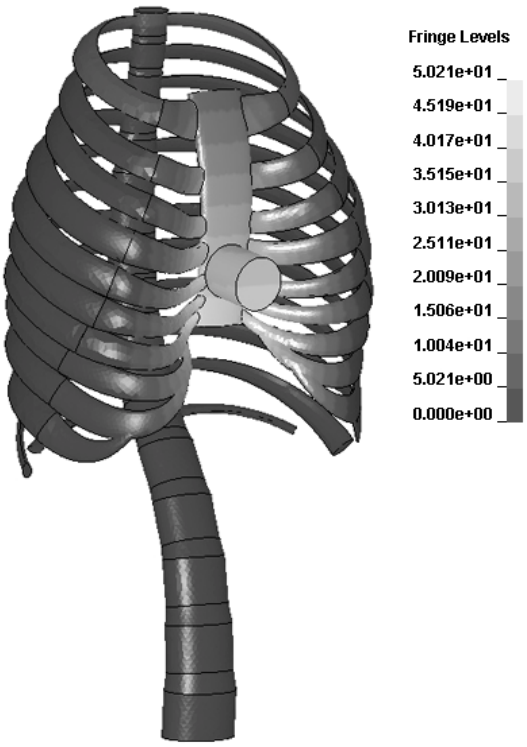


Рис. 17. Перемещения индентора и элементов грудной клетки (значения на шкале указаны в мм)

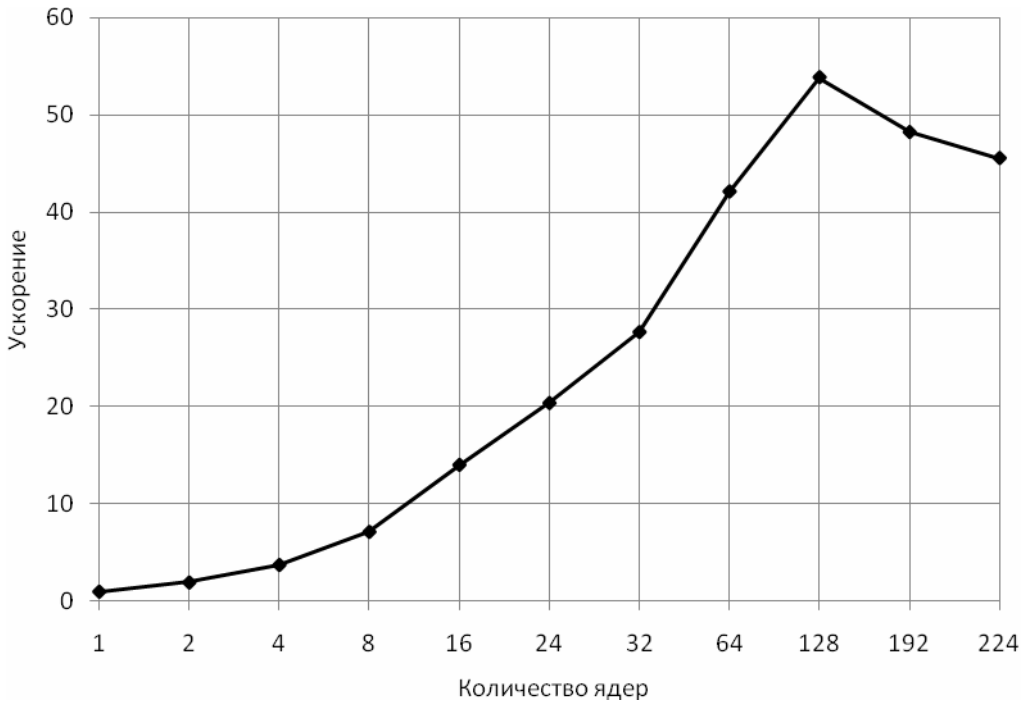


Рис. 18. Ускорение

Время, за которое были произведены расчеты по динамическому взаимодействию индентора и скелета грудной клетки человека, приведено в таблице 5.

Таблица 5. Время выполнения расчетов на вычислительном кластере «СКИФ Урал».

№ п.п.	Вычислительных ядер	Время выполнения, сек.
1	1	497 921

2	2	256 300
3	4	132 947
4	8	69 716
5	16	35 508
6	24	24 437
7	32	19 386
8	64	11 827
9	128	9 250
10	192	10 330
11	224	10 935

При рассмотрении задачи о деформировании грудной клетки при низкоскоростном ударе индентором отмечается хорошая масштабируемость вплоть до 128 ядер. Далее эффективность распараллеливания снижается из-за распределения индентора и области контакта с грудиной на несколько ядер, что приводит к увеличению межпроцессорных обменов и увеличению времени расчета.

5. Заключение

Вычислительные возможности кластеров позволяют решать сложные контактные задачи, в которых нельзя использовать механику сплошной среды, т.к. результаты получаются неадекватными. В таких задачах необходимо создавать геометрически подобную модель и рассматривать взаимодействие этой модели не только с внешними воздействиями, но и между элементами внутри этой модели. Это задачи ударного взаимодействия тканевого бронежилета с пулей, геологические задачи обработки почвы, моделирование схода лавин, компактирование (уплотнение) порошков и др. В задаче ударного взаимодействия тканевого бронежилета с пулей мы рассматривали нити как сплошную среду, в реальности же каждая нить состоит еще из порядка 150-200 волокон, которые также взаимодействуют между собой. В таких задачах важно создать такую геометрически подобную модель, чтобы результаты расчетов согласовывались с экспериментом, но в то же время не усложнять модель на столько, что ее невозможно будет решать и на суперкомпьютерах.

В данной работе были проведены численные эксперименты по исследованию масштабируемости задач динамического взаимодействия индентора с тканевыми преградами различных размеров и разным количеством слоев, а также динамического взаимодействия индентора со скелетом грудной клетки человека при помощи пакета программ LS-DYNA. Для пакетов больших размеров (30x30 см) относительно хорошая масштабируемость по сравнению с пакетами небольших размеров (5x5 см) объясняется тем, что на один процессор приходится больший объем материала и, соответственно, меньшие межпроцессорные обмены в зоне контакта индентора с тканевым пакетом, где происходит разрушение и вытягивание нитей. Использование большого количества процессоров заставляет разбивать объект на небольшие зоны, в которых происходит разрушение нитей и множественные контакты между элементами из разных зон, что заставляет усиливать межпроцессорные обмены и снижает, в итоге, масштабируемость.

При рассмотрении задачи о деформировании грудной клетки человека при низкоскоростном ударе индентором отмечается хорошая масштабируемость вплоть до 128 ядер. Далее эффективность распараллеливания снижается из-за распределения индентора и области контакта с грудиной на несколько ядер, что приводит к увеличению межпроцессорных обменов и увеличению времени расчета.

Задача динамического взаимодействия индентора со скелетом грудной клетки человека масштабируется намного лучше, чем задачи ударного нагружения тканевых пакетов различных размеров и разным количеством слоев ткани. Это объясняется тем, что в тканевых пакетах возникают множественные контакты между нитями внутри одного слоя и между нитями из разных слоев ткани, в то время как контакт между скелетом грудной клетки человека и индентором

происходит по небольшой поверхности. Таким образом, задачи с множественными контактами масштабируются хуже.

В результате проведенных исследований по масштабируемости задач было получено, что декомпозицию объектов необходимо обеспечивать таким образом, чтобы контактные зоны приходились на минимальное количество ядер, назначать минимально возможное количество контактирующих объектов.

Полученные нами результаты и методы исследования сопротивления тканевых преград ударам огнестрельного оружия используются при разработке новых средств защиты тела человека, значительно сокращая этап предварительной оценки служебных свойств такого рода изделий.

Литература

1. Recht, R.F. Analytical modeling of plate penetration dynamics. / R.F. Recht // High velocity impact dynamics / Jonas A. Zucas., R.F. Recht. - New York, Wiley, 1990.- 443-515 pp.
2. ГОСТ Р50744-95. Бронеодежда. Классификация и общие технические требования.
3. NIJ Standard - 0101.06. Ballistic Resistance of Body Armor.
4. Jack C. Roberts, Paul J. Biermann, James V. O'Connor, Emily E. Ward, Russell P. Cain, Bliss G. Carkhuff, Andrew C. Merkle. Modeling nonpenetrating ballistic impact on a human torso // Johns hopkins apl technical digest, volume 26, number 1, 2005.
5. Григорян В.А., Маринин В.М., Хромушин В.А. Расчетная оценка противоосколочной стойкости тканевых защитных структур на основе характеристик энергоемкости // Тезисы докладов VIII Международной конференции "Новейшие тенденции в области конструирования и применения баллистических материалов и средств защиты" (15-16 сентября 2005, г. Хотьково). - Хотьково, 2005. - с.14-15.
6. Ivanov D.S., Baudry F., Van Den Broucke B., Lomov S.V., Xie H., Verpoest I. Failure analysis of triaxial braided composite // Composites Science and Technology. - 2008. Т. 68. № 1.
7. H. Broos, K. Herlaar. Explicit FE modeling of ballistic impact on textile armour systems// Finite element modelling of textiles and textile composites, St-Petersburg, 2007, CD edition
8. Долганина Н.Ю., Сапожников С.Б. Связь динамической прочности арамидных тканей с искривлением нитей в них. // Наука и технологии., Труды XXV Российской школы и XXXV Уральского семинара, посвященных 60-летию Победы. М.: 2005. - с.103-110.
9. Сапожников С.Б., Долганина Н.Ю., Сахаров С.А. Моделирование динамики взаимодействия ударника и многослойного тканевого пакета // Вопросы оборонной техники, сер. 15, Композиционные и неметаллические материалы в машиностроении.- М.: МНЦ "Информтехника". - 2005. - Вып.3(140)-4(141), с.38-41.
10. Sapozhnikov S.B., Forental M.V., Dolganina N.Yu. Improved methodology for ballistic limit and blunt trauma estimation for use with hybrid metal/textile body armor. // Finite element modelling of textiles and textile composites, St-Petersburg, 2007, CD edition
11. LS-DYNA Keyword user's manual. v.970. LSTC, 2003. - 1564p.
12. Разрушение. Том 7. Разрушение неметаллов и композитных материалов. Часть II. Органические материалы (стеклообразные полимеры, эластомеры, кость). Под редакцией Ю.Н. Работнова. Издательство «Мир», Москва 1976.
13. Высокопроизводительный вычислительный кластер «СКИФ Урал»: [http://supercomputer.susu.ru/computers/ckif_ural/].

Параллельный код для трехмерного моделирования процессов космической газодинамики*

М.А. Еремин, В.Н. Любимов

Разработан параллельный код для численного моделирования процессов космической газодинамики с учетом тепловых процессов. Представлены результаты трехмерного моделирования с высоким разрешением неупругих столкновений облаков межзвездного газа H I при различных наборах начальных параметров. Проведено детальное исследование масштабируемости программы, в результате которого установлено, что эффективность распараллеливания составляет 70–90 %, в зависимости от параметров задачи.

1. Введение

Моделирование процессов космической газодинамики всегда являлось чрезвычайно ресурсоемкой сферой применения численных расчетов. Процессы, протекающие в астрофизических системах являются трехмерными, и как правило, существенно нестационарными. Исследование подавляющего числа астрофизических задач может быть выполнено только численно в силу ограниченности аналитических методов. Отметим, что физические условия в астрофизических системах часто приводят к образованию ударных волн и сильной турбулизации межзвездной среды. Для адекватного моделирования астрофизических систем необходимо использовать трехмерные численные схемы и высокое пространственное разрешение.

Одной из классических задач космической газодинамики является задача о столкновении облаков H I в многофазной межзвездной среде [1]. Укажем на тот факт, что только при большом пространственном разрешении возможно наблюдать развитие гидродинамических и тепловых неустойчивостей, играющих решающую роль в разрушении холодных плотных облаков H I. С учетом с трехмерности задачи это приводит к высоким требованиям к производительности вычислительных систем и делает невозможным решение данной задачи без использования технологий параллельных вычислений.

Некоторые физические аспекты столкновений облаков в рамках достаточно грубых приближенных моделей впервые были рассмотрены в работах [2,3]. Изучение столкновений облаков с использованием численного моделирования было проведено целым рядом авторов (см., например, работы [4–6]). Наиболее полное исследование столкновений облаков в двумерных численных моделях приведено в работах [7,8], в том числе, и с учетом магнитного поля. Тем не менее, целый ряд вопросов остался без рассмотрения, особенно в трехмерном случае.

Целями нашей работы являются:

1. Разработка и реализация трехмерной параллельной TVD схемы для моделирования процессов космической газовой динамики.

2. Применение разработанного кода для исследования неупругих столкновений облаков H I в многофазной межзвездной среде при различных начальных условиях, включая нелобовой характер столкновений и неидентичность сталкивающихся облаков.

3. Исследование масштабируемости параллельного кода и выявление основных причин издержек распараллеливания.

*Работа поддержана грантами РФФИ N07-02-01204, N09-02-97021 и ФЦП “Научные и научно-педагогические кадры инновационной России” (2009НК-21(7)).

2. Постановка задачи

2.1. Основные уравнения

Процесс столкновения облаков атомарного водорода H I, помещенных в теплую межзвездную среду, может быть описан уравнениями ньютоновской газовой динамики в одножидкостном приближении. Предположим, что газ является идеальным и политропным с показателем адиабаты $\gamma = 5/3$. В силу того, что масштаб Джинса намного больше размеров облаков $\lambda_J \gg R_c$, эффектами самогравитации газа пренебрегаем. Следует отметить, что на стадии компрессии облака испытывают существенное сжатие, что приводит к увеличению плотности, так что масштаб Джинса становится сравнимым с характерным вертикальным размером облака. Тем не менее, в данной работе для упрощения модели самогравитация газа не рассматривается.

Система уравнений газовой динамики с учетом тепловых процессов и теплопроводности записывается в виде:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0\,, \tag{1}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u}) = -\nabla \mathrm{p}\,, \tag{2}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot ([E + \mathrm{p}]\mathbf{u}) = \nabla[k(T)\nabla T] - L(p,n)\,. \tag{3}$$

Здесь ρ, p, $\mathbf{u} = \{u, v, w\}$ – плотность, давление и вектор скорости газа соответственно, $E = \rho\left(e + \frac{\mathbf{u}^2}{2}\right)$ – объемная энергия, $k(T)$ – коэффициент теплопроводности, $L(p,n) = \Lambda(T)n^2 - \Gamma(T)n$ – функция тепловых потерь, $\Lambda(T)$ – функция объемного охлаждения, $\Gamma(T)$ – функция объемного нагрева. Для идеального газа удельная энергия определяется выражением: $e = \frac{p}{\rho\,(\gamma-1)}$. Связь между давлением и температурой задается уравнением состояния: $p = nk_BT$, где k_B — постоянная Больцмана.

Функция охлаждения в наших численных моделях принималась отвечающей стандартному химическому составу межзвездной среды в окрестности Солнца (см., например, [9–11]). Расчеты показывают, что функция нагрева слабо зависит от температуры, поэтому мы полагали ее постоянной $\Gamma = 1{,}6 \cdot 10^{-25}$ эрг/с.

Коэффициент теплопроводности учитывает как атомную, так и электронную теплопроводности, зависимость теплопроводности от температуры имеет следующий вид:

$$k(T) = \begin{cases} k_a T^{1/2}/(1+\sigma_a), & \text{при } T < 10^4\,K, \\ k_e T^{5/2}/(1+\sigma_e), & \text{при } T \geq 10^4\,K. \end{cases} \tag{4}$$

При высоких температурах возникает эффект насыщения теплового потока, что в формулах (4) учитывается через величины σ_a и σ_e, представляющие отношение классического теплового потока к насыщенному [15–17]:

$$\sigma_a = \frac{q_{classic}}{q_{sat}} = \frac{k_a T^{1/2}|\nabla T|}{5\rho(k_BT/m_0)^{3/2}}\,, \tag{5}$$

$$\sigma_e = \frac{q_{classic}}{q_{sat}} = \frac{k_e T^{5/2}|\nabla T|}{5\rho(k_BT/m_0)^{3/2}}\,, \tag{6}$$

где m_0 – масса протона, $k_a = 2{,}0 \cdot 10^{-3}$ эрг/(с см $\mathrm{K}^{3/2}$) и $k_e = 5{,}6 \cdot 10^{-7}$ эрг/(с см $\mathrm{K}^{7/2}$) – коэффициенты атомной и электронной теплопроводности соответственно.

Система уравнений газовой динамики (1)–(3) записывалась в безразмерной форме с использованием нескольких базисных величин. Перечислим наиболее важные размерные

параметры задачи: $L_0 = 1$ пк – характерный пространственный масштаб, $T_0 = 10^4$ K — температура, $n_0 = 0{,}1\,\text{г/см}^3$ – концентрация, $c_{s0} \simeq 12$ км/с — адиабатическая скорость звука, $t_0 \approx 10^5$ лет — характерное время задачи.

Важное значение имеют равновесные параметры использованной физической модели. Укажем числовые значения следующих параметров, отвечающих тепловому равновесию: равновесная температура теплой фазы $T_{eq1} = 8922$ K; равновесная температура холодной фазы $T_{eq2} = 62{,}5$ K; концентрации $n_0 = 0{,}1$ г/см^3 соответствует равновесное давление межзвездной среды $p_{eq}/k_B = 892{,}2$ K/см^3 .

2.2. Разработка и реализация кода

2.2.1. Численная схема

Для компьютерного моделирования неупругих столкновений облаков в межзвездной среде мы реализовали явную численную схему для системы (1)–(3) в декартовой системе координат. При построении численной схемы использовался метод расщепления по физическим процессам, согласно которому численное решение строится как решение уравнений, описывающих различные физические процессы.

Гидродинамические процессы, описываемые левой частью уравнений (1)–(3), представляют собой уравнения в частных производных гиперболического типа, для которых разработаны схемы неубывания полной вариации TVD [13], эффективно подавляющие нефизичные осцилляции численного решения на скачках. Реализованная пространственно-нерасщепленная TVD схема относится к типу MUSCL [12], имеет третий порядок аппроксимации по пространству в областях гладкого течения и первый на скачках. За счет применения алгоритмов пересчета типа Рунге-Кутта, реализованная численная схема обладает вторым порядком по времени.

Учет тепловых процессов, описываемых правой частью системы (1)–(3), сводится к коррекции тепловой энергии на каждом временном слое после расчета гидродинамических параметров течения. Изменение внутренней энергии за счет тепловых процессов описывается уравнением на внутреннюю энергию, которое решалось численно как с помощью метода подшагов, так и с помощью метода, предложенного в работе [7].

2.2.2. Распараллеливание

Распараллеливание численного кода производилось с использованием стандарта MPI. Расчетная область равномерно распределялась между процессорами с последующим обменом границами после каждого гидродинамического шага. Деление на подобласти производилось вдоль одной оси прямоугольной сетки – деление на “слои” (рисунок 1). Укажем еще два варианта деления – это деление в направлении двух координат — деление на “столбцы”, и деление вдоль трех координат — деление на “кубы”. Вариант деления на “слои” был выбран для уменьшения количества вызовов функций обмена сообщениями, которые, как известно, приводят к дополнительным издержкам, не зависящим от объема пересылаемых данных.

При таком разбиении на подобласти стало возможным производить обмен границами только в два этапа. Сначала проводится обмен данными на всех нечетных границах одновременно – между 1-м и 2-м, 3-м и 4-м, 5-м и 6-м и т. д. процессами. Затем обмениваются границами 2-й и 3-й, 4-й и 5-й, 6-й и 7-й и т. д. процессы. Это приводит к тому что время, затрачиваемое на обмен границами, в идеальном случае не будет зависеть от количества процессоров, что положительно скажется на масштабируемости программы. В реальности при увеличении количества процессоров издержки все же растут, что связано с увеличением нагрузки на соединительную шину кластера. Такой способ обмена границами указывает на целесообразность использования разбиения расчетной области на “слои”, так как при

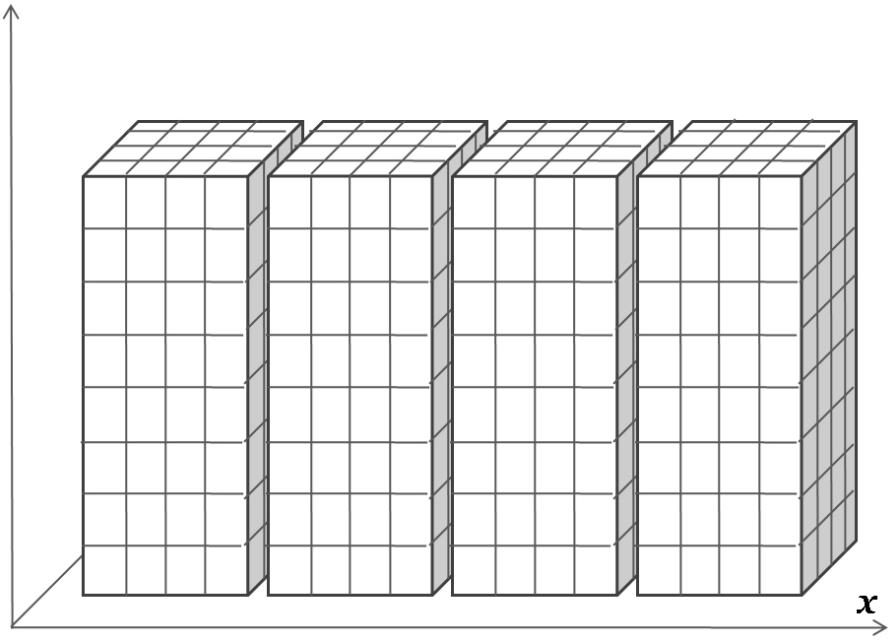


Рис. 1. Схема распределения расчетной области между процессорами

разбиении, скажем, на “кубы”, необходимо как минимум шесть этапов обмена границами (по количеству стыкующихся с соседними процессами граней у каждой подобласти).

Отметим также, что в коде проведена оптимизация распараллеливания – используется неблокирующая пересылка данных стандарта MPI [14], что делает возможным продолжение расчета в некоторой части подобласти во время обмена данными между процессами. Это особенно актуально при использовании кластеров с медленной соединительной сетью, как, например, у нетеряющей популярности недорогой вычислительной системы, состоящей из персональных компьютеров, объединенных локальной сетью.

Объем пересылаемых данных зависит от размера расчетной области и пропорционален N^2 (где N – количество узлов вдоль одной оси, N^3 – количество ячеек расчетной области). Таким образом, с ростом размера задачи время, затрачиваемое на обмен границами, растет гораздо медленнее времени расчета газодинамических процессов, которое пропорционально N^4, что приводит к повышению эффективности.

2.3. Начальные и граничные условия

Во всех численных моделях предполагалось, что в начальный момент времени температура и концентрация газа, отвечающих теплой фазе равны $T_w = 10^4$ К и $n_w = 0{,}1$ г/см^3, температура облаков H I $T_c = 80$ К, концентрация газа в них в 100 раз больше, чем в межоблачной среде $n_c = 10$ г/см^3. Кроме того, в начальный момент теплая и холодная фазы межзвездной среды находятся в состоянии механического равновесия, то есть $p_w = p_c$. Облака начинают движение навстречу друг другу с относительным числом Маха $M = u_c/c_w$, где c_w — адиабатическая скорость звука в межоблачной среде, u_c — скорость движения облака.

Радиус сталкивающихся облаков варьировался от 0,5 пк до 1,5 пк. Нами были рассмотрены различные варианты столкновений: лобовые столкновения идентичных облаков, лобовые столкновения неидентичных облаков и нелобовые столкновения.

3. Результаты

3.1. Моделирование

Расчеты проводились с разрешением до 800^3 ячеек. Основной моделью является модель трехмерного лобового столкновения облаков H I.На рисунке 2 показаны основные стадии эволюции: a) стадия компрессии или сжатия, когда ударные волны, образовавшиеся при столкновении, распространяются наружу; b) стадия перерасширения, на которой волны разрежения, распространяясь назад, формируют центральную область низкого давления и плотности, на этой стадии образуется выброс, подверженный действию неустойчивостей типа Кельвина-Гельмгольца; c) стадия коллапса, когда расширение останавливается давлением внешней окружающей среды. На этой стадии развивается неустойчивость Релея-Тейлора; d) стадия разрушения облаков.

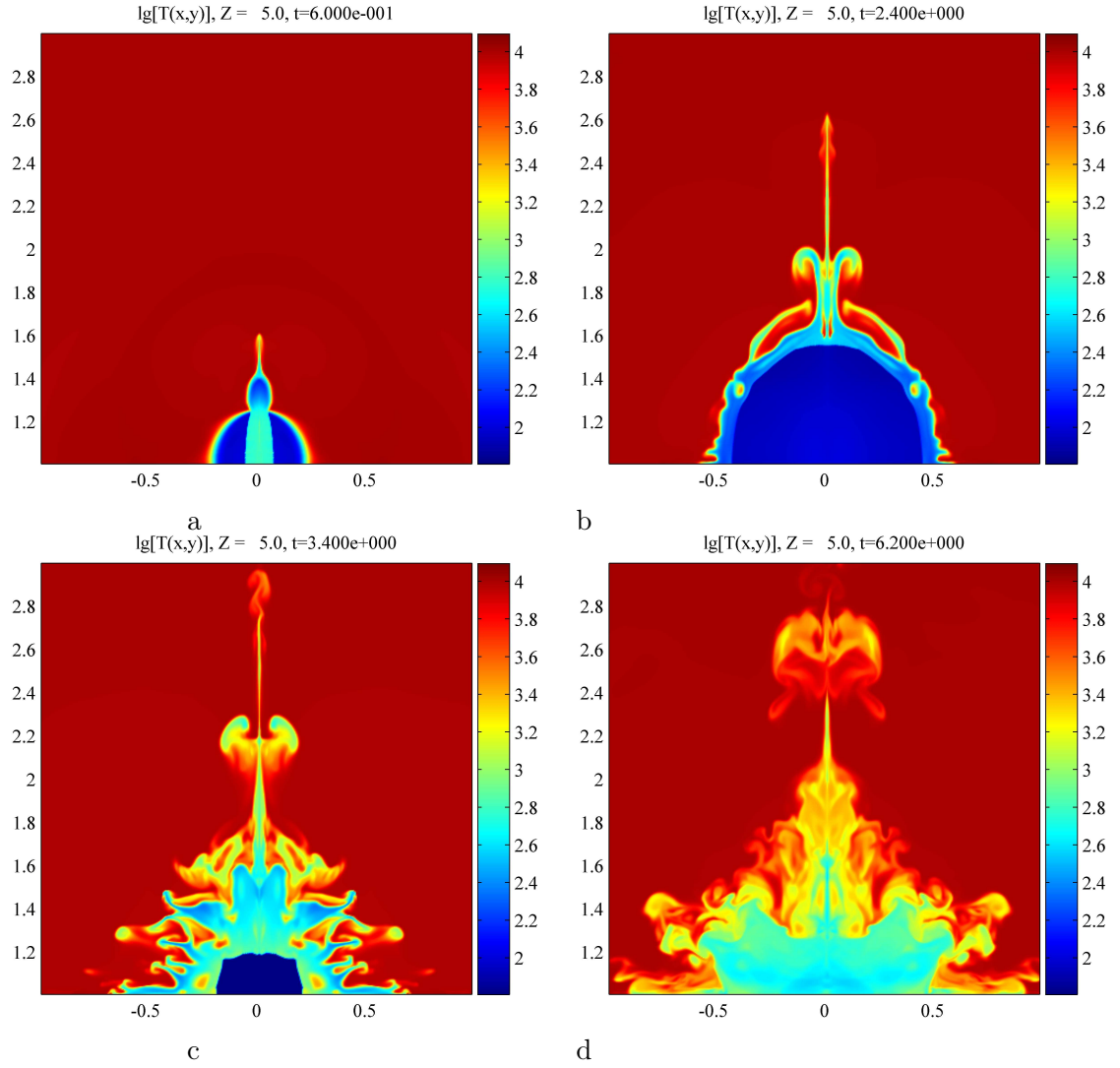


Рис. 2. Распределение логарифма температуры ($\lg T/T_0$, $T_0 = 1\,K$) для трехмерной модели лобового столкновения идентичных облаков (сечение плоскостью $Z = 0$)

Следует отметить, что для адекватного моделирования лобовых столкновений идентичных облаков необходимо использовать модели с высоким пространственным разрешением (не менее 50 ячеек на радиус облака). Только в этом случае возможно образование филаментных структур, приводящих к фрагментации облаков. В противном случае происходит сильная деградация численного решения, вследствие чего в результате слияния двух стал-

кивающихся облаков возможно формирование нового облака, что как раз и наблюдалось в работе [7].

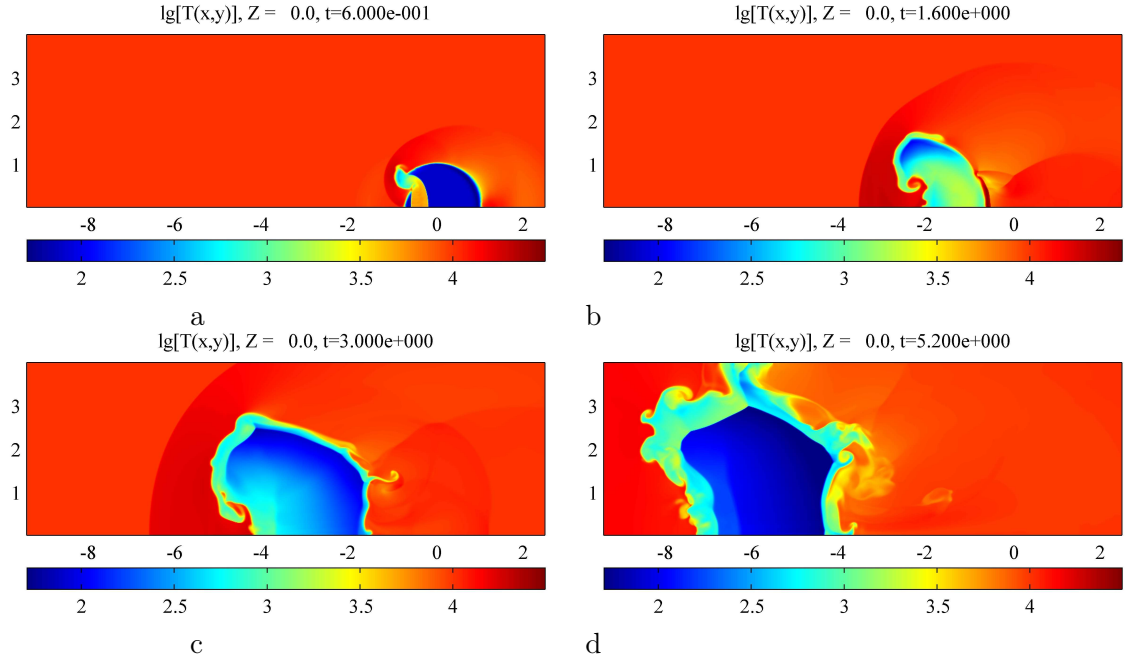


Рис. 3. Распределение логарифма температуры ($\lg T/T_0$, $T_0 = 1\,K$) для трехмерной модели лобового столкновения неидентичных облаков (сечение плоскостью $Z = 0$)

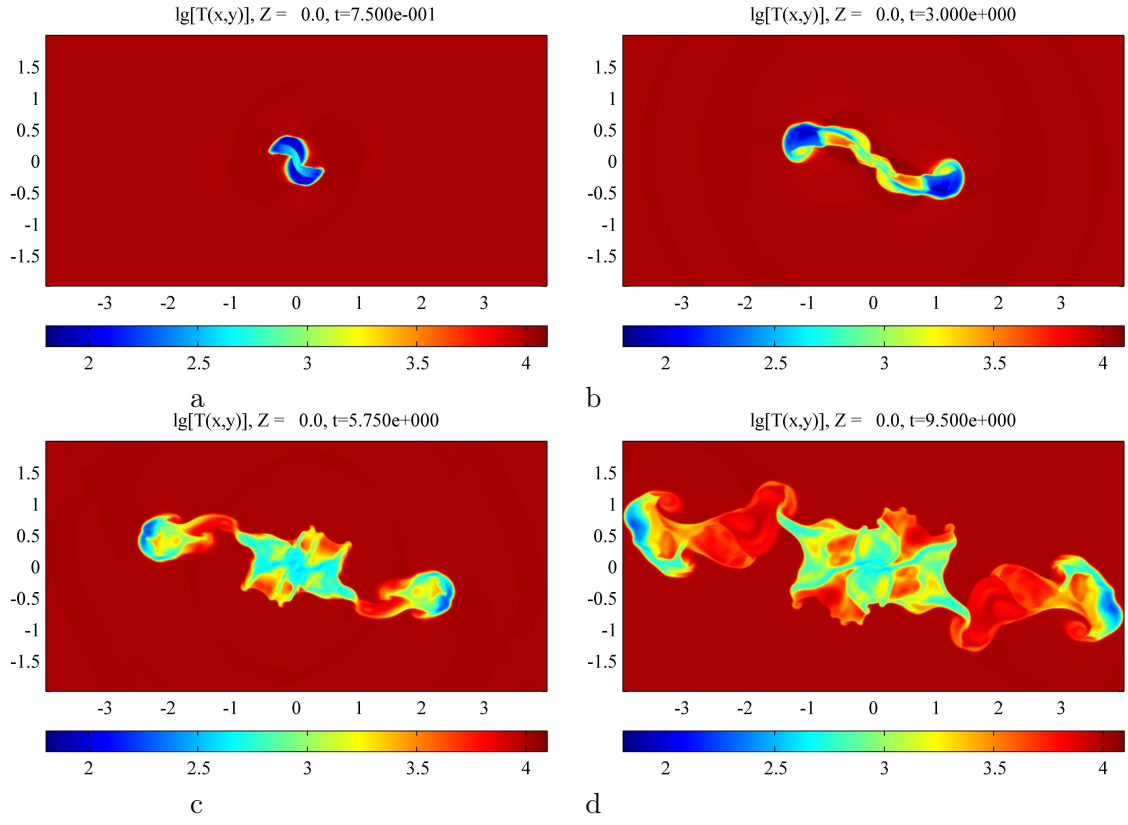


Рис. 4. Распределение логарифма температуры ($\lg T/T_0$, $T_0 = 1\,K$) для трехмерной модели нелобового столкновения идентичных облаков (сечение плоскостью $Z = 0$)

На рисунке 3 показаны несколько стадий лобового столкновения двух неидентичных облаков с радиусами $R_{c1} = 0{,}5$ пк и $R_{c2} = 1{,}0$ пк. В результате столкновения облаков происходит интенсивное развитие гидродинамических неустойчивостей, турбулизирующих среду. На основании проведенных нами численных экспериментов можно заключить, что столкновения неидентичных облаков приводят к их полному разрушению и переводу газа из холодной фазы в теплую.

Также интерес представляет трехмерная модель нелобового столкновения облаков (рисунок 4). Идентичные облака с радиусом 0,5 пк двигаются навстречу друг другу ($M_1 = M_2 = 1{,}5$), таким образом, что центры облаков смещены на расстояние, равное одному радиусу. В результате столкновения образуются макроскопические вихревые филаменты с характерным размером несколько парсек.

3.2. Масштабируемость

Все расчеты и тесты проводились на сорока ядерном кластере, состоящем из пяти узлов, объединенных высокоскоростной шиной InfiniBand. Каждый узел содержит по два процессора Intel Xeon и 8 ГБ оперативной памяти.

Эффективность программы определялась как отношение ускорения программы S при использовании p процессоров к числу процессоров:

$$E = \frac{S_p}{p}\,. \tag{7}$$

Ускорение программы вычисляется как отношение времени выполнения параллельной программы T_p ко времени выполнения последовательной программы T_1:

$$S_p = \frac{T_p}{T_1}\,. \tag{8}$$

Для измерения времени, затрачиваемого на обмен границами в тестах использовалась версия программы с блокирующей пересылкой данных. Результаты тестов представлены в таблице 1.

Таблица 1. Зависимость эффективности программы от количества процессоров

Количество процессоров	**Время выполнения (сек)**	**Время обмена границами (сек)**	**Доля времени обмена границами (%)**	**Эффективность программы (%)**
1	12811,5	0	0	100
4	3285	8,4	0,26	97,5
20	733	21,1	2,88	87,39
40	470	45,85	9,76	68,15

Анализ данных, приведенных в таблице 1 показывает, что время, затрачиваемое на пересылку данных, значительно растет с увеличением количества используемых процессоров, чего в идеале не должно наблюдаться исходя из реализованного алгоритма обмена границами (см. пункт 2.2.2). Очевидно, это вызвано косвенными издержками, возникающими при параллельных расчетах. Все же даже такой рост времени обмена границами не объясняет чрезвычайно сильное падение эффективности программы с увеличением количества используемых процессоров. Отметим, что на сорока ядрах эффективность составляет

менее 70%, издержки же на обмен данными не превышают 10%. Очевидно, существуют другие механизмы, не связанные с обменом границами, или возрастанием нагрузки на шину InfiniBand, но также влияющие на скорость счета программы, и возрастающие с увеличением количества используемых процессоров. Одной из таких причин может являться нехватка скорости внутренней шины данных для обеспечения одновременного доступа к оперативной памяти сразу всеми восьми процессорами вычислительного узла. В этом случае скорость счета программы будет зависеть от количества используемых на одном узле ядер. Для проверки данного предположения были проведены тесты скорости выполнения программы при одинаковом количестве процессов, но при различном их распределении по узлам кластера (см. таблицу 2).

Таблица 2. Зависимость времени счета от загруженности узлов

Количество используемых узлов	**Количество используемых ядер на каждом узле (из 8)**	**Время выполнения (сек)**
1	8	4324
2	4	2538
4	2	2620

Во всех тестах количество используемых ядер равно восьми, но они располагаются на одном, двух или четырех узлах. Таким образом, сохраняя суммарную производительность вычислительной системы, мы можем оценить влияние архитектуры кластера. Как видно, время счета меняется до двух раз, причем максимальное время расчета достигается при использовании всех ядер одного узла, что подтверждает наше предположение о причинах замедления выполнения программы.

Для исключения влияния архитектуры конкретной вычислительной системы на эффективность кода были произведены тесты производительности при одинаковом заполнении узлов кластера, результаты которых представлены в таблице 3.

Таблица 3. Эффективность программы при полной загруженности используемых вычислительных узлов

Количество процессоров	**Время выполнения (сек)**	**Время обмена границами (сек)**	**Доля времени обмена границами (%)**	**Эффективность программы (%)**
1	32082	0	0	100
8 (1 узел)	4359	37,72	0,87	92
16 (2 узла)	2095	112,21	5,36	95,71
24 (3 узла)	1447	94,15	6,51	92,38
32 (4 узла)	1162	123,27	10,61	86,27
40 (5 узлов)	921	93,67	10,17	87,09

Следует обратить внимание на то, что абсолютное время обмена данными практически не меняется, что говорит о эффективности алгоритма параллельного обмена границами в

два этапа (см. пункт 2.2.2). Также следует заметить, что небольшое падение эффективности (порядка 10%), полностью соответствует затратам времени на пересылку данных. Это означает, что в данном тесте нам действительно удалось исключить влияние архитектуры кластера, и мы можем наблюдать реальную эффективность реализации параллельного алгоритма. Наглядное сравнение результатов двух тестов приведено в виде графиков на рисунке 5.

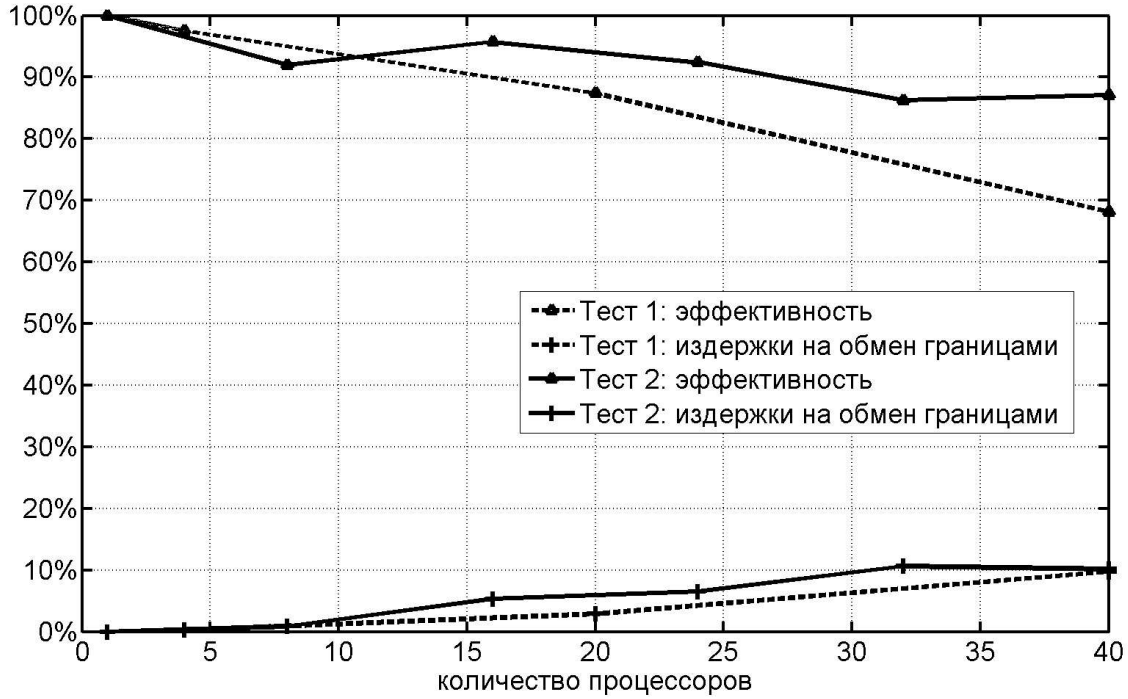


Рис. 5. Масштабируемость программы: Тест 1 – с использованием всех узлов кластера (таблица 1) Тест 2 – при полном использовании части вычислительных узлов (таблица 3)

Следует упомянуть, что для расчетов (в отличие от тестов) используется версия программы с неблокирующей пересылкой данных, что фактически позволяет исключить издержки на обмен границами, и приводит к более высокой масштабируемости разработанного кода. К сожалению, технические возможности не позволили нам провести тесты с более чем сорока ядрами.

4. Основные выводы

В заключение нашей работы сформулируем основные выводы:

1. Разработана и реализована трехмерная параллельная схема для численного моделирования процессов космической газодинамики.

2. Разработанный код обладает хорошей масштабируемостью и высокой эффективностью (порядка 90%).

3. Моделирование столкновений облаков H I с использованием полученного кода показывает, что вне зависимости от характера столкновения происходит разрушение облаков с образованием филаментных структур.

Литература

1. McKee C.F., Ostriker J.P. A theory of the interstellar medium - Three components regulated by supernova explosions in an inhomogeneous substrate. // The Astrophysical Journal, 1977, 218, pp. 148-169.

2. Stone M.E. Collisions Between HI Clouds. I. One-Dimensional Model. // The Astrophysical Journal, 1970, 159, p. 277.

3. Stone M.E. Collisions Between HI Clouds. II. Two-Dimensional Model. // The Astrophysical Journal, 1970, 159, p. 293.

4. Hausman M.A, Collisional mergers and fragmentation of interstellar clouds. // The Astrophysical Journal, 1981, 245, pp. 72-91.

5. Gilden D.L. Clump collisions in molecular clouds - Gravitational instability and coalescence. // The Astrophysical Journal, 1984, 279, pp. 335-349.

6. Lattanzio J.S, Monaghan J.J., Pongracic H., Scywartz M.P. Interstellar Cloud Collisions. // Monthly Notes of the Royal Astronomical Society, 1985. – V. 215, P. 125.

7. Miniati F., Jones T.W., Ferrara A., Ryu D. Hydrodynamics of Cloud Collisions in Two Dimensions: The Fate of Clouds in a Multiphase Medium. // The Astrophysical Journal, 1997, 491, p. 216.

8. Miniati F., Ryu D., Ferrara A., Jones T.W. Magnetohydrodynamics of Cloud Collisions in a Multiphase Interstellar Medium. // The Astrophysical Journal, 1999, 510, pp. 726-746.

9. Spitzer L. Physics of fully ionized gases. // New York: Interscience, 1962.

10. Марочник Л. С., Сучков А. А. Галактика. // М.: Наука, 1984.

11. Bochkarev N. G. Основы физики межзвездной среды. // М: МГУ, 1991.

12. van Leer B. Towards the ultimate conservative difference scheme. V. A second order sequel to Godunov's methods. // Journal of Computational Physics, 1979, 32, №1, pp. 101-136.

13. Harten A. High Resolution Schemes for Hyperbolic Conservation Laws. // Journal of Computational Physics, 1983, 49, №3, p. 357.

14. Антонов А.С., Параллельное программирование с использованием технологии MPI. // -М.: МГУ, 2004, 71с.

15. Cowie L. L., McKee C. F. The evaporation of spherical clouds in a hot gas. I - Classical and saturated mass loss rates. // The Astrophysical Journal, 1977, 211, pp. 135-146.

16. Balbus S. A., McKee C. F. The evaporation of spherical clouds in a hot gas. III - Suprathermal evaporation. // The Astrophysical Journal, 1982, 252, pp. 529-552.

17. Giuliani J. On the dynamics in evaporating cloud envelopes. // The Astrophysical Journal, 1984, 277, pp. 605-614.

18. Ricotti M., Ferrara A., Miniati F. Energy Dissipation in Interstellar Cloud Collisions. // The Astrophysical Journal, 1997, 485, p. 254.

19. Gregory R. Andrews, Foundations of Multithreaded, Parallel, and Distributed Programming. // Addison-Wesley, 2000

Исследование масштабируемости задач вычислительной гидроаэродинамики на различных многоядерных и многопроцессорных архитектурах[1]

В.А. Васильев, А.Ю. Ницкий

Проведено численное исследование влияния архитектуры кластера на эффективность решения задач вычислительной гидроаэродинамики (ВГАД) на примере задач течения турбулентной жидкости в тонком слое уплотнительных узлов гидромашин с использованием пакета ANSYS CFX. Получена существенная зависимость производительности от архитектуры и способа загрузки вычислительных узлов. Проведена оптимизация решения задач ВГАД на кластере “СКИФ Урал” ЮУрГУ.

Задачи вычислительной гидроаэродинамики требуют значительных вычислительных ресурсов и мощных программных средств, способных использовать возможности, предоставляемые современными вычислительными системами. При решении больших и сверхбольших задач важным фактором является оптимизация совместной работы вычислительной системы (кластера) и программного продукта. Цель данной работы – исследование эффективности суперкомпьютера СКИФ Урал ЮУрГУ, на базе четырехъядерных процессоров E5472 и определение необходимых и достаточных условий оптимизации решения совместных задач гидродинамики и динамики ротора на вычислительном кластере ЮУрГУ.

Для проведения вычислительных экспериментов в качестве тестовой задачи использованы задачи расчета гидродинамики тонкого турбулентного слоя жидкости в щелевом уплотнении мощного питательного насоса. Тесты ориентированы на использование известного параллельного пакета, широко используемого на различных суперкомпьютерных платформах – пакета инженерного анализа ANSYS® CFX v. 11.0. В отличие от использования тестов, таких как LINPACK, основанных на решении системы линейных уравнений прямыми методами, гидродинамическая задача большой размерности, как правило, решается итерационными методами и представляет собой сложную математическую модель реальных объектов и конструкций.

Расчеты проводились на кластере СКИФ Урал ЮУрГУ, имеющим LINPACK-производительность 12,2 TFlops. Технические характеристики приведены в таблице 1.

Таблица 1. Технические характеристики вычислительного кластера СКИФ Урал ЮУрГУ

Количество узлов / процессоров / zlth	166 / 332 / 1328
Процессоры вычислительного узла	2 x Intel Xeon E5472 (4 ядра по 3.0 GHz)
ОЗУ/ дисковая память вычислительного узла	8GB DDR3 / 120 GB
Общая дисковая память / система хранения	49,39 TB / Panasas ActiveStorage 5100
Тип системной сети	InfiniBand (20Gbit/s, макс. задержка 2 мкс)
Тип управляющей (вспомогательной) сети	Gigabit Ethernet
Сервисная сеть	СКИФ ServNet
Пиковая производительность/Linpack	16 TFlops / 12,2 TFlops
Операционная система	SUSE Linux Enterprise Server 10 , Windows CCS
Тестируемое програмное обеспечение	ANSYS® CFX Academic Research, v. 11.0, SP1
Библиотека MPI (ОС Linux)	HP MPI v.02.02.05.01
Система бесперебойного электропитания	APC Symmetra 160 kVA

[1] Авторы статьи выражают благодарность официальному представителю ANSYS компании "Делкам-Урал" за предоставленные на время проведения тестирования HPC лицензии CFX v11.0 SP1

Также для расчетов использовалась рабочая станция Supermicro, представляющая собой 2-х процессорную ЭВМ с объемом ОЗУ 32 Гбайт на базе процессоров AMD Opteron. Технические характеристики ЭВМ приведены в таблице 2. Кроме того, ряд тестов на малой задаче был проведен на рабочей станции Supermicro с двумя четырехъядерными процессорами Intel Xeon E5520@2,27 ГГц и 24 ГБайт DDR3 ОЗУ.

Таблица 2. Технические характеристики рабочей станции

Количество процессоров	2
Количество вычислительных ядер	4
Тип процессора	2 x AMD Opteron 2216 (Santa Rosa, 2 ядра по 2.4 GHz) 2 x AMD Opteron 2354 (Barcelona, 4 ядра по 2.2 GHz) 2 x AMD Opteron 2427 (Istanbul, 6 ядер по 2.2 GHz)
ОЗУ	32GB DDR2
Дисковая память вычислительного узла	144 GB
Операционная система	Windows XP Pro x64 SP2, SUSE LES 10
Тестируемое програмное обеспечение	ANSYS® CFX Academic Research, v. 11.0, SP1
Библиотека MPI	MPICH2

С целью оценки влияния размера задачи на эффективность использования вычислительной мощности кластера анализ проводился на трех задачах разного размера. **Малая задача** – 46 218 узлов (29 204 элементов), **средняя задача** – 328 392 узлов (242 663 элементов) и, наконец, **большая задача** - 9 903 873 узлов (9 871 040 элементов).

Малая задача представляет собой расчет гидродинамики тонкого турбулентного слоя в радиальной щели стендовой установки в двумерной осесимметричной постановке. Проводилось исследование течения жидкости в радиальном зазоре, образуемым диском, закрепленным на свободном конце модельного ротора стендовой установки (рис. 2). Диск диаметром 130 мм образует с корпусом радиальную щель зазором от 200 до 500 мкм. Внутренний радиус щели 66мм, внешний 93мм, длина радиального зазора 27 мм. На вход радиальной щели подавалась жидкость – вода с температурой порядка 40˚С. На рис. 3. представлена двумерная осесимметричная модель радиальной щели в ANSYS® CFX-Pre и фрагмент расчетной сетки.

Средняя задача представляет собой расчет гидродинамики уравновешивающего устройства питательного насоса СВПТ 1150, проводимый для определения упругих и демпфирующих свойств уплотнений проточной части, в частности, уплотнений разгрузочного устройства. Расчеты полей давления, полей скоростей в щелевом уплотнения уравновешивающего устройства насоса СВПТ 350-850 проводились для цилиндрической щели диаметром 180 мм, длиной 185мм, радиальной щели, наружный диаметр которой составлял 300мм и внутренний 250 мм. Зазор цилиндрической и радиальной щелей принят соответственно 350 и 120 мкм. Перекачиваемая среда - питательная вода, температура 170˚С, давление на входе в цилиндрическую щель 34,4 МПа, давление на выходе из радиальной щели 2,15 МПа.

Задача решалась в осесимметричной постановке (рис. 4) с заданием граничных условий по давлению на входе и выходе. Сетка строилась на одноградусном секторе, размер сетки составил 328392 узлов, 242663 элементов.

Большая задача представляет собой расчет гидродинамики тонкого турбулентного слоя в радиальной перекошенной щели стендовой установки. В отличие от первого варианта расчета (малая задача), в котором зазор в щели параллелен, решение данной задачи возможно лишь в трехмерной постановке. Это позволяет получить распределения давления по длине щели, присущие конфузорной и диффузорной геометрии зазора (рис.5.).

Для перекошенной щели сетка построена из двух расчетных подобластей (Domain), которые затем стыкуются с помощью интерфейсов (Domain Interfaces), как показано на рис 6. Общее число узлов на перекошенной радиальной щели составило 9 903 873. Необходимая оперативная память 31,6 Гбайта, dремя счета на 16 ядрах кластера Скиф Урал составляет порядка 4 часов.

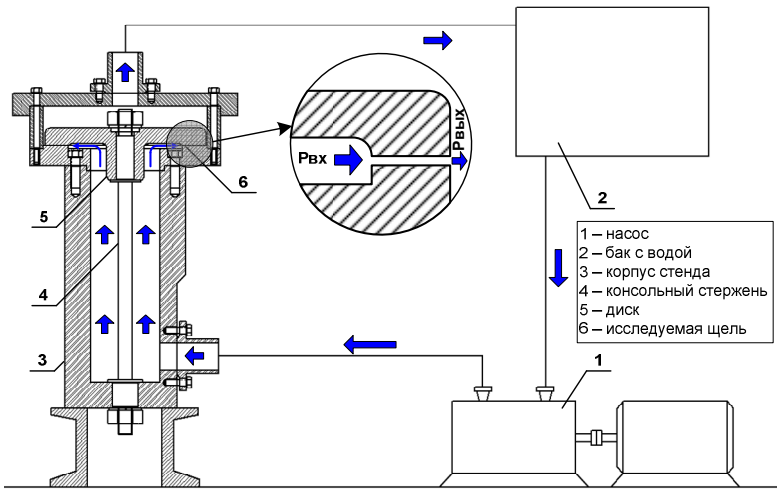


Рис. 2. Принципиальная схема стендовой установки и модельного ротора

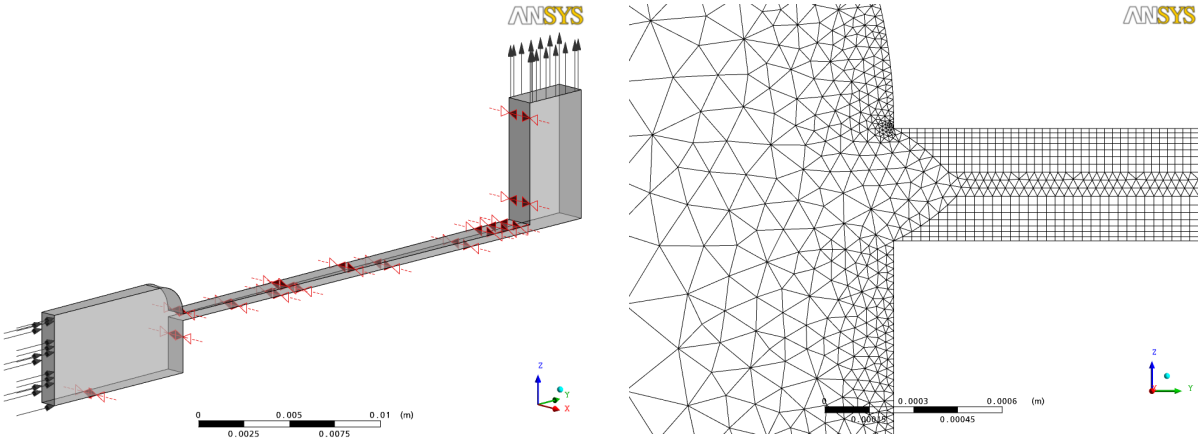


Рис.3. Модель малой задачи в ANSYS CFX-Pre и расчетная сетка

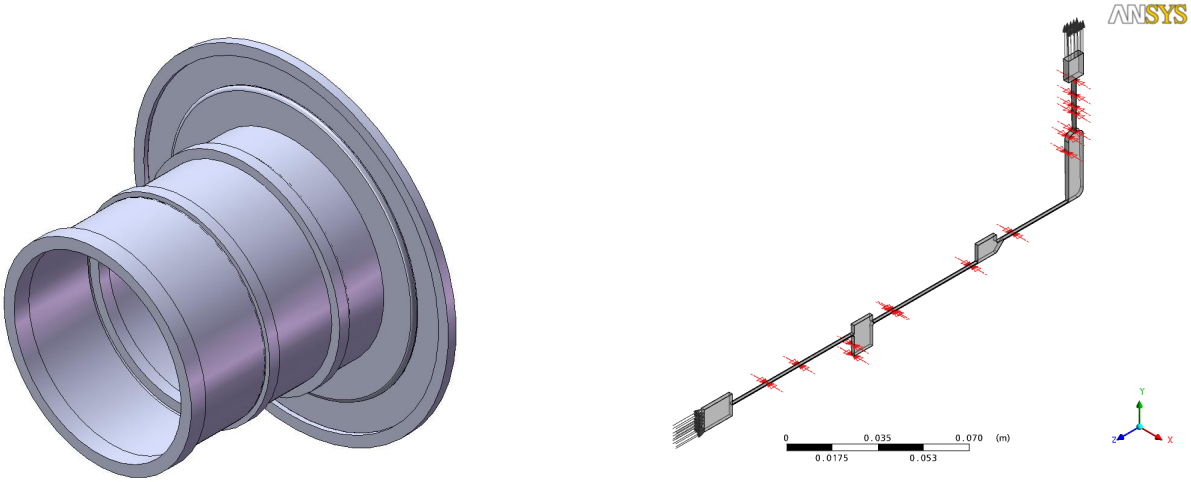


Рис.4. Геометрия щели уравновешивающего устройства питательного насоса СВПТ 350-850

В большинстве вариантах расчетов использовалась версия решателя CFX с двойной точностью чисел с плавающей точкой (double), кроме того, проведен ряд вычислительных экспериментов на решателе с одинарной точностью на тех задачах, где одинарная точность давала устойчивую сходимость решения.

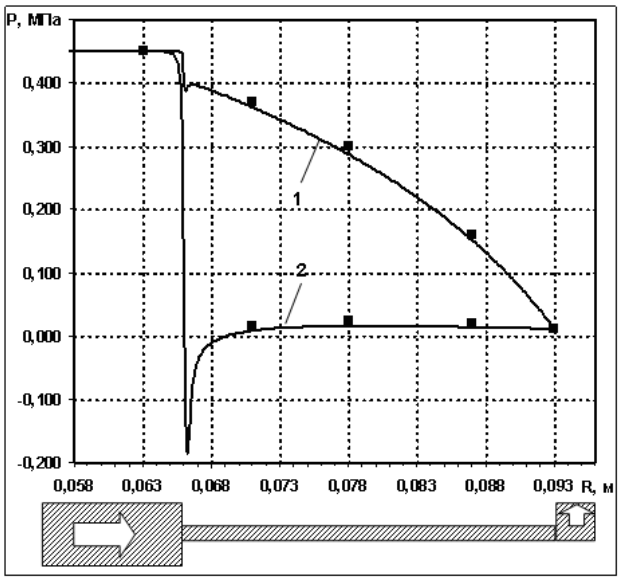


Рис. 5. Распределение давления по длине щели в сечениях конфузор (1), диффузор (2)

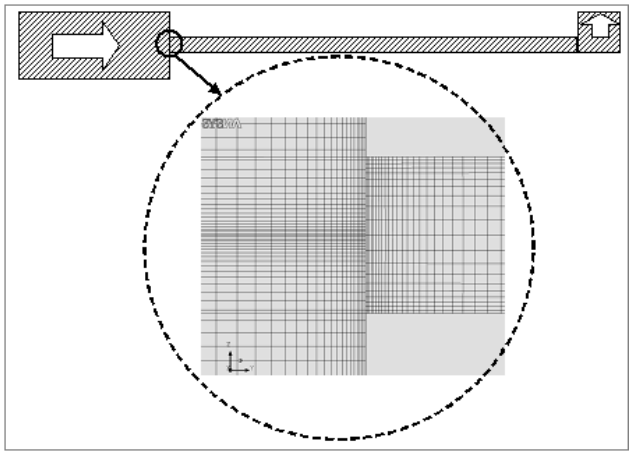


Рис. 6. Участок расчетной сетки на стыке 2х доменов: входной зоны и щелевого уплотнения

Результаты сравнительного анализа эффективности вычислений

Эффективность расчета, в значительной степени, зависит от процесса генерации сетки и оптимизации процесса распараллеливания сетки для последующего использования мощных кластерных систем. Эффективность разделения сетки при использовании пакета ANSYS CFX зависит от: метода разделения и числа блоков сетки при разделении.

В свою очередь, эффективность работы в параллельном режиме зависит также от размера задачи или числа узлов сетки, типа элемента, используемого при генерации сетки. Например, для тетраэдрических сеток рекомендуется не использовать меньше 30 000 узлов на блок сетки (на вычислительное ядро), для гексаэдрических сеток минимальное количество узлов на блок рекомендовано не менее 75 000 узлов [1]. Количество узлов на процессор не является жестким; при решении реальной задачи на определенной аппаратной платформе критическое количество узлов может быть больше или меньше.

Кроме того, узел кластера, содержащий два процессора и несколько ядер на каждом, может иметь ограничение по масштабированию из-за недостаточной пропускной способности шины памяти. По существу два центральных процессора могут потребовать больше ресурсов доступа к памяти, чем может обеспечить шина памяти.

Масштабируемость задачи на кластере в зависимости от числа используемых вычислительных ядер

Вычисления проводились на большой задаче с числом узлов равным 9 903 873. Результаты приведены на рис. 7 и в табл. 3. Эффективность вычислений оценивалась по числу итераций в час. Получены две кривые: первая с насыщением на уровне 200 ядер, вторая кривая показывает рост производительности до 300 ядер. Отличие в эффективности использования кластера зависит от количества используемых ядер вычислительных узлов. Первая кривая соответствует использованию полной вычислительной мощности каждого узла с загрузкой всех восьми ядер. Вторая кривая соответствует специальной загрузке, при которой на каждом отдельном узле загружается только 4 ядра. Более эффективным является второй вариант загрузки кластера. Это свидетельствует о вероятных проблемах, связанных с совместной работой процессоров на узле кластера. В ходе проведения вычислительных экспериментов было выявлено, что при использовании от 1го до 4х (включительно) ядер на узел производительность вычислений зависит от общего числа используемых ядер и не зависит от количества используемых ядер на вычислительном узле. На рис 7. и далее зеленой линией показано линейное ускорение.

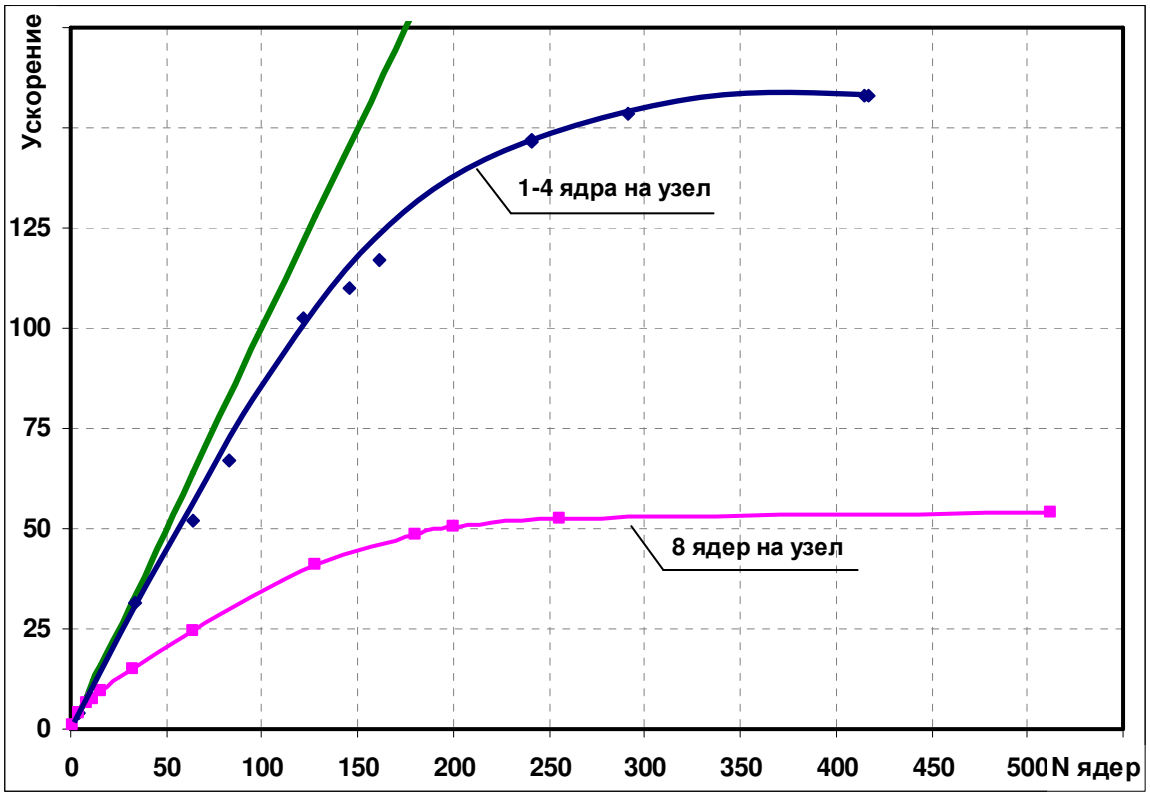


Рис. 7. Масштабирование большой задачи в зависимости от использования числа ядер внутри узла

Таблица 3. Масштабирование большой задачи

Кол-во ядер	Способ запуска	Время 100 итер.	Итер/час	Ускор.	Эфф-ть
417	104*4+1	0:13:08	457,0	157,8	37,8%
415	138*3+1	0:13:05	458,0	158,2	38,1%
291	145*2+1	0:13:31	444,4	153,5	52,7%
241	80*3+1	0:14:09	425,5	147,0	61,0%
241	120*2+1	0:14:08	424,6	146,6	60,8%
161	80*2+1	0:25:02	339,0	117,1	72,7%
146	146	0:18:47	319,1	110,2	75,5%
121	121	0:20:12	297,0	102,6	84,8%
83	83	0:31:03	193,5	66,8	80,5%
64	64	0:40:13	150,0	51,8	80,9%
33	33	1:05:33	91,6	31,6	95,9%
4	4	8:37:56	11,6	4,0	100,0%

Масштабируемость задач внутри вычислительного узла

Вычисления проводились на всех типах задач, производительность вычислений оценивалась по числу итераций в минуту. Результаты для малой задачи приведены на рис. 8. (узел кластера СКИФ Урал) и рис. 9. (рабочая станция AMD с процессорами Opteron 2354 и 2427, рабочая станция с двумя процессорами Intel E5520). При тестировании на узле СКИФ Урал получены две кривые: первая с насыщением на уровне 4х ядер при запуске задачи внутри узла, вторая кривая показывает больший рост производительности, и соответствует специальной загрузке, при которой к первому ядру одного узла последовательно подключаются по одному вычислительному ядру на каждом следующем узле, подключенному через сеть интерконнекта. Данный способ загрузки позволяет оценить "параллельный" потенциал задачи без ограничения архитектуры узла. Как оказалось, узел кластера, содержащий два процессора и восемь вычисли-

тельных ядер, имеет ограничение по масштабируемости, вероятнее всего, из-за недостаточной пропускной способности шины памяти. Подобное поведение наблюдается и на других пакетах ВГАД, использующих отличные от CFX алгоритмы решателей [2,3,4] и тестах производительности, основанных на решении систем линейных уравнений итерационными методами [5].

Аналогичная картина наблюдается и на рабочих станциях (рис. 9), однако общий уровень масштабируемости этих систем на малой задаче выше, чем предыдущей. Следует отметить, что новое поколение процессоров AMD Opteron Istanbul существенно не улучшило масштабируемость по сравнению с Barcelona, а линейка процессоров Intel Nehalem улучшила свои показатели по сравнению с предыдущей Harpertown и сравнялась по масштабируемости с системами AMD. Однако уровень абсолютной производительности систем оказался разный, и процессор E5520 значительно обгоняет конкурентов теста (рис.10), несмотря на значительное отставание по частоте от Xeon X5472 и по количеству ядер от Opteron 2427.

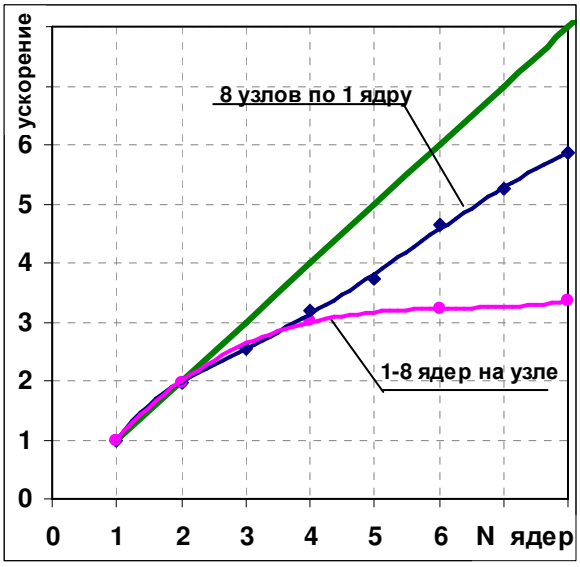


Рис. 8. Масштабируемость малой задачи на узле кластера СКИФ Урал при использовании различных способов запуска задачи

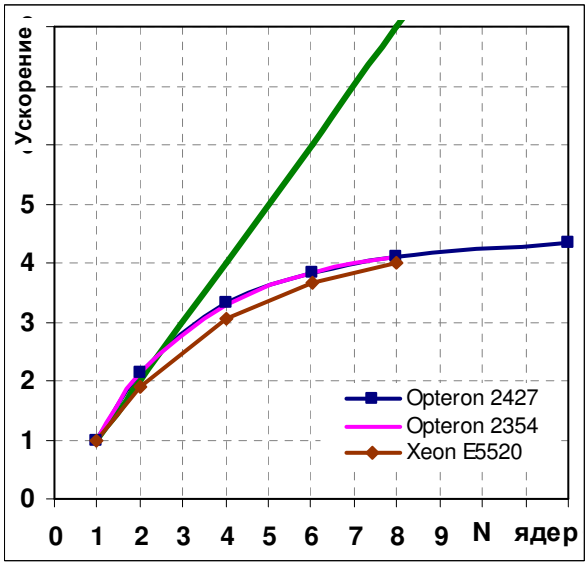


Рис. 9. Масштабируемость малой задачи 2x AMD Opteron 2354 , 2x AMD Opteron 2427, 2 x Intel Xeon E5520

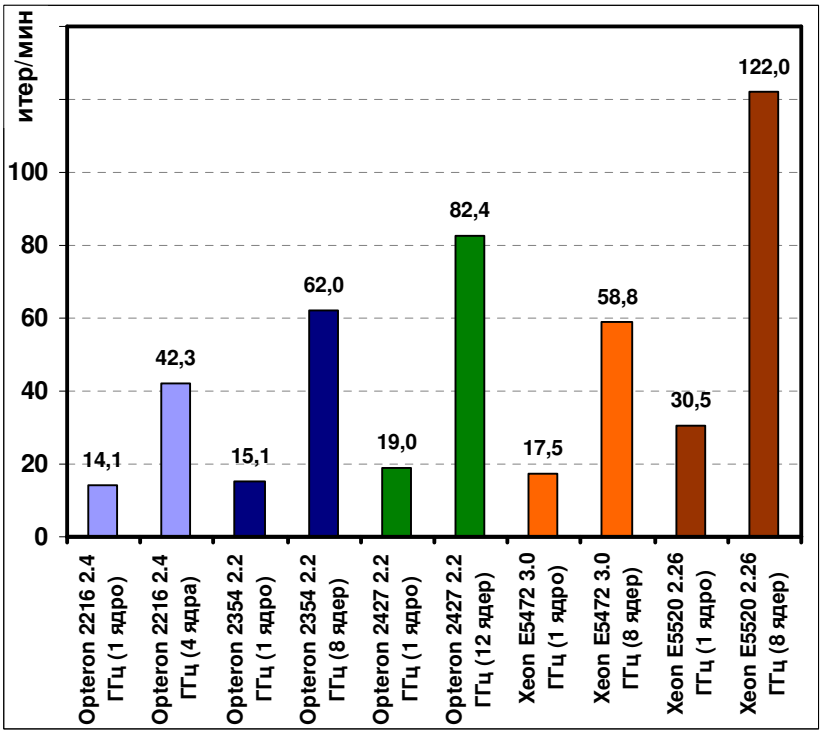


Рис. 10. Абсолютная производительность различных систем при решении малой задачи

Результаты тестирования средней и большой задачи на рабочей станции AMD Opteron 2427 приведены на рис. 10, в тестах были использованы версии решателя с одинарной и двойной точностью. Очевидно, на большой задаче с использованием решателя с одинарной точностью не достигается ограничение пропускной способности шины памяти, кроме того, масштабируемость решателя с двойной точность превышает масштабируемость на малой и средней задаче (рис. 9, 10). Средняя задача показывает наименьшую масштабируемость среди трех тестовых задач как для решателя с одинарной, так и с двойной точностью. Вероятно, это обуславливается наличием нескольких областей с высоким градиентом скоростей и давления, и, как следствие, существенным различием по сходимости и времени расчета между подобластями декомпозиции расчетной области.

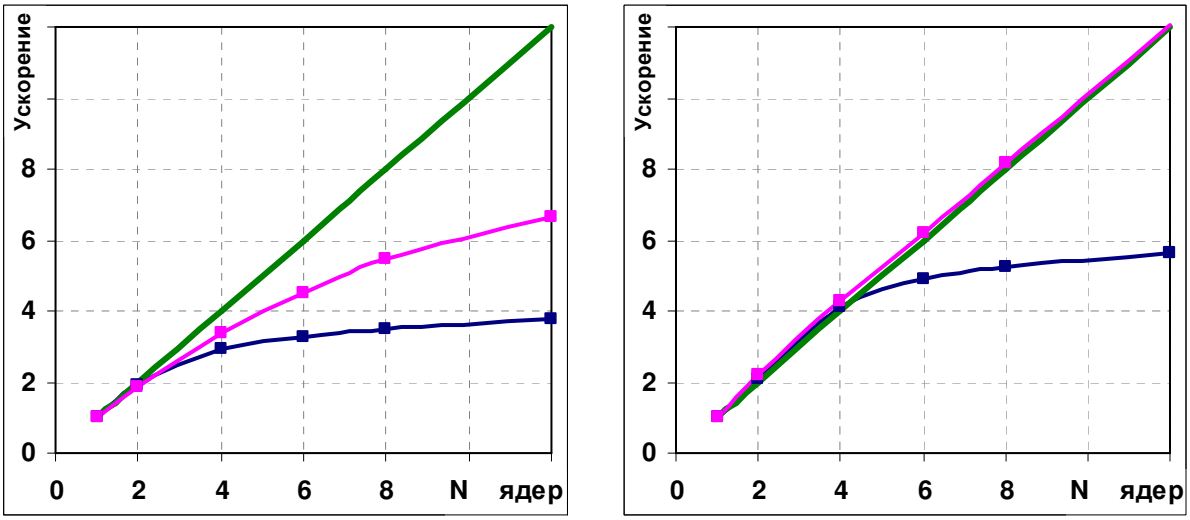


Рис. 10. Масштабируемость средней (слева) и большой (справа) задачи
2x AMD Opteron 2427, решатели одинарной (верхняя линия) и двойной (нижняя линия) точности

Эффективность параллельных вычислений на малых задачах

Эффективное использование кластеров ограничивается размером задачи. Разбиение сетки приводит к созданию областей перекрытия в зонах связи между расчетными подобластями задачи. Процентное отношение количества наложенных (перекрытых) узлов расчетной сетки к общему количеству узлов сетки в оптимальном варианте должно быть минимальным, не более 10 % . Величина перекрытия более 20 % свидетельствует о том, что размер расчетной области на один процесс решателя достиг критического минимума - время вычислений на каждой подобласти будет незначительным по сравнению со временем обмена информацией между ними. Величина перекрытия сохраняется в диагностической информации процесса решателя ANSYS CFX при проведения вычислений.

Тестовые вычисления проводились на малой задаче с числом узлов равным 46 218. Загрузка вычислительных ядер проводилась по самой эффективной схеме – по одному ядру на узел. Эффективное ускорение в11,3 раза (эффективность 56,3%) было получено при работе кластера на 20 ядрах (рис. 11). При этом увеличение вычислительной мощности происходит практически линейно с коэффициентом ускорения равным 14,3. Минимальное эффективно используемое количество узлов сетки на одно ядро для исследуемой задачи находится в пределах от 2000 до 3000 (около 1500 элементов), что существенно ниже уровня, рекомендуемого документацией ANSYS CFX [1].

Эффективность использования методов декомпозиции расчетной области

Оптимизации процесса распараллеливания сетки для последующего использования кластерных систем происходит с использованием различных типовых программных средств декомпозии (в терминах ANSYS CFX – "partitioning"). Эффективность этих программных средств в большой мере зависит от типа и геометрии задачи. При проведении параметрических вычис-

лений, требующих больших временных затрат, необходимо выбрать оптимальный продукт, обеспечивающий максимальное ускорение вычислений. Для получения максимального масштабирования рекомендуется использовать предлагаемый по умолчанию декомпозитор (partitioner) "MeTiS". На рис. 7-11 приведены результаты масштабирования, полученные с использованием MeTiS'a. Использование этого программного средства позволило получить эффективное разбиение даже при очень низких числах узлов сетки на одно ядро. При использовании других методов разделения задачи при проведении вычислительных экспериментов было выявлено увеличение процента перекрытия расчетных подобластей по сравнению с MeTiS'ом, и, как следствие значимое снижение скорости вычислений. Эффективность использования MeTiS'a при масштабирования других типов задача требует дополнительных исследований.

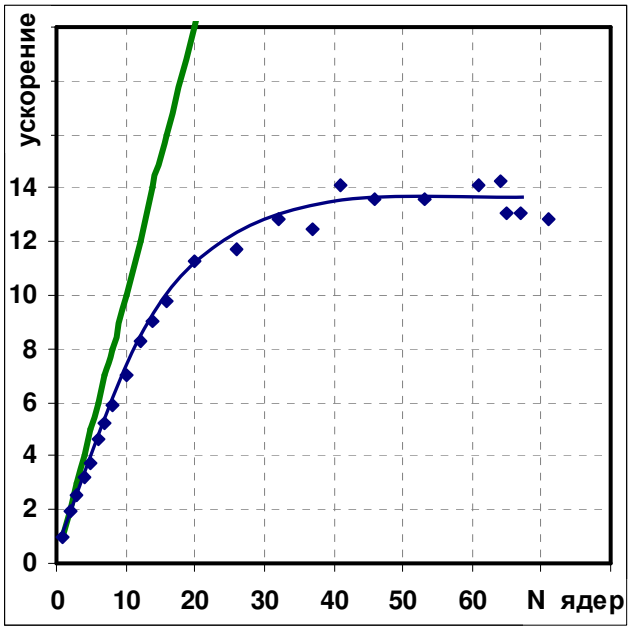


Рис.11 Максимальное масштабирование на малой задаче

Зависимость масштабирования от операционной системы и метода распараллеливания

На рис. 10 приведены результаты расчета малой задачи на двухпроцессорной четырехъядерной рабочей станции, характеристики которой приведены в таблице 2. Расчеты проводились в операционной системе MS Windows XP Professional x64 с использованием метода распараллеливания PVM Local Parallel, задаваемого по умолчанию, с использованием MPICH; в операционной среде Linux с использованием MPICH. Интенсивность вычислений оценивалась по числу итераций в минуту. При использовании метода PVM Local Parallel при увеличении числа вычислительных ядер происходит существенное снижение эффективности работы. При переходе на MPICH увеличение числа процессоров приводит к увеличению скорости вычислений. Максимальное ускорение получено при работе в операционной среде Linux.

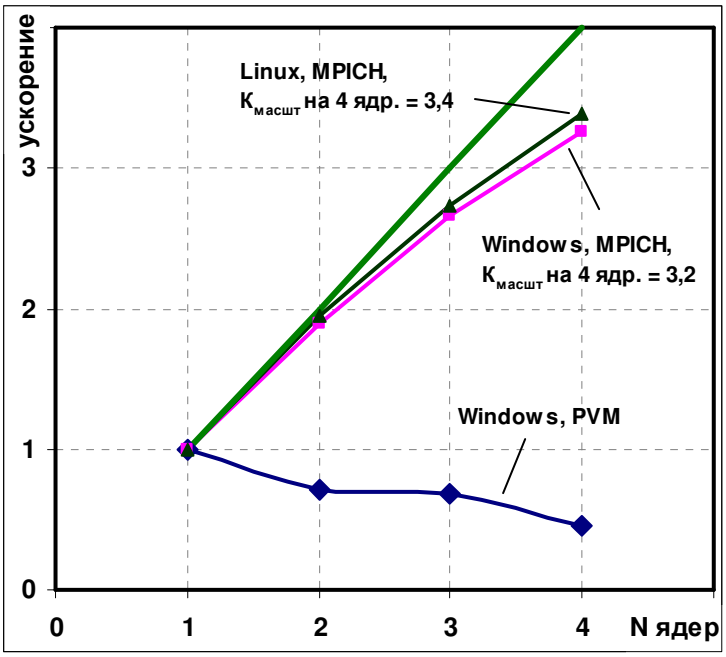


Рис.10. Масштабируемость раб. станции 2x AMD Opteron 2216
в зависимости от операционной системы и метода распараллеливания

Выводы

Вычислительная мощность современных аппаратных платформ и наличие хорошо масштабируемых программных пакетов вычислительной гидроаэродинамики позволяет эффективно решать большие задачи с размером вычислительной сетки в десятки миллионов узлов.

Проведен сравнительный анализ производительности и масштабируемости новейших процессоров AMD Opteron Istanbul и Intel Nehalem

Выявлены проблемы оценки производительности суперкомпьютерных систем – HPL тест не отражает реальное поведение системы на прикладном программном обеспечении (конечно-элементные пакеты, ВГАД)

При наладке кластера или установке нового программного обеспечения необходимо проведение работ по оптимизации загрузки суперкомпьютерной системы различным прикладным программным обеспечением для максимально эффективного использования вычислительных мощностей и имеющихся программных лицензий.

Литература

1. Руководство пользователя ANSYS CFX: ANSYS CFX-Solver Modeling Guide, Using the Solver in Parallel, Advice on Using CFX in Parallel.
2. Интернет-ресурс http://www.fluent.com/software/fluent/fl6bench/fl6bench_12.0/index.htm.
3. А.А.Аксенов, А.А.Дядькин, В.А.Кутин, И.В.Москалѐв, Г.Б.Сушко, С.А.Харченко Решение больших задач вычислительной гидродинамики на СКИФ МГУ с помощью FLOWVISION // Материалы Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ: решение больших задач», Новороссийск, 22–27 сентября 2008 г. М.: Изд-во Моск. Уни-та. 2008. С. 69-73.
4. М.А. Еремин, В.Н. Любимов Параллельный код трехмерного моделирования процессов космической газодинамики // Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 30 марта - 3 апреля 2010 г.).
5. М.В. Кудрявцев, В.В. Мошкин, М.А. Полунин Л.К. Эйсымонт Оценочное тестиррование кластеров на базе процессоров AMD Barcelona и Shanghai с сетями Infiniband DDR и QDR Вычислительные методы и программирование, 2009, т. 10, стр. 69-77.

Опыт решения задачи параметрического оценивания гидродинамической модели нефтяного месторождения на вычислительном кластере

Р.А. Байков, В.Г. Волков, А.В. Гагарин, Г.А. Макеев

В статье рассматривается автоматизированная система идентификации параметров гидродинамических моделей нефтяного месторождения, решающая задачу многомерного поиска и оптимизации. Исследуется применение алгоритмов многомерной оптимизации общего назначения, разработанных интеллектуальных методов поиска в многомерном пространстве, алгоритмов планирования эксперимента для анализа чувствительности и взаимозависимостей между искомыми параметрами. Предложенная система может быть использована для решения любой задачи оптимизации, в которой целевая функция является неявной и требует больших вычислительных ресурсов.

Введение

Решение задач параметрического оценивания и анализа чувствительности геолого-гидродинамических моделей (ГДМ) принципиально требует проведения массовых расчетов моделей с привлечением всех доступных вычислительных ресурсов, что сопряжено со значительными временными затратами и проведением большого объема рутинной работы специалистов по моделированию. Общий поток работ включает в себя, в частности:

- генерацию множества вариантов некоторой модели;
- распределение задач по доступным вычислительным мощностям;
- сбор и обработку результатов выполнения моделей.

Для формализации задачи параметрического оценивания предположим, что объект исследования (ОИ) характеризуется контролируемыми выходными сигналами $\mathbf{y}(k) = (y_1(k), y_2(k), \ldots, y_m(k))$, которые регистрируются в ходе натурного эксперимента в дискретные моменты времени t_k, $k = 1, 2, \ldots$ [11, 12]. Им соответствуют выходные сигналы $\hat{\mathbf{y}}(k \mid \mathbf{x}) = (\hat{y}_1(k \mid \mathbf{x}), \hat{y}_2(k \mid \mathbf{x}), \ldots, \hat{y}_m(k \mid \mathbf{x}))$ цифровой модели ОИ, определяемой вектором параметров $\mathbf{x}$. Невязкой $\varepsilon(k, \mathbf{x})$ называется разность между выходными сигналами ОИ $\mathbf{y}(k)$ и настраиваемой модели $\hat{y}_i(k \mid \mathbf{x})$:

$$\varepsilon(k, \mathbf{x}) = \mathbf{y}(k) - \hat{\mathbf{y}}(k \mid \mathbf{x}).$$

Пусть имеются данные наблюдения $\mathbf{y}(1), \ldots, \mathbf{y}(N)$ за ОИ в N различных моментов времени. Тогда задача параметрического оценивания модели заключается в выборе из всего множества параметров $\mathbf{x} \in D_M$ некоторого оптимального вектора $\mathbf{x}^{\text{opt}}$ таким образом, чтобы невязка $\varepsilon(k, \mathbf{x}^{\text{opt}})$, $k = 1, \ldots, N$ была по возможности мала.

Как правило, данная задача сводится к задаче минимизации целевой функции (ЦФ) вида:

$$f(\mathbf{x}) = \sum_{k=1}^{N} \varepsilon(k, \mathbf{x})^T \cdot \mathbf{W} \cdot \varepsilon(k, \mathbf{x}), \tag{1}$$

где $\mathbf{W}$ — матрица весовых коэффициентов.

Данная функция является мерой близости настраиваемой модели к реальному ОИ и отражает качество идентификации ее параметров. Соответственно,

$$\mathbf{x}^{\text{opt}} = \underset{\mathbf{x} \in D_M}{\operatorname{argmin}} f(\mathbf{x}).$$

На практике ищется минимум $f(\mathbf{x})$ с заданным порогом δ, позволяющим контролировать точность получаемого результата. Иными словами, необходимо найти решение $\mathbf{x}^*$ такое, что:

$$f(\mathbf{x}^*) < f\left(\mathbf{x}^{\text{opt}}\right) + \delta. \tag{2}$$

Для нахождения минимума $f(\mathbf{x})$, удовлетворяющего условию (2), обычно применяются итерационные методы оптимизации, генерирующие последовательность приближений $\left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N_o)}\right\}$ такую, что $\mathbf{x}^{(N_o)} \in D_M^*$ (то есть $\mathbf{x}^{(N_o)}$ удовлетворяет условию (2)). Значение индекса N_o показывает количество приближений, требуемых для нахождения оптимума ЦФ с заданным порогом. На практике, в дополнение к критерию (2) часто применяется ограничение по времени, когда оптимизация останавливается по условию $N_o = N_o^{\max}$.

В зависимости от сложности модели и величины N, ресурсоемкость вычисления $\hat{\mathbf{y}}(k \mid \mathbf{x})$, $k = 1, \ldots, N$, может быть велика. Например, время численного моделирования течения многофазной смеси жидкости и газа в пористой среде, широко используемое в моделях нефтяных месторождений, даже на современных суперкомпьютерах с применением коммерческих симуляторов, таких как Shlumberger ECLIPSE, Roxar Tempest MORE, NGT BOS[1] и т. д., может исчисляться сутками. Задача оптимизации ГДМ является примером некорректно поставленной обратной задачи с ЦФ вида (1). У нее может не существовать решения, может существовать больше одного решения, и неизвестна устойчивость решения. В таких условиях одним из методов решения могут быть методы глобальной и локальной оптимизации общего назначения, такие как градиентные и генетические алгоритмы [1–3,6]. Среди особенностей задачи моделирования можно выделить большое количество искомых параметров и возможную неявную зависимость между этими параметрами.

В связи с этим большую актуальность приобретает задача автоматизации процесса многовариантного моделирования, то есть полного цикла генерации вариантов моделей, их расчета и обработки результатов для решения различных прикладных задач. Результатом работы должна стать единая универсальная среда выполнения возможных сценариев для оптимизации моделей, пользуясь которой специалисты смогут сосредоточиться на решении только принципиальных задач моделирования.

1. Описание разработанной системы

Разрабатываемая система автоматизированной оптимизации ГДМ основана на совместном использовании вычислительной среды MATLAB и гидродинамического симулятора NGT BOS Core. Типичный цикл оптимизации модели с помощью этой системы включает следующие этапы (рисунок 1).

[1]Разработка ООО «РН-УфаНИПИнефть»

Рис. 1. Общая структура системы

На первом этапе производится выбор варьируемых параметров модели, определяется вид ЦФ, и способ преобразования аргументов целевой функции в варьируемые параметры модели. Далее под параметрами понимаются именно аргументы ЦФ.

На втором этапе определяется область поиска оптимальных значений параметров (границы) и шаги дискретизации области поиска по всем параметрам.

На третьем этапе анализируется чувствительность ЦФ к искомым параметрам, определяются однородные и неоднородные области параметрического пространства.

На четвертом этапе с помощью алгоритмов оптимизации общего назначения (генетических, градиентных и т.д.) система самостоятельно выполняет цикл, в котором генерируются ГДМ, соответствующие приближениям , производится численное моделирование, необходимое для вычисления ЦФ и обрабатываются результаты расчета.

На пятом этапе анализируются найденные решения и кластеры решений, определяется их устойчивость.

И на шестом этапе строятся отчеты по лучшим найденным решениям и интегральные отчеты в разрезе всех рассчитанных решений.

1.1. Матрица «параметры-оценки»

Параметры и соответствующие им рассчитанные значения ЦФ в системе формируют матрицу параметров и оценок (расширенный и отформатированный вид ее приведен на рис. 2). Матрица имеет $n + l$ столбцов и e строк, где n — количество параметров, а l — количество оценок. Каждая строка соответствует одному эксперименту (одной ГДМ). Все модели различаются только значениями n варьируемых параметров, а в остальном идентичны.

	Параметры						Оценки			
	P1	P2		PN			E1	E2	…	EM
N эксп.	Забойное давление там-то	Длина трещины такой-то	…	Признак установки эквалайзера		**Состояние**	Общая добыча нефти	Газовый фактор на скважине такой-то		Средняя обводненность
1	250	50		1		готово	120	1		0.7
2	300	75		1		готово	130	1		0.7
…										
E	250	50		1		**рассчитывается**				

Рис. 2. Пример матрицы параметров и оценок

Добавление новой строки в матрицу «параметры-оценки» приводит к автоматическому созданию новой гидродинамической модели. Система позволяет произвести численное моделирование как на локальном компьютере, так и на кластере. После проведения численного моделирования вычисляется значение ЦФ, соответствующее данной ГДМ.

Поскольку матрица «параметры-оценки» доступна в среде MATLAB, то добавляться новые строки могут:

- на основе приближений $x^{(\nu)}$ генерируемых алгоритмом оптимизации;
- вручную, если специалисту требуется проверить модель с заданными набором параметров;
- функцией группового добавления: например, в соответствии с алгоритмом планирования эксперимента (экспериментальный дизайн).

Важной особенностью матрицы «параметры-оценки» является то, что в ней не может быть двух моделей, все параметры которых совпадают. Это означает, что численное моделирование с уникальным набором параметров модели производится только один раз.

1.2. Параметризация модели

Система позволяет использовать в качестве аргумента ЦФ любое значение, прямо или опосредованно влияющее на параметры ГДМ. С точки зрения системы любой аргумент ЦФ имеет свою область определения и шаг дискретизации. С помощью языка MATLAB можно производить сложные преобразования аргументов ЦФ в параметры ГДМ. Формально, преобразование аргументов ЦФ в параметры модели в разработанной системе выполняется с помощью двух механизмов:

- подстановка на место параметра ГДМ аргумента целевой функции;
- вызов некоторой функции MATLAB, осуществляющей преобразование аргументов ЦФ в параметры ГДМ.

1.3. Применяемые алгоритмы оптимизации

Поскольку задача параметрического оценивания модели формулируется в виде задачи оптимизации, важнейшую роль в системе играет используемый алгоритм оптимизации, в значительной степени определяющий способность исследования пространства поиска, скорость поиска решения, возможность нахождения глобального оптимума и т.д.

Разработанная система построена по модульному принципу и позволяет использовать различные оптимизационные блоки, обладающие одинаковым программным интерфейсом (API). В частности, на данный момент система включает реализации градиентного поиска, усовершенствованных генетических алгоритмов (ГА), эвристического алгоритма, основанного на методе кригинга (Kriging), алгоритма Монте-Карло. Кроме того, имеются также модули планирования эксперимента, факторного анализа, анализа чувствительности и зависимостей между параметрами и оценками.

Поскольку нахождение параметров адекватной модели, обладающей хорошими прогнозными свойствами — нетривиальная задача, которую невозможно полностью автоматизировать, специалист по моделированию может на различных уровнях настраивать поведение системы. В частности, именно он определяет выбирает оптимизационное ядро для решения прикладных задач. Однако, на практике в общем случае ЦФ имеет очень сложную форму и область локализации ее глобального оптимума неизвестна, поэтому целесообразно на начальном этапе использовать алгоритмы с возможностями глобальной оптимизации и уже на этапе уточнения решения переходить к методам локального поиска.

1.3.1. Гибридный генетический нейросетевой алгоритм

Хорошо известно, что использование стандартного ГА предполагает на каждой итерации для каждого набора параметров (хромосомы в терминах ГА) вычисление значений ЦФ, включающее в себя моделирование ОИ (то есть расчет значений $\hat{\mathbf{y}}(k \mid \mathbf{x})$). Разработанный *гибридный генетический нейросетевой алгоритм* (ГА+НС) является модификацией стандартного ГА, в дальнейшем называемого *главным*, на каждой итерации которого создается нейросетевая аппроксимация $\widetilde{f}(\mathbf{x})$ целевой функции $f(\mathbf{x})$, предназначенная для получения прогноза $\widetilde{\mathbf{x}}^*$ оптимального решения функции $f(\mathbf{x})$. В большинстве случаев $\widetilde{f}(\mathbf{x})$ будет вычисляться значительно быстрее $f(\mathbf{x})$, поскольку не требует предварительного расчета модели. Поиск $\widetilde{\mathbf{x}}^*$ осуществляется с помощью дополнительного ГА, в дальнейшем называемого *вспомогательным*, для которого в качестве ЦФ используется $\widetilde{f}(\mathbf{x})$ (то есть значения, вычисляемые нейронной сетью). Найденный прогноз затем добавляется в популяцию потомков главного ГА на текущей итерации и цикл повторяется.

Адекватный прогноз $\widetilde{\mathbf{x}}^*$ оптимального решения может значительно ускорить эволюционный поиск основного ГА. Если прогноз окажется неудачным, эволюционные механизмы обеспечат стабильность оптимизации. Приближения $\mathbf{x}$, генерируемые основным ГА на про-

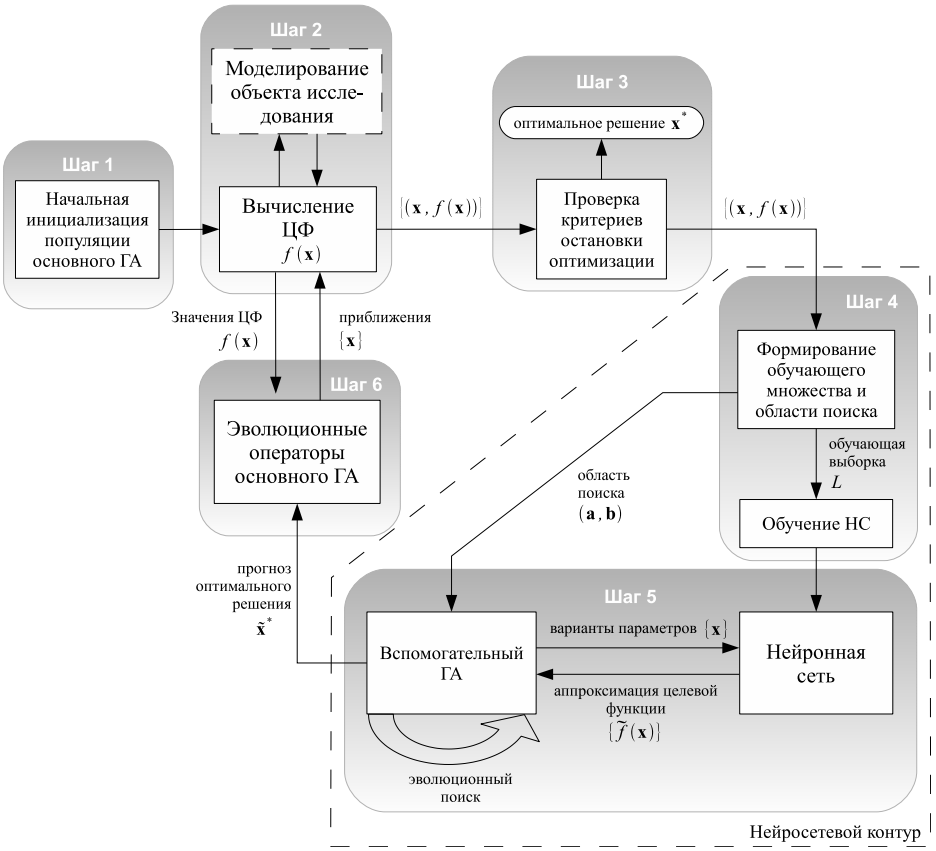


Рис. 3. Схема алгоритма ГА+НС

тяжении всего процесса оптимизации, вместе с соответствующими значениями ЦФ $f(\mathbf{x})$ участвуют в формировании обучающей выборки для НС (стандартный ГА хранит лишь текущую популяцию фиксированного размера). Вспомогательный ГА может выполнять значительно более интенсивный поиск по сравнению с главным ГА (большой размер популяции, много итераций и т. п.) и с большой вероятностью находить глобальный оптимум $\widetilde{f}$, так как данная функция вычисляется сравнительно быстро.

В современной литературе методы идентификации систем, основанные на замене сложной модели реального ОИ на значительно менее ресурсоемкую модель, приближенно воспроизводящую отклик исходной модели, называются суррогатным моделированием или метамоделированием [8, 10]. Суррогатные модели ОИ обычно строятся с минимальным привлечением данных из предметной области на основе имеющихся вычислительных экспериментов с исходной моделью или данных наблюдения за реальным ОИ с помощью различных методов построения аппроксимации многомерных зависимостей (аппроксимирующие полиномы, сплайны, радиально-базисные функции, кригинг, нейронные сети и т. д.). Прямое применение данного подхода с использованием НС для построения суррогатной модели путем обучения на заранее подготовленной обучающей выборке, и дальнейшего применения ГА для идентификации параметров суррогатной модели описано, например, в работах [5, 7, 9]. Такую схему можно рассматривать как одну итерацию нейросетевого контура предлагаемого алгоритма ГА+НС. Отличительной особенностью ГА+НС является то, что НС используется не для построения отображения $(\mathbf{x}, k) \mapsto \hat{\mathbf{y}}(k \mid \mathbf{x})$, а для аппроксимации ЦФ (которая, в свою очередь зависит от $\mathbf{y}(k)$ и $\hat{\mathbf{y}}(k \mid \mathbf{x})$). Кроме того, предлагаемый алгоритм не требует наличия априорно заданной обучающей выборки, нейросетевая аппроксимация $\widetilde{f}$ строится динамически на каждой итерации главного ГА на основе имеющегося на данный момент множества пар $\{(\mathbf{x}, f(\mathbf{x}))\}$.

Наиболее близкий аналог ГА+НС предлагается в работе [4]. Гибридный алгоритм в

[4] не содержит вспомогательного ГА, поскольку на вход НС подается значение ЦФ, а на выходе получается прогноз соответствующего вектора искомых параметров. Однако такая схема имеет очевидный недостаток, заключающийся в некорректности обратной задачи, решаемой НС.

1.3.2. Кригинг-оптимизатор

Кригинг — семейство геостатистических алгоритмов интерполяции неизвестной случайной величины в некоторой точке по известным реализациям этой случайной величины в некоторых других точках. В общем виде значение $Z^*(u)$ в интерполируемой точке u определяется следующим образом через известные реализации случайных величин в точках u_α, $\alpha = \overline{1,n}$:

$$Z^*(u) = m(u) + \sum_{\alpha=1}^{n(u)} \lambda_\alpha(u) \left(Z(u_\alpha) - m(u_\alpha)\right),$$

где $Z(u_\alpha), Z(u)$ — реализации случайных величин в точках u и u_α;
$m(u_\alpha), m(u)$ — мат. ожидания этих случайных величин;
$\lambda_\alpha(u)$, $\alpha = \overline{1,n}$ — веса известных реализаций , меняющиеся при переходе к следующей точке u.

Основная идея кригинга в том, что веса $\lambda_\alpha(u)$, $\alpha = \overline{1,n}$ выбираются так, чтобы интерполируемое значение $Z^*(u)$ соотносилось с истинным (неизвестным) значением $Z(u)$ в этой же точке следующим образом:

$$E\left(Z^*(u) - Z(u)\right) = 0,$$
$$D\left(Z^*(u) - Z(u)\right) \to \min.$$

Данные, которыми оперирует кригинг, включают не только значения $Z(u_\alpha)$, но и априорная информация о моделируемых случайных величинах в виде ковариационной функции $Cov\left(Z(u), Z(u+h)\right) = C_R(h)$ при допущении, что последняя не зависит от положения u, а зависит только от расстояния h между точками.

Кригинг-оптимизатор — набор параллельно работающих эвристик, основанных на кригинг-интерполяции параметрического пространства и принятии решений о новых точках на основании результата интерполяции.

Использование кригинг-интерполяции в данном случае не имеет под собой твердой математической основы, так как в случае параметрического пространства целевой функции нельзя говорить о каких-либо случайных величинах, тем более об их ковариациях. Однако использование кригинг-интерполяции как еще одного алгоритма интерполяции, обладающего свойствами сохранения значений в известных точках и удовлетворительного управляемого сглаживания вполне допустимо.

Кригинг получает на вход все имеющиеся наборы аргументов и значений ЦФ, рассматриваемые как реализации некоторой случайной величины, и строит:

- оценку мат. ожидания E в требуемых точках — прогноз значения ЦФ в неизвестной точке;
- оценку дисперсии D в требуемых точках — в некотором смысле, меру «неизвестности» параметрического пространства в неизвестной точке (она же мера удаленности неизвестной точки от всех известных);
- оценку «локальной изменчивости» вида $LD = \frac{\sum_{i=1}^{N} \lambda_i (y_i - E)^2}{N}$ — в некотором смысле меру неоднородности параметрического пространства в неизвестной точке.

Эвристические стратегии поиска, используемые кригинг-оптимизатором, выбирают место в параметрическом пространстве для создания новой точки так, чтобы:

- мат. ожидание $E \to \max$ — попытка воспользоваться интерполяцией для поиска оптимального значения;
- дисперсия $D \to \max$ — исследование неизвестных областей параметрического пространства;
- локальная изменчивость $LD \to \max$ — исследование областей параметрического пространства с высокой изменчивостью;
- $D \cdot LD \to \max$ — исследование неизвестных областей параметрического пространства с высокой изменчивостью.

Кригинг-оптимизатор обладает следующими достоинствами:

- одновременно использует все эвристики в определенном соотношении;
- не имеет «истории», может быть остановлен в любой момент и запущен опять;
- равномерно заполняет параметрическое пространство, но в отличие от регулярных методов, делает это при любом имеющемся количестве «попыток».

1.3.3. Модули планирования экспериментов

Модули планирования экспериментов предназначены для исследования параметрического пространства путем заполнения его множеством точек в соответствии с некоторым алгоритмом планирования эксперимента:

- случайный засев;
- полный факторный анализ;
- алгоритм Box-Behnken Design;
- алгоритм Central Composite Design.

1.3.4. Модули анализа чувствительности и независимости параметров

Выделение независимых аргументов или групп аргументов ЦФ, основанное на определении чувствительности ЦФ, а также контролируемых параметров модели $\mathbf{y}(k)$ к аргументам ЦФ, является эффективным способом уменьшения размерности пространства поиска.

Анализ чувствительности в разработанной системе основан на оценке частных производных по каждому параметру в некоторых (заданных пользователем или выбираемых вручную) точках. Результатом работы блока оценки чувствительности является отчет, приведенный на рисунке 4. Каждой строке соответствует аргумент ЦФ, а каждому столбцу соответствует контролируемый параметр или значение ЦФ. Отчет позволяет выявить разбиение аргументов ЦФ на независимые группы.

2. Опыт применения для задач моделирования

В этом разделе приводится описание задач оптимизации ГДМ, для которых была применена разработанная система. В общем виде все задачи следуют одной и той же схеме. Раз система оптимизации умеет менять только числа, и с помощью алгоритмов параметрической оптимизации находить оптимальные их значения для некоторой определенной функции, то для любой фактической задачи моделирования требуется:

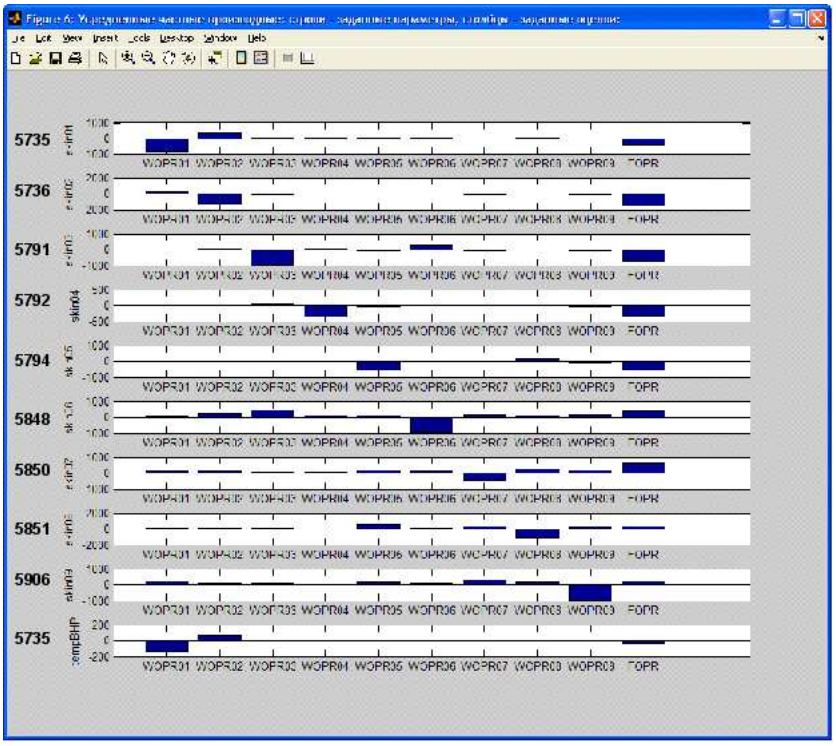


Рис. 4. Пример оценки чувствительности параметров

- параметризовать модель, то есть определить набор изменяемых численных параметров со своими областями определения, и определить процесс трансформации определенных значений параметров в конкретные (часто достаточно большие и сложные) изменения в модели;
- определить целевую функцию, которая для рассчитанной модели возвращает некоторое число, характеризующее то, насколько текущая модель «близка» к поставленной цели;
- запустить систему параметрической оптимизации, которая решает задачу оптимизации путем расчета большого количества ГДМ.

Далее приведено несколько примеров, которые должны проиллюстрировать разнообразие задач, которые могут быть решены в рамках разработанной системы.

2.1. Настройка относительных фазовых проницаемостей в виде степенной функции и кусочно-линейной функции

Относительная проницаемость указывает на способность нефти и воды одновременно течь в пористой среде и определяется как отношение эффективной проницаемости фазы к абсолютной (когда порода заполнена только одной фазой). ОФП задается в виде кривой зависимости относительной проницаемости данной фазы (нефть, вода, газ) от водонасыщенности, нормированной к диапазону $[0; 1]$. Форма кривых ОФП воды и нефти аппроксимировалась степенными функциями, форма которых задавалась с помощью восьми параметров. Задача заключалась в поиске таких значений этих параметров, чтобы результаты расчета полученной гидродинамической модели, минимизировали следующую ЦФ:

$$f(\mathbf{x}) = \sum_{k=1}^{N} \sum_{i=1}^{m} \sum_{j=1}^{u} \frac{(y_{i,j}(k) - \hat{y}_{i,j}(k \mid \mathbf{x}))^2}{\sigma_i^2}$$

где k — номер временного шага;

i — индекс интересующего контрольного параметра;

j — номер скважины;

σ_i — дисперсия наблюдаемых значений i-го параметра.

Учитывались следующие контрольные величины: мгновенные и суммарные дебиты воды и нефти, дебит жидкости, суммарный объем закачки, обводненность скважин. Исторические значения этих величин были предварительно получены из результатов расчета модели с экспертно заданными ОФП. Эти данные были приняты в качестве эталонных и использовались для сравнения с расчетными значениями в процессе оптимизации. В качестве ядра оптимизации применялся алгоритм ГА+НС.

В ходе оптимизации было сделано 10 итераций алгоритма ГА+НС, на каждой из которых оценивалась популяция из 10 моделей. Результаты эксперимента представлены на рисунках 5 и 6.

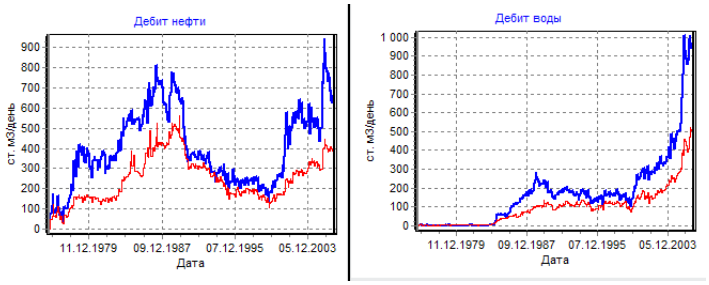


Рис. 5. Начальное приближение контрольных величин

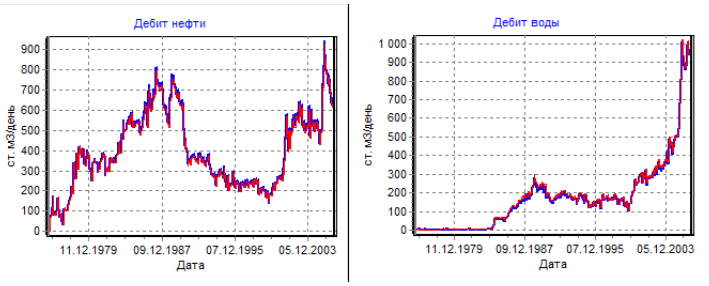


Рис. 6. Контрольные величины после оптимизации

2.2. Подбор петрофизической зависимости между проницаемостью и пористостью

Трехмерные массивы, определяющие пористость и проницаемость в каждой ячейке модели месторождения представляют собой часть численной модели, с которой работает симулятор. Исходными данными для построения этих кубов являются разнообразные исследования керна скважин при бурении и в процессе работы скважины, и прочие источники информации, обладающие различной достоверностью.

Принято считать, что статистически пористость m и проницаемость k связаны соотношением $m = a + b \cdot \log_{10} k$, где a и b — константы для некоторого региона. Эти константы, как правило, находятся каким-нибудь методом линейной регрессии по облаку точек $(m, \log_{10} k)$ (рисунок 7, слева).

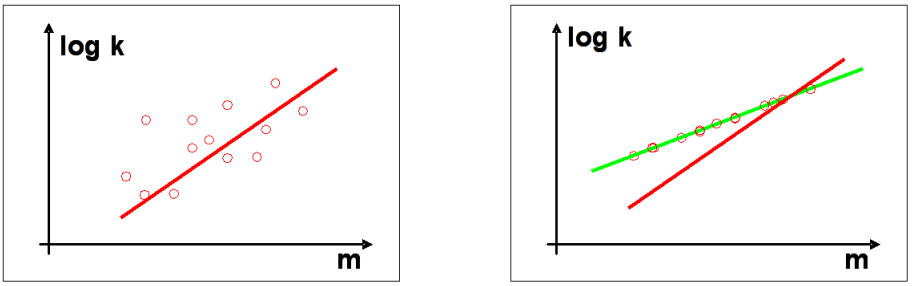


Рис. 7. Зависимость между пористостью и проницаемостью

Была решена задача в следующей постановке. Куб пористости модели был зафиксирован, как обладающий достаточной достоверностью. Выбирались некоторые значения a и b, и по приведенному выше петрофизическому соотношению из куба пористости рассчитывался куб проницаемости. Фактически, в облаке точек проводилась новая прямая, и все точки «сажались по вертикали» на эту прямую: значение m оставалось одно и то же, значение $\log_{10} k$ менялось (правый рисунок).

Полученная модель рассчитывалась на симуляторе, и далее рассчитывалась невязка между историческими и расчетными значениями добычи нефти и воды. Таким образом, задача сводилась к задаче параметрической оптимизации петрофизических параметров a и b для минимизации численного значения невязки. И уже эта задача параметрической оптимизации решалась стандартными средствами разработанной системы. Таким образом, система оптимизации варьировала целый куб проницаемости ГДМ, путем параметризации этого куба всего двумя параметрами.

2.3. Адаптация карты проницаемости

Данные трехмерного куба проницаемости в модели как правило являются вторичными, пересчитанными из данных куба пористости по принятым (пусть и в результате эксперимента) петрофизическим зависимостям. Принято считать, что эти данные можно менять в процессе оптимизации модели, однако изменения не должны приводить к очевидно нефизичным результатам. Одним из таких изменений, например, является повышение или понижение проницаемости в ячейках куба, прилегающих к скважинам, однако такое изменение должно быть «плавным».

Для адаптации куба проницаемости был реализован следующий алгоритм. Выбирались некоторые значения множителей на проницаемость на некоторых скважинах, полученные значения добавлялись в пустую «карту множителей», которая после этого заполнялась целиком интерполяцией добавленных множителей. Каждый горизонтальный слой куба проницаемости затем умножался на гладкую карту множителей. Полученная модель рассчитывалась на симуляторе и вычислялась невязка между историческими и расчетными

значениями добычи нефти и воды.

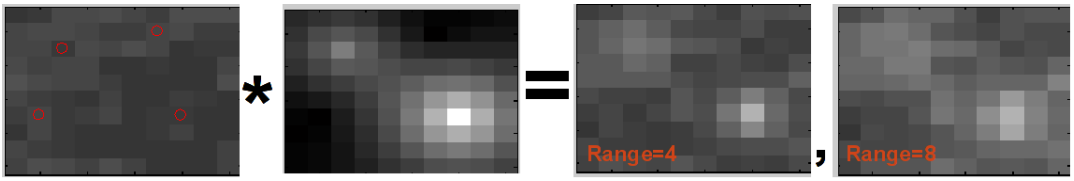


Рис. 8. Преобразование карты проницаемости

На рисунке 8 показан фрагмент одного горизонтального слоя начального куба проницаемости и четыре скважины. На левой нижней и правой верхней скважинах множитель устанавливался равным 1. На левой верхней скважине он был равен 2, а на правой нижней множитель был равен 3. На среднем рисунке показан фрагмент интерполированной карты множителей. На правом рисунке показан фрагмент произведения слоя начальной пористости на карту множителей. Задача заключалась в параметрической оптимизации некоторого ограниченного числа множителей на проницаемость на некоторых скважинах для минимизации численного значения невязки. И опять-таки, после параметризации задачи, она была решена стандартными средствами разработанной системы. Таким образом, система оптимизации варьировала целый куб проницаемости ГДМ, путем параметризации этого куба значениями множителей на некотором наборе скважин.

2.4. Оптимизация траектории скважины

Задачи, так или иначе связанные с оптимизацией траекторий новых скважин, могут быть поставлены множеством различных способов. Это может быть выбор местоположения устья скважины, подбор схемы и параметров размещения групп скважин на месторождении (системы разработки), выбор направления и угла наклона сегмента горизонтальной скважины и т.д.

Во всех этих постановках требуется, как и во всех предыдущих задачах оптимизации, выбрать правильную параметризацию, чтобы система оптимизации вносила изменения в модель, меняя только малое количество численных параметров.

Рис. 9. Оптимизация траектории скважины

Например, в случае, если траектория скважины задается в модели путем определения координат всех точек траектории, фиксация X и Y координат начала и конца некоторого сегмента, но варьируя Z координату этих точек в некоторых пределах (красные отрезки на рисунке), мы получаем задачу выбора угла наклона сегмента горизонтальной скважины. Данная задача решалась одновременно с выбором марки устройства управления лабиринтным притоком, которое устанавливается на скважину для предотвращения преждевременного прорыва газа. Целевой функцией в этом случае была максимизация добычи нефти при минимизации (или контроле непревышения некоторого порога) добычи газа.

Выводы

1. Использование кластерных вычислений позволяет эффективно решать разнообразные обратные задачи при идентификации параметров ГДМ путем решения множества прямых задач.

2. Использование разработанных алгоритмов оптимизации позволяет резко сократить количество численных экспериментов.

3. Разработанная система оптимизации ГДМ активно и успешно используется сотрудниками РН-УфаНИПИнефть при адаптации ГДМ.

Список литературы

1. *Ballester, P. J.* A parallel real-coded genetic algorithm for history matching and its application to a real petroleum reservoir / P. J. Ballester, J. N. Carter // *Journal of Petroleum Science and Engineering*. — 2007. — Vol. 59. — Pp. 157 – 168.

2. Evolutionary algorithms applied to history matching of complex reservoirs / R. Schulze-Riegert, J. Axmann, O. Haase, et al. // *SPE Reservoir Evaluation & Engineering*. — 2002.

3. *Gomez, S.* Gradient-based history-matching with a global optimization method / S. Gomez, O. Gosselin, J. Barker // *Society of Petroleum Engineering Journal*. — 2001. — Vol. 6. — Pp. 200–208.

4. *Javadi, A. A.* A hybrid intelligent genetic algorithm / A. A. Javadi, R. Farmani, T. P. Tan // *Advanced Engineering Informatics*. — 2005. — Vol. 19, no. 4. — Pp. 255–262.

5. *Kuo, J.-T.* A hybrid neural-genetic algorithm for reservoir water quality management. / J.-T. Kuo, Y.-Y. Wang, W.-S. Lung // *Water Res.* — 2006. — Apr. — Vol. 40, no. 7. — Pp. 1367–1376. `http://dx.doi.org/10.1016/j.watres.2006.01.046`.

6. *Soleng, H.* Oil reservoir production forecasting with uncertainty estimation using genetic algorithms / H. Soleng // Proc. Congress on Evolutionary Computation CEC 99. — Vol. 2. — 1999. — 6–9 July.

7. *Srinivas, V.* An integrated approach for optimum design of bridge decks using genetic algorithms and artificial neural networks / V. Srinivas, K. Ramanjaneyulu // *Advances in Engineering Software*. — 2006. — no. 38. — Pp. 475 – 487. `www.elsevier.com/locate/advengsoft`.

8. *Wang, G. G.* Review of metamodeling techniques in support of engineering design optimization / G. G. Wang, S. Shan // *Journal of Mechanical Design*. — 2007. — Vol. 129, no. 4. — Pp. 370–380. `http://dx.doi.org/10.1115/1.2429697`.

9. *Wang, L.* A hybrid genetic algorithm-neural network strategy for simulation optimization / L. Wang // *Applied Mathematics and Computation*. — 2005. — Vol. 170. — Pp. 1329–1343.

10. *Кулешов, А. П.* Когнитивные технологии в основанных на данных адаптивных моделях сложных объектов / А. П. Кулешов // *Информационные технологии и вычислительные системы*. — 2008. — № 1. — С. 18–29.

11. *Льюнг, Л.* Идентификация систем. Теория для пользователя: Пер. с англ. / Л. Льюнг; Под ред. Я. З. Цыпкина. — М.: Наука. Гл. ред. физ.-мат. лит., 1991. — С. 432.

12. *Цыпкин, Я. З.* Информационная теория идентификации / Я. З. Цыпкин. — М.: Наука. Физматлит, 1995. — С. 336.

Асинхронное программирование численных задач

С.Б. Арыков

В статье обсуждается система параллельного программирования Аспект, позволяющая разрабатывать асинхронные программы для решения численных задач. Предложены формальная модель вычислений, являющаяся развитием асинхронной модели с массовыми операциями и специализированный язык для фрагментированного представления алгоритмов. Кратко рассмотрены особенности реализации системы программирования Аспект, приведены результаты тестовых испытаний на модельных задачах.

1. Введение

Численное моделирование – одна из областей, где задача разработки параллельных программ была актуальна всегда. И хотя именно в этой области накоплен наибольший опыт параллельных вычислений, создание параллельных программ по-прежнему требует высокой квалификации и специфических знаний. Поэтому необходимо развитие высокоуровневых систем программирования, которые, по возможности, скрывают от программиста технические детали, позволяя сосредоточиться на алгоритме решаемой задачи.

Исследовательские работы по повышению уровня систем программирования активно ведутся как в России, так и за рубежом. Среди российских систем можно отметить такие проекты как DVM [1] и mpC [2], избавляющие программиста от необходимости программировать коммуникаций, а также OpenTS [3] и НОРМА [4], выполняющие автоматическую генерацию параллельных программ; среди зарубежных работ выделяются Charm++ [5], ALF [6] и RapidMind [7]. Несмотря на большое количество проектов, до сих пор основными средствами разработки остаются OpenMP и MPI, что говорит об актуальности дальнейших исследований в данном направлении.

В работе рассматривается система параллельного программирования Аспект [8-9], в которой за счет специализации предполагается существенно повысить уровень языка программирования. Отличительными чертами системы Аспект являются использование фрагментированного представления алгоритма, асинхронной модели вычислений и ориентации на задачи численного моделирования.

Фрагментированное представление изначально разрабатывалось для параллельных вычислений и позволяет автоматически генерировать высокоэффективные параллельные программы для выбранной предметной области, а также обеспечивать ряд динамических свойств, таких, как настройка на доступные ресурсы и балансировка нагрузки. Специализированность вводит ряд ограничений, существенных для технической реализации системы.

Далее рассматривается формальная модель вычислений, на базе которой построена система Аспект, приводится обзор языка Аспект и реализации транслятора и исполнительной подсистемы, а также приводятся результаты тестовых испытаний на модельных задачах.

2. Модель вычислений

2.1 Представление алгоритма

Процедурное представление алгоритма плохо подходит для высокоуровневой системы параллельного программирования, так как накладывает чрезмерно жесткое прямое управление на алгоритм, скрывает его естественный параллелизм, и, таким образом, существенно усложняет решение задачи по распределению ресурсов.

Фрагментированное представление позволяет преодолеть указанный недостаток. Оно заключается в представлении алгоритма в виде множества *фрагментов данных* и *фрагментов кода*. Фрагмент кода получает на вход набор *входных фрагментов данных*, на основе которых вычисляет набор *выходных фрагментов данных*. Подстановка фрагментов данных в качестве

параметров фрагмента кода называется *применением* фрагмента кода к фрагментам данных (один и тот же фрагмент кода может применяться к различным фрагментам данных). Совокупность фрагмента кода и его входных и выходных фрагментов данных называется *фрагментом вычислений*. На множестве фрагментов вычислений задаётся частичный порядок (*управление*).

Исполнение фрагментированной программы состоит в исполнении фрагментов вычислений в любом порядке, не противоречащем заданному управлению, при этом каждый фрагмент вычислений получает свои ресурсы в момент назначения на исполнение, порождает новый процесс программы и может мигрировать с одного процессора на другой.

Поясним использование фрагментированного подхода на простом примере. Пусть имеются три матрицы *A*, *B* и *C* размером 9x9 каждая, матрица *C* инициализирована нулями. Необходимо разработать фрагментированный алгоритм умножения матриц.

Разобьем каждую матрицу на подматрицы размера 3x3 (рис. 1). Каждая подматрица есть фрагмент данных, который будем обозначать именем матрицы с соответствующим номером. Например, $C_{(1,1)}$ обозначает первый фрагмент матрицы *C*.

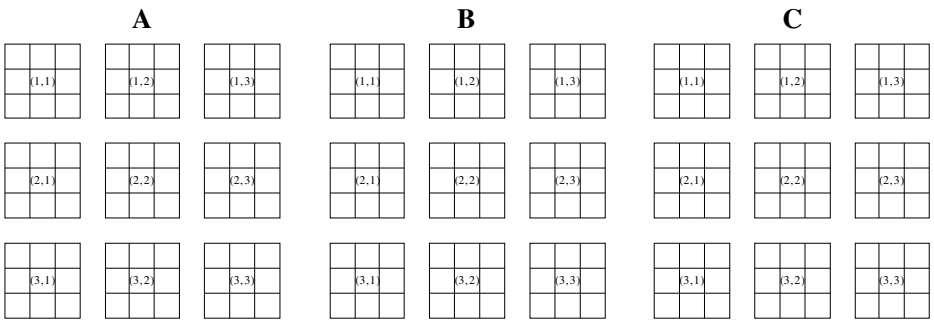


Рис. 1. Фрагментация задачи умножения матриц

Далее, зададим фрагмент кода *F*, который будет получать на вход два фрагмента данных и выполнять умножение первого фрагмента на второй по правилу умножения матриц:

$$c_{ij} = \sum_k a_{ik} b_{kj} \qquad (1)$$

(здесь c_{ij} – элемент в результирующем фрагменте). Тогда произведение матрицы *A* на матрицу *B* можно вычислить по следующей формуле:

$$C_{(i,j)} = \sum_k A_{(i,k)} B_{(k,j)}$$

(здесь $C_{(i,j)}$ – фрагмент матрицы *C*). Сложение результатов можно выполнять как отдельным фрагментом кода, так и встроить его во фрагмент кода *F* (далее предполагается именно этот вариант).

Теперь необходимо применить фрагмент кода к соответствующим фрагментам данных, т. е. задать фрагменты вычислений. Будем обозначать фрагмент вычислений буквой фрагмента кода, за которой в скобках перечислены фрагменты данных, используемые фрагментом кода. Тогда, чтобы полностью вычислить $C_{(1,1)}$, необходимо выполнение трёх фрагментов вычислений: $F(A_{(1,1)}, B_{(1,1)}, C_{(1,1)})$, $F(A_{(1,2)}, B_{(2,1)}, C_{(1,1)})$, $F(A_{(1,3)}, B_{(3,1)}, C_{(1,1)})$. Поскольку каждый из них осуществляет запись во фрагмент данных $C_{(1,1)}$, для исключения ситуации гонок данных необходимо задать порядок исполнения фрагментов вычислений. Например, можно потребовать, чтобы $F(A_{(1,2)}, B_{(2,1)}, C_{(1,1)})$ исполнялся после $F(A_{(1,1)}, B_{(1,1)}, C_{(1,1)})$, а $F(A_{(1,3)}, B_{(3,1)}, C_{(1,1)})$ после $F(A_{(1,2)}, B_{(2,1)}, C_{(1,1)})$. Аналогично формируются фрагменты вычислений для нахождения остальных фрагментов данных матрицы *C*, в результате чего получается фрагментированная программа. Заметим, что фрагменты вычислений, вычисляющие различные фрагменты данных матрицы *C*, могут исполняться параллельно.

Из приведённого примера можно сделать несколько важных выводов:

1. Алгоритм, использованный для реализации фрагмента кода, отличается от формулы (1) дополнительной операцией сложения. Это отличие не случайно: фрагментация алгоритма может потребовать его существенного изменения и, следовательно, не может быть выполнена автоматически.
2. Фрагментировать алгоритм можно различными способами, при этом полученная программа будет иметь разную степень непроцедурности. Например, для матриц размером 8x8 можно использовать фрагменты 2x2, 2x4, 4x4 и др. Важно лишь, чтобы из всех фрагментов данных можно было «собрать» исходные структуры данных.

Алгоритмы различных предметных областей могут фрагментироваться с различным качеством. В работе рассматривается фрагментированное представление алгоритмов предметной области «численное моделирование», для которой оно предоставляет ряд существенных преимуществ:

1. Возможность автоматической генерации параллельной программы. При фрагментированном подходе наиболее сложные зависимости между операциями скрываются внутри фрагментов кода, что позволяет формализовать процесс конструирования параллельной программы на основе схемы управления между фрагментами.
2. Возможность автоматически обеспечить параллельной программе ряд динамических свойств, в том числе настройку на доступные ресурсы вычислителя (количество процессоров/ядер, объем оперативной и кэш-памяти) и динамическую балансировку загрузки. Это можно сделать благодаря фрагментированной структуре программы.
3. Переносимость между различными архитектурами. Поскольку фрагментированная программа допускает свободу (в рамках заданного управления) в выборе порядка исполнения фрагментов, имеется существенный резерв для адаптации программы к конкретному вычислителю.
4. Возможность накапливать управляющие схемы решения задач в библиотеке. Во фрагментированном представлении управление полностью отдельно от вычислений, что позволяет накапливать в библиотеке не только вычислительные процедуры (фрагменты кода), но и управляющие схемы.

Фрагментированное представление алгоритма близко к блочному представлению, которое используется в высокопроизводительных библиотеках линейной алгебры (ATLAS [10], Plasma[11] и др.). Основная цель блочного представления – оптимизация выбранного численного алгоритма, достигаемая за счёт «дружественности» к кэш-памяти. Управление между блоками явно не выделяется, а сам подход не ориентирован на создание больших прикладных программ.

Размер фрагмента данных определяет общее количество фрагментов вычислений в программе и является важнейшим параметром фрагментированной программы, а потому заслуживает отдельного исследования. Понятно, что фрагментация должна быть достаточно мелкой, чтобы загрузить доступные ресурсы вычислителя, и в то же время достаточно крупной, чтобы расходы на управление оказались приемлемыми. Частично этот вопрос обсуждается в [12].

2.2 Асинхронная модель с массовыми операциями

Фрагментированное представление алгоритма может использоваться в различных моделях вычислений, однако наиболее подходящей является асинхронная модель с массовыми вычислениями [13], поскольку она представляет программу в виде множества *A*-блоков, что близко к представлению в виде множества фрагментов вычислений, а также полностью отделяет вычисления от управления.

Асинхронная программа (или *А-программа*) есть набор $P=(M,A,A_0)$, где *M – память*; *A* – конечное множество *А-блоков*, $A=\left\{A_k \middle| k\in\overline{1,n}\right\}$; A_0 – конечное множество готовых экземпляров.

Память *M* состоит из ячеек с неразрушающим чтением и записью, стирающей предыдущее содержимое ячеек. Выделяется информационная *IM* (используется для хранения данных решаемой задачи) и управляющая *CM* (используется для организации управления в программе) памяти: $M=IM\cup CM$. Информационная память состоит из множества $X=\{x,y,\ldots,z\}$ *простых* переменных и множества $Y=\{\bar{x},\bar{y},\ldots,\bar{z}\}$, разбитого на конечное число счетных, непересекающихся, линейно упорядоченных подмножеств, которые называются *массивами*. Элементы массива $\bar{x}=\{x_1,x_2,\ldots x_n\}$ называются *компонентами* $\bar{x}$; компонент $\bar{x}_i$ обозначается $\bar{x}[i]$. $X\cap Y=\emptyset$.

Каждый *А*-блок A_k образован четверкой (M_k,T_k,O_k,C_k), где M_k – подмножество памяти *M*, используемое *А*-блоком A_k (его входные, выходные и управляющие переменные); T_k – *спусковая функция* (или *триггер-функция*), представляющая собой предикат от переменных $x_1,x_2,\ldots x_m$, $x_i\in M_k$; O_k – *операция*, которая по значениям входных переменных $x_1,x_2,\ldots,x_m$ вычисляет значения выходных переменных $y_1,y_2,\ldots,y_n$, $x_i\in M_k$, $y_j\in M_k$; C_k – *управляющий оператор*, который по значениям входных переменных $z_1,z_2,\ldots,z_l$ вычисляет значения выходных переменных $z_1,z_2,\ldots,z_l$, $z_l\in M_k$ (каждая переменная z_i является входной и выходной одновременно).

Пусть *N* – множество натуральных чисел, $G\subseteq A x\mathrm{N}$. Элемент $(A_k,i)\in G$ обозначается A_k^i и называется *i*-м *экземпляром* *А*-блока A_k. Пусть $N_{A_k}=\left\{i\in\mathrm{N}\middle|(A_k,i)\in G\right\}$. *А*-блок A_k называется *простым*, если N_{A_k} – одноэлементное множество, и *массовым*, если N_{A_k} содержит более одного элемента. Множество N_{A_k} называется *областью применимости* *А*-блока A_k.

Каждому экземпляру A_k^i в управляющей памяти ставится в соответствие переменная логического типа, которую будем называть *признаком завершения* экземпляра A_k^i и обозначать $P(A_k^i)$. Истинность переменной $P(A_k^i)$ означает, что экземпляр A_k^i завершил исполнение. Если переменная ложна, то экземпляр может быть неготовым к исполнению, готовым к исполнению либо исполняться. В начальном состоянии памяти $P(A_k^i)$=ЛОЖЬ для всех A_k^i.

Вычисления по асинхронной программе организуются следующим образом:

1. В пустое множество готовых экземпляров A' включаются все экземпляры из множества A_0.
2. Из множества готовых экземпляров A' выбирается некоторое подмножество A'', экземпляры которого запускаются на исполнение. Исполнение экземпляра $A_k^i\in A''$ состоит в вычислении его операции O_k^i, а затем управляющего оператора C_k^i. Управляющий оператор присваивает признаку завершения $P(A_k^i)$ значение ИСТИНА и добавляет в множество A' экземпляры, готовые к исполнению.
3. Выполнение *А*-программы завершается, когда множество A' становится пустым и не существует исполняющихся экземпляров.

Если в этом определении рассматривать каждую переменную (простую либо компонент массива) как фрагмент данных, каждый *А*-блок – как фрагмент кода, а каждый экземпляр *А*-блока – как фрагмент вычислений, то представление алгоритма в асинхронной модели будет соответствовать фрагментированному представлению. При этом, однако, управление необходимо программировать вручную в управляющих операторах, что является рутинным процессом, ведущим к большому числу ошибок. Поэтому эту работу необходимо автоматизировать.

2.3 Алгоритм генерации управляющих операторов

Пусть задан частичный строгий порядок на множестве экземпляров фрагментов вычислений M. Сконструировать управляющий оператор для экземпляра фрагмента вычислений $s_i \in M$ можно по следующему алгоритму:

1. Генерируем команду, которая установит признак завершения для s_i в значение ИСТИНА.
2. Формируем множество T экземпляров фрагментов вычислений, зависящих от s_i: $T = \{t_j | s_i < t_j, t_j \in M\}$. В результате исполнения s_i каждый фрагмент вычислений t_j потенциально может стать готовым к исполнению.
3. Для каждого фрагмента вычислений t_j:
   - формируем множество U экземпляров фрагментов вычислений, которые должны исполниться до запуска t_j: $U = \{u_k | u_k < t_j, u_k \in M\}$.
   - генерируем команду создания переменной с именем $flag_j$ и присваиваем ей значение ИСТИНА.
   - для каждого элемента u_k генерируем команду, объединяющую значение признака завершения u_k со значением переменной $flag_j$ по принципу И.
   - генерируем набор команд, который:
     1) проверяет значение переменной $flag_j$;
     2) если оно истинно, проверяет значение триггер-функции t_j;
     3) если триггер-функция истина, добавляет фрагмент t_j в очередь готовых фрагментов.

3. Язык программирования Аспект

3.1 Ключевые особенности

Язык программирования Аспект [15] разработан на основе языка ОПАЛ (Описание Параллельных Алгоритмов) [16]. ОПАЛ был создан для описания задач на вычислительных моделях с массивами и содержит наиболее характерные способы задания массовых вычислений. Он позволяет формализовать некоторую предметную область, описав множество допустимых на ней алгоритмов, а затем формулировать задачи для этой предметной области. При этом алгоритм решения каждой задачи может быть синтезирован автоматически [13].

В отличие от ОПАЛ, язык Аспект позволяет задать представление конкретного алгоритма с необходимой степенью непроцедурности и частичным распределением ресурсов. Ключевыми особенностями языка являются:

1. Статическая типизация.
2. Явное описание зависимостей между операциями. В императивных языках на множестве операций задается линейный порядок, что ограничивает возможности параллельного исполнения программ. В отличие от них, Аспект позволяет на множестве операций задать частичный порядок.
3. Частичное распределение ресурсов. Как и в императивных языках, в Аспект допускается повторное присваивание значения переменной. Это означает, что в одной переменной программы могут находиться (в разное время) различные переменные алгоритма.
4. Ориентация на регулярные структуры данных. Язык программирования Аспект ориентирован на решение задач численного моделирования, поэтому основой организации вычислений являются массовые операции, позволяющие эффективно обрабатывать регулярные структуры данных, а основной представления данных – массивы.

5. Отсутствие средств описания вычислений. Аспект позволяет описать данные, операции, и взаимосвязи между ними. Для описания вычислений внутри операций предлагается использовать существующие языки программирования (в настоящее время поддерживается только C++).

Если в императивных языках все команды по умолчанию исполняются последовательно, а участки программы, пригодные для параллельного исполнения, необходимо явно указать, то в языке Аспект используется прямо противоположный подход: по умолчанию считается, что все фрагменты вычислений могут исполняться параллельно, а если необходим порядок (например, из-за зависимости по данным), то его необходимо явно указать.

Детальное описание языка Аспект выходит за рамки статьи, поэтому ниже поясняются только его основные конструкции, необходимые для понимания процесса разработки программ.

3.2 Структура программы и основные конструкции

3.2.1 Структура программы

Упрощённая структура Аспект-программы показана на рис. 2. Каждая Аспект-программа имеет уникальное имя и несколько разделов с объявлениями.

```
program <Имя программы>
preface {
   <Раздел объявлений внешнего языка>
};
data fragments
   <Раздел объявления фрагментов данных>
code fragments
   <Раздел объявления фрагментов кода>
task data
   <Раздел объявления данных задачи>
task computations
   <Раздел объявления вычислений задачи>
task control
   <Раздел описания управления>
end
```

Рис. 2. Структура Аспект-программы

Разделы «code fragments», «task data» и «task computations» являются обязательными. Остальные разделы могут быть опущены.

Раздел «preface» позволяет задать необходимые определения внешнего языка. Для C++ это объявление типов данных, объявление констант, подключение заголовочных файлов и др.

3.2.2 Разделы объявления фрагментов данных и данных задачи

Раздел объявления фрагментов данных состоит из набора строк, каждая из которых имеет следующий синтаксис:

<Тип внешнего языка> <Имя фрагмента данных>;

или

<Тип внешнего языка> <Имя фрагмента данных>[<Индекс1>][<Индекс2>]...[<ИндексN>];

где <Тип внешнего языка> – тип данных внешнего языка (встроенный или объявленный в секции «preface»), <Имя фрагмента данных> – произвольный идентификатор, <Индекс> – число либо константа, объявленная в разделе «preface». Определение первого вида позволяет задать простой фрагмент данных, а определение второго вида – фрагмент-массив.

Раздел объявления данных задачи состоит из набора строк, каждая из которых имеет следующий синтаксис:

<Имя фрагмента данных> <Имя переменной>;

или

<Имя фрагмента данных> <Имя переменной>[Индекс1][Индекс2]...[ИндексN];

Простые числовые типы данных внешнего языка (для C++ это int, float, double) являются фрагментами данных по умолчанию. Таким образом, все переменные (данные задачи) в языке Аспект состоят из фрагментов.

3.2.3 Разделы объявления фрагментов кода и вычислений задачи

Раздел объявления фрагментов кода позволяет задать один или более фрагментов кода, для чего используется следующий синтаксис:

<Имя фрагмента кода> (<Имя фрагмента данных> <Имя переменной>, ...) {
  <Вычисления на внешнем языке>
};

После имени фрагмента кода в скобках задаются имена переменных (параметров). Для каждой переменной указывается её тип. <Вычисления на внешнем языке> – это любой допустимый набор команд внешнего языка, который осуществляет вычисления на основе переданных параметров без побочных эффектов.

Раздел объявления вычислений задачи позволяет задать применение фрагментов кода к фрагментам данных и имеет следующий синтаксис:

<Имя фрагмента вычислений>: <Имя фрагмента кода> (<переменная1>, …, <переменнаяN>);

или

<Имя фрагмента вычислений>[<Индекс1>]...[<ИндексN>]: <Имя фрагмента кода> (<переменная1>, …, <переменнаяN>) where <Индекс1>: <нач. зн.> … <кон. зн.>, ... , <ИндексN>: <нач. зн.> … <кон. зн.>

где <Индекс> – это произвольный идентификатор, <Переменная> – это одна из переменных, объявленных в разделе «task data».

В первом случае задаётся простой фрагмент вычислений (однократное применение фрагмента кода к данным), а во втором случае – массовый фрагмент вычислений.

Каждый экземпляр массового фрагмента вычислений однозначно идентифицируется его индексами. Значения, которые может принимать каждый индекс, задаются после ключевого слова «where». Запись <Индекс>: <нач. зн.> … <кон. зн.> означает, что <Индекс> может принимать все целые значения начиная с <нач. зн.> и заканчивая <кон. зн.>.

3.2.4 Раздел описания управления

Управление в языке Аспект задается определением частичного порядка на множестве фрагментов вычислений. Выделяется два типа управления: между различными фрагментами вычислений и между экземплярами одного массового фрагмента вычислений. Если между фрагментами вычислений имеется информационная зависимость, управление является потоковым, иначе – прямым.

Для описания управления используется следующий синтаксис:

<Имя фрагмента вычислений1> < <Имя фрагмента вычислений2>;

где <Имя фрагмента вычислений1> – имя фрагмента вычислений, который должен выполниться первым, <Имя фрагмента вычислений2> – имя зависимого фрагмента вычислений. Если фрагмент вычислений является массовым, после его имени указываются индексы, идентифицирующие номер экземпляра.

Допускается задание сложного управления, например

((S1 & S2) | S3) < S4

означает, что фрагмент вычислений S4 может выполниться, либо если выполнился фрагмент вычислений S3, либо если выполнились оба фрагмента вычислений S1 и S2.

Все фрагменты вычислений, для которых порядок не задан, могут выполняться одновременно.

3.3 Пример Аспект-программы

В качестве примера на рис. 3 приведён текст Аспект-программы умножения матриц (фрагментация алгоритма рассмотрена в разделе 2.1). Инициализация данных и вывод результата опущены.

```
program MultMatrix
preface {
   const int M = 3;
   const int N = 3;
};
data fragments
   double Frg[M][M];
code fragments
   Mult(Frg X, Frg Y, Frg Z) {
      for(int i=0; i<M; ++i)
         for(int j=0; j<M; ++j)
            for(int k=0; k<M; ++k)
               Z[i][j] += X[i][k]*Y[k][j];
   };
task data
   Frg A[N][N];
   Frg B[N][N];
   Frg C[N][N];
task computations
   S[i][j][k]: Mult(A[i][k], B[k][j], C[i][j])
      where i: 0..N-1, j: 0..N-1, k: 0..N-1;
task control
   S[i][j][k] < S[i][j][k+1];
end
```

Рис. 3. Аспект-программа умножения матриц

В программе заданы матрицы A, B и C, состоящие из NxN элементов. Каждый элемент есть матрица размера MxM (фрагмент данных Frg). Кроме того, определён фрагмент кода Mult, который получает две матрицы (фрагменты данных X и Y) на вход и вычисляет их произведение (фрагмент данных Z) на выходе. Вычисления задаются на языке C++.

В секции «task computation» задан массовый фрагмент вычислений S с областью применимости (0..N-1)x(0..N-1)x(0..N-1), при этом имя S[i][j][k] обозначает соответствующий экземпляр S. Каждый экземпляр осуществляет вычисление фрагмента кода Mult с фрагментами данных, указанными для экземпляра в качестве фактических параметров.

В секции «task control» на множестве фрагментов вычислений задан такой частичный порядок, что каждый *k*-й фрагмент вычислений должен выполниться перед соответствующим (*k*+1)-м фрагментом. Управление отражает тот факт, что для вычисления C[i][j] необходимо просуммировать все матрицы, полученные в результате вычисления произведения фрагментов

A[i][k] на B[k][j] (это делается внутри фрагмента Mult добавлением результата умножения к старому значению C[i][j]), и для исключения «гонки» данных (data race) суммирование необходимо выполнять последовательно.

Все фрагменты вычислений S[i][j][k] с одинаковым индексом k могут исполняться одновременно.

4. Реализация системы программирования Аспект

4.1 Общая архитектура

Система асинхронного параллельного программирования Аспект [8-9] разрабатывается в Институте вычислительной математики и математической геофизики СО РАН. Она предназначена для решения задач численного моделирования на мультипроцессорных/многоядерных вычислителях.

Рис. 4. Разработка программы в системе Аспект

В основе системы Аспект лежит асинхронная модель вычислений, описанная в разделе 2.2. Система состоит из двух основных компонентов: транслятора и исполнительной подсистемы (рис. 4). На вход транслятору подается текст программы на языке Аспект, на выходе генерируется асинхронная программа на C++, которую необходимо скомпилировать вместе с исполнительной подсистемой компилятором C++ (например, g++ или icpc) в исполняемый файл.

4.2 Транслятор

Транслятор имеет стандартную архитектуру и состоит из лексического анализа, синтаксического анализа, семантического анализа, генератора внутреннего представления и генератора кода. На выходе он создаёт один *.cpp файл с асинхронной программой.

В начало файла записывается подключение необходимых библиотек, а также копируется раздел «preface» Аспект-программы. Затем генерируются класс с *A*-программой, после чего генерируются функции aplRun (запрашивает очередной *A*-блок из очереди готовых *A*-блоков и осуществляет его исполнение) и main (осуществляет инициализацию и запуск исполнительной подсистемы).

Исходная Аспект-программа преобразуется в одноимённый класс, наследуемый от виртуального класса AProgram, который предоставляет интерфейс для добавления *A*-блоков в очередь исполнения, а также осуществляет учёт необходимых параметров (количество исполняемых *A*-блоков, количество готовых *A*-блоков и др.).

Для каждого фрагмента данных создаётся отдельный тип, с использованием которого объявляются данные задачи (переменные класса). Для каждого фрагмента кода генерируется метод класса, а также две переменных: признак завершения и переменная синхронизации. Для триггер-функций и управляющих операторов генерируются соответствующие методы (тела управляющих операторов генерируется согласно алгоритму, описанному в разделе 2.3).

Также генерируется специальный метод aplMain, который выполняет инициализацию всех переменных программы и добавляет в очередь на исполнение *A*-блоки, не зависящие от других *A*-блоков.

4.3 Исполнительная подсистема

Исполнительная подсистема состоит из слоя абстрагирования от операционной системы, набора функциональных модулей (менеджер потоков, менеджер памяти, планировщик) и интерфейса системных вызовов.

Слой абстрагирования от ОС включается в себя все подпрограммы, в которых используются API операционной системы (определение доступных ресурсов, работу с потоками, функции для работы со временем и др.). Это позволяет переносить исполнительную подсистему из одной ОС в другую заменой лишь одного, выделенного слоя.

Менеджер потоков управляет распределением работы между процессорами (ядрами). При старте программы по умолчанию создается по одному потоку на каждый процессор (ядро). Каждый поток обращается за очередной порцией работы к планировщику.

Менеджер памяти управляет распределением памяти в системе, осуществляет её выделение/освобождение для очередей и *A*-программ.

Планировщик управляет набором очередей, каждая из которых имеет определённый приоритет и реализована в виде монитора. Алгоритм планирования имеет сложность O(1) и заключается в следующем: при каждом обращении очереди просматриваются в порядке убывания приоритетов; если в текущей очереди имеется готовый *A*-блок, он запускается на исполнение. Таким образом, *A*-блоки из очередей с более высоким приоритетом всегда будут запущены на исполнение раньше, чем *A*-блоки из очередей с более низким приоритетом. Программа завершается, когда все очереди становятся пустыми.

5. Результаты испытаний

Тестовые испытания проводились в Сибирском суперкомпьютерном центре Института вычислительной математики и математической геофизики СО РАН на системе HP ProLiant DL580 G5 со следующей конфигурацией:

- Аппаратное обеспечение: 4 процессора Intel Xeon X7350 (16 ядер), 256 Гбайт RAM, 9 Тбайт HDD.
- Программное обеспечение: Cent OS 5.3 64 bit, Intel C++ compiler professional for Linux 11.1.

Для тестирования были выбраны три задачи: умножение матриц, LU-разложение и явная разностная схема. Все задачи были реализованы на языке Аспект и на C++/OpenMP и компилировались с максимальной оптимизацией (ключ -O3) для архитектуры x86_64. Каждая задача с выбранными параметрами запускалась на исполнение 100 раз и в таблицу заносилось минимальное время исполнения в секундах.

Таблица 1. Результаты измерений производительности задачи умножения матриц

Реализация	Количество ядер				
	1	2	4	8	16
C++/OpenMP	460,21	245,66	156,53	105,14	61,22
Аспект	148,98	74,73	37,52	18,98	9,68

Все использованные матрицы – квадратные, размера 5040x5040. Элементы матриц – вещественные числа с двойной точностью (тип double), инициализировались случайными числами. Размер фрагментов данных в Аспект-реализации – 90x90.

Таблица 2. Результаты измерений производительности задачи LU-разложения

Реализация	Количество ядер				
	1	2	4	8	16
C++/OpenMP	267,14	139,10	90,53	67,27	67,29

Аспект	31,01	15,6	7,93	4,13	2,35

Фрагментированное представление алгоритма умножения матриц было реализовано по описанию раздела 2.1. Аналогичным образом фрагментировался алгоритм явной разностной схемы. Для алгоритма LU-разложения было разработано специальное фрагментированное представление [9].

Таблица 3. Результаты измерений производительности явной разностной схемы

Реализация	Количество ядер				
	1	2	4	8	16
C++/OpenMP	43,87	22,69	12,51	11,39	11,35
Аспект	40,68	21,76	12,24	11,27	11,18

Существенное превосходство Аспект-реализации над C++/OpenMP-реализацией в первых двух тестах объясняется фрагментированным представлением алгоритмов, за счёт которого достигается более эффективное использование кэш-памяти. По этой же причине фрагментированный код лучше масштабируется.

6. Заключение

В статье предложен подход к асинхронному программированию численных задач, основанный на фрагментированном представлении алгоритмов. Разработаны специализированная модель вычислений, являющаяся уточнением асинхронной модели с массовыми операциями и алгоритм автоматического конструирования управляющих операторов. На основе предложенной модели разработаны и реализованы язык и система асинхронного параллельного программирования Аспект.

Проведённое тестирование показало достаточно высокую эффективность системы и подтвердило основную научную гипотезу исследования: по высокоуровневому фрагментированному представлению алгоритма для выбранной предметной области возможна автоматическая генерация достаточно эффективных асинхронных параллельных программ.

Дальнейшие планы исследований связаны с расширением модели вычислений с целью поддержки архитектур с распределенной памятью, а также с реализацией методов динамической балансировки загрузки для таких архитектур.

Литература

1. Коновалов Н.А., Крюков В.А., Сазанов Ю.Л. C-DVM – язык разработки мобильных параллельных программ // Программирование. – 1999. – № 1. – С.46-55.
2. Lastovetsky A.L. Parallel Computing on Heterogeneous Networks. – John Wiley & Sons, 2003. – 350 p.
3. Moskovsky A., Roganov V., Abramov S. Parallelism granules aggregation with the T-system // 9th International Conference on Parallel Computing Technologies. LNCS 4671, pp. 293-302.
4. Андрианов А.Н. Система Норма: разработка, реализация и использование для решения задач математической физики на параллельных ЭВМ. Дис. д-ра техн. наук. 05.13.11. – Москва, 2001. – 158 с.
5. Kale L., Krishnan S. CHARM++ : A Portable Concurrent Object Oriented System Based On C++. Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications (Sept-Oct 1993). ACM Sigplan Notes, Vol. 28, No. 10, pp. 91-108.

6. Accelerated Library Framework for Cell Broadband Engine: [http://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/41838EDB5A15CCCD002573530063D465], 10.01.2010.
7. McCool M., Wadleigh K., Henderson B., Lin H. Performance evaluation of GPUs using the RapidMind development platform // ACM/IEEE Conference on Supercomputing, Tampa, Florida, Article No. 181. ACM, New York (2006).
8. Арыков С.Б., Малышкин В.Э. Система асинхронного параллельного программирования "Аспект" // Вычислительные методы и программирование. – 2008. – Т.9. – № 1. – С. 205-209.
9. Arykov S., Malyshkin V. Asynchronous Language and System of Numerical Algorithms Fragmented Programming // 10th International Conference on Parallel Computing Technologies (Novosibirsk, Russia, August 31-September 4, 2009). LNCS 5698, pp. 1-7.
10. Automatically Tuned Linear Algebra Software (ATLAS): [http://math-atlas.sourceforge.net/], 10.01.2010.
11. Buttari A., Langou J., Kurzak J., Dongarra J. A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures // Parallel Computing. – 2009. – Vol. 35, Issue 1. – P. 38-53.
12. Арыков С.Б., Малышкин В.Э. Алгоритмы конструирования асинхронных программ заданной степени непроцедурности методом группировки // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. – 2009. – Т. 7, вып. 1. – С. 3-15.
13. Вальковский В.А., Малышкин В.Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 129 с.
14. Арыков С.Б. Группировка данных в системе асинхронного параллельного программирования Аспект // Труды международной научной конференции «Параллельные вычислительные технологии (ПаВТ'2009)» (Нижний Новгород, 30 марта – 3 апреля 2009 г.). – Челябинск: ЮУрГУ, 2009. – С. 357-363.
15. Арыков С.Б. Язык программирования Аспект // Известия Томского политехнического университета. – 2008. – Т. 313. – № 5. – С. 89-92.
16. Малышкин В.Э. ОПАЛ – язык описания параллельных алгоритмов / В кн. Теоретические вопросы параллельного программирования и многопроцессорные ЭВМ. – Новосибирск: ВЦ СО АН СССР, 1983. – С. 91-109.

Параллельная реализация двумерных БИХ-фильтров в распределенной системе обработки изображений*

А.В. Никоноров, М.Г. Милюткин, В.А. Фурсов

Рассматривается информационная технология построения двумерных фильтров с бесконечной импульсной характеристикой (БИХ-фильтров). Для определения их характеристик используются малые тестовые фрагменты, которые формируются из искаженного изображения с использованием априорной информации о геометрической форме регистрируемых объектов. Двумерный БИХ-фильтр строится в виде параллельного соединения физически реализуемых фильтров с опорной областью в виде квадранта. Предложена архитектура распределенной вычислительной системы, в которой эксплуатируются параллелизм, как на уровне декомпозиции крупноформатного изображения, так и на уровне формирования структуры самого БИХ-фильтра.

1. Введение

В связи с созданием и развитием центров приема космической информации крайне востребованной является задача обработки крупноформатных изображений с целью улучшения их качества (например, резкости). Для этой цели обычно используются, либо фильтры с конечной импульсной характеристикой (КИХ-фильтры), либо БИХ-фильтры (с бесконечной импульсной характеристикой). КИХ-фильтры всегда физически реализуемы и устойчивы. Однако при значительных искажениях требуется опорная область больших размеров, что приводит к существенному возрастанию вычислительной сложности. Кроме того, возможности улучшения качества изображений с помощью КИХ-фильтров ограничены.

С использованием БИХ-фильтра можно обеспечить компенсацию сильных искажений с использованием опорной области небольших размеров. Однако эти фильтры для некоторых форм опорной области физически нереализуемы, кроме того, они могут оказаться неустойчивыми. Связано это с тем, что при радиально симметричных искажениях для определения выходного отсчета используются окружающие его отсчеты опорной области, некоторые из которых еще не вычислены. Указанные проблемы рассматривались в работах [1-4], в частности, использовалась итерационная схема реализации.

В настоящей работе для обработки крупноформатных изображений используется параллельное соединение физически реализуемых БИХ-фильтров с опорными областями в виде квадрантов. Реализация таких алгоритмов в распределенной вычислительной системе позволяет использовать декомпозицию задачи на двух уровнях. Первый уровень – декомпозиция по данным (разбиение крупноформатного изображения на фрагменты) между узлами вычислительной системы, второй уровень – функциональная декомпозиция потоков, соответствующих реализации отдельных БИХ-фильтров, каждый из которых реализован в «своем» квадранте опорной области.

2. Общая схема формирования двумерного БИХ-фильтра

Предполагается, что характеристики искажающей системы допускают построение восстанавливающих фильтров в классе линейных инвариантных к сдвигу систем. Известно, что для этого класса систем имеют место ассоциативные и дистрибутивные свойства, обеспечивающие возможность последовательного и параллельного соединения подсистем. В частности, применяя правило дистрибутивности для нахождения общего импульсного отклика для двух параллельно соединенных систем, имеем:

$$y = (x**h) + (x**g) = x**(h+g), \qquad (1)$$

* Работа выполнена при финансовой поддержке РФФИ (проект № 09-07-00269-а).

где x – общий входной сигнал, y – общий выходной сигнал, h, g – импульсные отклики систем, а (…**…) – означает операцию двумерной свертки. Опираясь на свойство (1) представим двумерный фильтр в виде параллельного соединения физически реализуемых БИХ-фильтров.

В общем случае выражение, определяющее способ вычисления выходного отсчета $y(n_1,n_2)$ двумерного БИХ-фильтра, имеет вид

$$y(n_1,n_2)=\sum_{r_1}\sum_{r_2}a(n_1-r_1,n_2-r_2)x(r_1,r_2)-\sum_{\substack{k_1\\(k_1k_2\neq\ n_1n_2)}}\sum_{k_2}b(n_1-k_1,n_2-k_2)y(k_1,k_2). \qquad (2)$$

Нетрудно заметить, что для вычисления значения конкретного выходного отсчета выходная маска должна накрывать только известные значения отсчетов (за исключением вычисляемого). На рисунке 1, а приведен пример опорной маски 5×5, для которой требование рекурсивной вычислимости не выполняется (вычисляемый выходной отсчет обозначен кружком в центре опорной области).

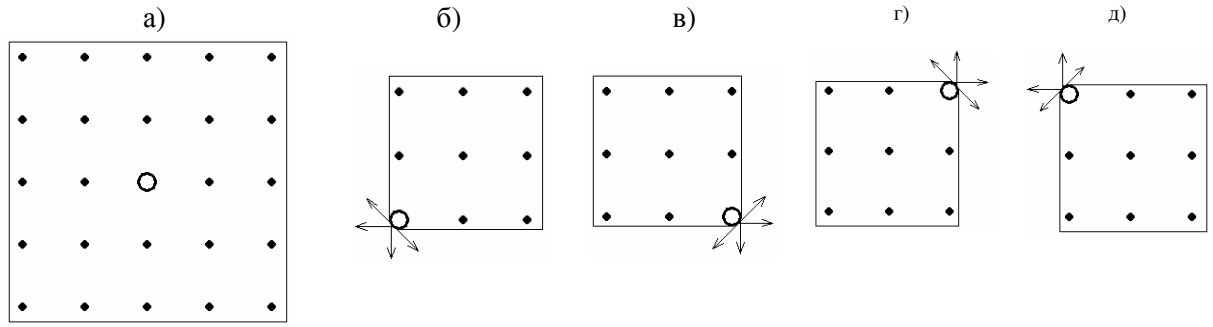


Рисунок 1 Исходная маска 5×5 – а) и соответствующие ей маски и допустимые направления рекурсии для: 1-го квадранта – б); 2-го квадранта – в); 3-го квадранта – г); 4-го квадранта – д).

Известно [5], что импульсный отклик симметричного БИХ-фильтра с опорной областью на всей (n_1,n_2)-плоскости можно разбить на четыре отдельных импульсных отклика, по одному на каждый квадрант. На рисунках 1, б – д приведены соответственно маски 1-го, 2-го, 3-го и 4-го квадрантов и направления рекурсии для них. С использованием указанного разбиения можно построить фильтры, соответствующие этим квадрантам и соединив их параллельно, получить общий импульсный отклик двумерного БИХ-фильтра. На рисунке 2 приведена общая схема параллельного соединения БИХ-фильтров одного квадранта, реализующая двумерный БИХ-фильтр на (n_1,n_2)-плоскости.

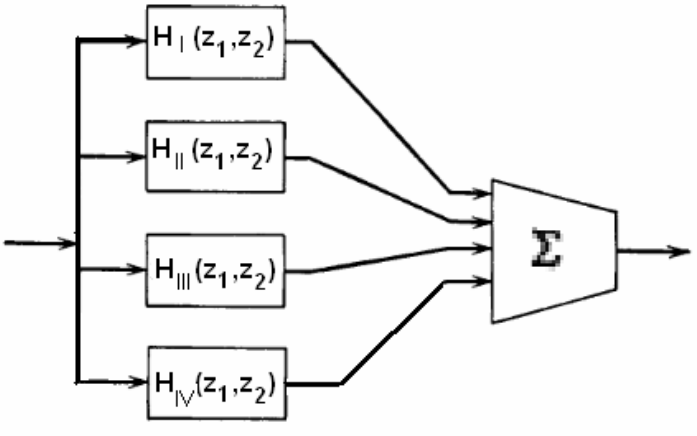


Рисунок 2 Параллельное соединение 4-х фильтров одного квадранта, реализующее двумерный БИХ-фильтр на (n_1,n_2)-плоскости.

Пусть построенные на каждом i-м квадранте БИХ-фильтры устойчивы и описываются передаточными функциями вида

$$H_z^{(i)}(z_1,z_2)=\frac{A_z^{(i)}(z_1,z_2)}{B_z^{(i)}(z_1,z_2)} \quad i=1,2,3,4. \qquad (3)$$

В соответствии со схемой (рисунок 2) составленную из них общую передаточную функцию двумерного БИХ-фильтр на (n_1, n_2)-плоскости можно записать в виде:

$$H_z^p(z_1, z_2) = \frac{A_z^p(z_1, z_2)}{B_z^p(z_1, z_2)} = \frac{\sum_{j=1}^{4} \prod_{i \neq j} A_z^{(j)}(z_1, z_2) B_z^{(i)}(z_1, z_2)}{\prod_{i=1}^{4} B_z^{(i)}(z_1, z_2)}. \quad (4)$$

Нетрудно заметить, что знаменатель общей передаточной функции является произведением знаменателей БИХ-фильтров на квадрантах.

В рамках описанной схемы формирования двумерного БИХ-фильтра наиболее сложная часть задачи заключается в построении устойчивых простых БИХ-фильтров на квадрантах. В следующем разделе описывается использовавшаяся в работах [1-5] и развиваемая здесь технология определения характеристик фильтров путем решения задачи идентификации.

3. Идентификация характеристик простых БИХ-фильтров

Задача построения БИХ-фильтров на квадрантах может быть решена путем идентификации его характеристик по заданным тестовым изображениям. Например, для настройки оптических систем фотоаппаратов часто используются так называемые миры [6]. Идентификация характеристик восстанавливающих БИХ-фильтров в этом случае может осуществляться по всему изображению. Построенный таким образом фильтр, компенсирующий искажения оптики, затем может использоваться для улучшения качества регистрируемых изображений в каждом эпизоде съемки.

Если ставится задача улучшения качества крупноформатных изображений, регистрируемых системами аэрокосмического мониторинга Земли, эталонное изображение, как правило, отсутствует. Тем не менее, можно сформировать «неискаженный» фрагмент изображения, используя априорные сведения о геометрической форме зарегистрированных объектов. Например, известно, что граница между крышей здания и отбрасываемой зданием тенью обязана быть резкой. Способ автоматизированного формирования резкой границы на малом фрагменте изображения впервые рассматривался в работе [7]. В последующих работах [1-4] этот способ неоднократно использовался в различных технологиях.

Для заданной опорной маски в виде квадранта можно выписать соотношение вида (2). Для некоторой пары тестовых фрагментов, один из которых сформирован путем компьютерного ретуширования резкой границы, можно записать систему линейных уравнений [8]:

$$\mathbf{Y} = \mathbf{S}\varphi + \xi, \quad (5)$$

где, в соответствии с (2)

$$\mathbf{Y} = \begin{bmatrix} y_1(n_1, n_2) \\ y_2(n_1, n_2) \\ \vdots \\ y_N(n_1, n_2) \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} x_1(r_1, r_2) & \cdots & y_1(k_1, k_2) & \cdots \\ x_2(r_1, r_2) & \cdots & y_2(k_1, k_2) & \cdots \\ \vdots & \vdots & \vdots & \vdots \\ x_N(r_1, r_2) & \cdots & y_N(k_1, k_2) & \cdots \end{bmatrix}, \quad \xi = \begin{bmatrix} \xi_1(n_1, n_2) \\ \xi_2(n_1, n_2) \\ \vdots \\ \xi_N(n_1, n_2) \end{bmatrix},$$

N – число отсчетов на тестовых фрагментах, по которым сформирована система (5),

$\varphi = \left[a(n_1 - r_1, n_2 - r_2), \ \cdots \ b(n_1 - k_1, n_2 - k_2), \ \cdots \right]^T$

- искомый вектор параметров фильтра.

Основная проблема при реализации указанной схемы идентификации состоит в том, что по фрагментам, имеющим перепад функции яркости в одном направлении, может быть идентифицирована передаточная функция невысокого порядка. В то же время, формирование заданной функции яркости для фрагментов, содержащих объекты сложной геометрической формы, представляется не всегда возможным. Связано это обычно, как с недостатком априорной информации о деталях изображения, так и трудностями отыскания резких границ объектов сложной формы. В настоящей работе для идентификации фильтра высокого порядка матрицу **S** в (5)

предлагается составлять из блочных матриц, каждая из которых формируется по простым фрагментам, на каждом из которых функция яркости изменяется в одном направлении, но эти направления для разных фрагментов различны. При этом обеспечивается хорошая обусловленность задачи при идентификации параметров фильтра достаточно высокого порядка.

Для вычисления параметров фильтра по данным системы (5) может использоваться простейшая оценка метода наименьших квадратов (МНК):

$$\hat{\Phi} = \left[\mathbf{S}^T\mathbf{S}\right]^{-1}\mathbf{S}^T\mathbf{Y}. \qquad (6)$$

В ряде случаев предпочтительнее более устойчивая к грубым ошибкам типа сбоев оценка метода наименьших модулей (МНМ-оценка).

Заметим, что идентификация восстанавливающего фильтра в классе БИХ-фильтров по соотношениям (5), (6) в вычислительном отношении существенно выгоднее по сравнению с использованием для этой цели КИХ-фильтров. Связано это с малыми размерами опорных областей. Малые размеры опорной области позволяют формировать тестовые фрагменты малых размеров, что важно с практической точки зрения.

4. Технология обработки цветных изображений

Если исходное искаженное цветное изображение задано в цветовом пространстве RGB, каждый отсчет представляется в виде вектора $(r \quad g \quad b)^T$. На первом этапе для каждого отсчета цветного изображения осуществляется переход от модели RGB к модели XYZ по формуле [8]:

$$\begin{pmatrix} \mathrm{x} \\ \mathrm{y} \\ \mathrm{z} \end{pmatrix} = \begin{pmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{pmatrix} \begin{pmatrix} \mathrm{r} \\ \mathrm{g} \\ \mathrm{b} \end{pmatrix}. \qquad (7)$$

Затем по соотношениям

$$\left.\begin{aligned} L &= 116 f(y/y_N) - 16, \\ a &= 500[(f(x/x_N) - f(y/y_N)], \\ b &= 200[f(y/y_N) - f(z/z_N)], \end{aligned}\right\} \qquad (8)$$

где $x_n = 96.422$, $y_n = 100$, $z_n = 82.521$,

$$f(t) = \begin{cases} t^{1/3} & \forall\, t > 0.008856, \\ 7,7867t + 16/116 & \forall\, t \le 0.008856 \end{cases} \qquad (9)$$

осуществляется переход к модели Lab.

Известно [5], что компонента *L* отвечает за уровень яркости, а две другие – за цветообразование, поэтому есть основания полагать, что указанным выше искажениям будет подвергаться именно *L*-компонента. Следовательно, описанную выше технологию формирования структуры и идентификации параметров простых БИХ-фильтров одного квадранта для улучшения качества цветных изображений достаточно применить к одной *L*-компоненте.

После того как сформирована *L*-компонента, обработанная параллельными простыми БИХ-фильтрами одного квадранта, осуществляется обратный переход от пространства *Lab* в пространство *RGB*. Сначала с использованием соотношения (8), (9) осуществляется обратный переход к координатам *x*,*y*,*z*, а затем к r, g, b:

$$\begin{pmatrix} \mathrm{r} \\ \mathrm{g} \\ \mathrm{b} \end{pmatrix} = \begin{pmatrix} 3.240479 & -1.537150 & -0.498535 \\ -0.969256 & 1.875992 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \qquad (10)$$

Преобразование (10) осуществляется для каждого отсчета изображения.

5. Архитектура распределенной вычислительной системы

Описанные алгоритмы обработки изображений обладают высокой степенью параллелизма. Этот факт иллюстрируется на примере рассматриваемой ниже реализации соответствующей информационной технологии, в виде распределенной вычислительной системы, подобной рассматривавшейся в работе [9].

Задача обработки изображений хорошо декомпозируется по данным, поэтому для обработки крупноформатного изображения применена простейшая клиент-серверная архитектура. Декомпозицию изображения на фрагменты, с учетом числа и производительности доступных вычислительных узлов, выполняет приложение-сервер, инициирующий распределенные вычисления. Вычисления выполняются приложениями-клиентами локальной сети, которые подключены к серверу. Схема приведена на рисунке 3.

Важной особенностью реализации алгоритмов фильтрации в данном случае является то, что наряду с декомпозицией по данным на каждом узле реализуется параллельная схема обработки четырьмя БИХ-фильтрами с опорной областью в виде одного квадранта. При этом для каждого элементарного БИХ-фильтра создается свой вычислительный поток, что позволяет эффективно использовать современные многоядерные (MultyCore) процессоры.

Каждый клиент в предложенной схеме выполняет вычисления для своей области данных, такой подход целесообразен для загрузки клиента в фоновом режиме. Приведенное на рисунке 1 разбиение на квадранты может быть использовано для многопоточной обработки одного и того же фрагмента изображения.

При таком подходе отдельный поток реализует фильтр одного квадранта. Разбиение, приведенное на рисунке 1, описывает 4 квадранта, что естественным образом реализуется четырьмя вычислительными потоками. В случае, когда число ядер на узле больше 4-х, например, 8 или 16 целесообразно использовать простые БИХ-фильтры с опорной областью в виде сектора. При этом, целесообразна структура фильтра, при которой число простых БИХ-фильтров одного сектора совпадает с числом ядер на узле (8 или 16). Описанная схема реализации вычислений на одном многопоточном узле позволит существенно снизить накладные расходы.

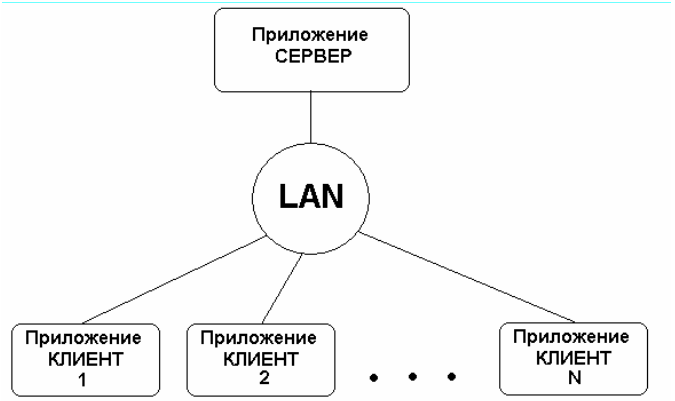


Рисунок 3 Схема распределенной системы

Повышение эффективности многопоточного подхода может быть достигнуто также за счет сочетания технологии Java и исполнения native-кода на каждом узле. Такой подход предполагает использование технологии Java JNI.

Таким образом, для более эффективной реализации БИХ фильтров в распределенной многопоточной среде целесообразно использовать два уровня декомпозиции и две технологии для распределения вычислений. Декомпозиция данных и Java для распределения вычислений по вычислительным узлам, и многопоточная JNI на каждом многоядерном вычислительном узле. С использованием JNI параллельная реализация простых БИХ-фильтров может быть выполнена в среде RDMA.

6. Результаты вычислительных экспериментов

Наиболее просто описанная технология определения характеристик БИХ-фильтров и обработки изображений реализуется в обычной схеме коррекции оптических искажений цифровых фотоаппаратов. Для этого может использоваться тестовое изображение – мира, например, показанная на рисунке 4. Для идентификации характеристик мира регистрируется фотоаппаратом, а

затем изображения исходной и зарегистрированной миры (рисунок 5) используются для формирования систем (5) для БИХ-фильтра каждого квадранта.

На рисунке 6 приведено полученное суммированием результатов обработки фильтрами, построенными путем идентификации параметров четырех БИХ-фильтров одного квадранта. Восстанавливалось приведенное на рисунке 5 изображение, полученное моделированием размытия значений отсчетов гауссианом с радиусом 5 с помощью графического пакета Paint .NET. Для восстановления использовалось параллельное соединение простых БИХ-фильтров одного квадранта, опорные области которых показаны на рисунках 1, (б-д).

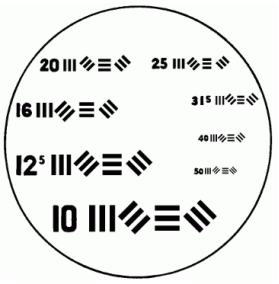


Рисунок 4 - исходное изображение

Рисунок 5 – искаженное изображение

Рисунок 6 – восстановленное изображение

На рисунках 7, а, б приведен другой пример применения описанной технологии к обработке черно-белых аэрокосмических снимков. Как и в предыдущем случае использовались четыре параллельно соединенных БИХ-фильтра одного квадранта с опорными областями, показанными на рисунках 1, б – д.

а) 

б) 

Рис. 7 – спутниковые черно-белые изображения: а) – исходное, б) – восстановленное

Для идентификации параметров фильтров одного квадранта использовались тестовые фрагменты малых размеров, показанные на рисунках 8, a – h, с изменением функции яркости в одном из 8-ми направлений. Соответствующие им «неискаженные» фрагменты, сформированные путем компьютерного ретуширования, показаны на рисунках 8, i – p. В соответствии с количеством пар тестовых фрагментов было сформировано 8 матричных блоков размерности 30×17, из которых затем составлена матрица **S** размером 240×17.

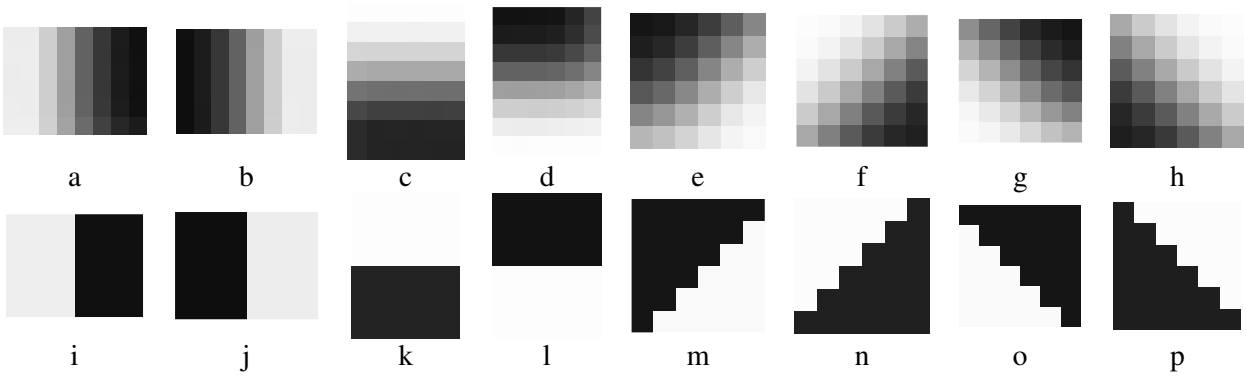


Рис. 8 Тестовые фрагменты: (a – h) – на исходном изображении,
(i – p) – те же фрагменты после компьютерного ретуширования.

На рисунках 9, а, б приведен пример обработки цветных аэрокосмических снимков. Использовалась технология, описанная в разделе 4. В качестве тестовых использовались фрагменты, аналогичные показанным на рисунке 8. Отличие состояло в том, что функции яркости на фрагментах формировались только для одного компонента цветного изображения - *L*.

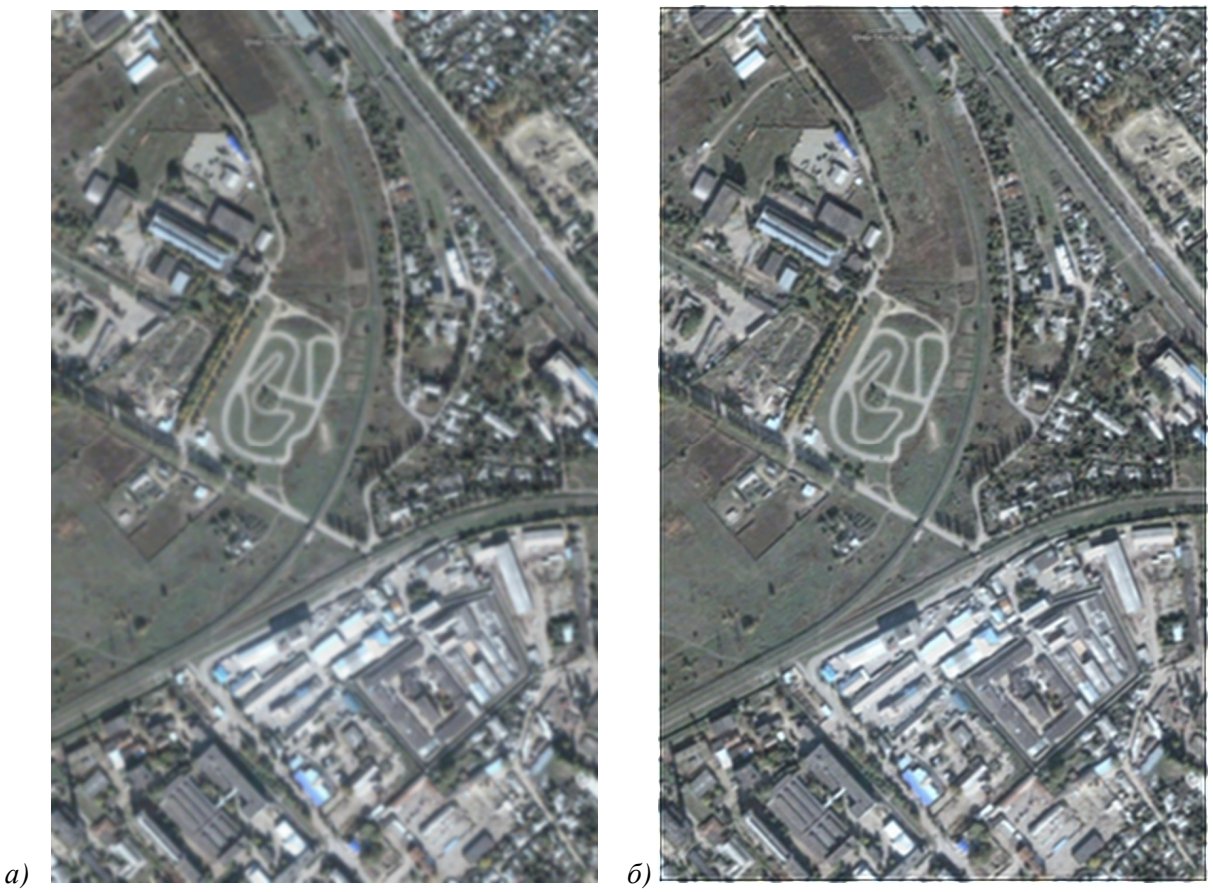


Рис. 9 – спутниковые цветные изображения: а) – исходное, б) – восстановленное

6. Заключение

Рассмотренный подход к построению процедур обработки крупноформатных изображений может служить примером того, как путем отказа от традиционного метода реализации алгоритмов обработки изображений (КИХ-фильтрами) удается существенно повысить эффективность параллельной вычислительной системы с многоядерными узлами. Рассмотренная в настоящей работе реализация в виде распределенной вычислительной системы связана с исполь-

зуемым способом распределенного хранения крупноформатных изображений. Описанный подход без существенных изменений может быть реализован также на кластере.

Наиболее интересными для дальнейших исследований являются направления, связанные с различными вариантами декомпозиции. При этом возможны различные схемы распределенной реализации, как на кластерных системах, так и на системах с общей памятью. Также представляет интерес анализ эффективности параллельной реализации простых БИХ-фильтров в RDMA системах.

Литература

1. Попов С.Б., Сойфер В.А., Тараканов А.А., Фурсов В.А. Кластерная технология формирования и параллельной фильтрации больших изображений // Компьютерная оптика. № 23, 2002, с. 75-78.
2. Дроздов М.А, Зимин Д.И, Скуратов С.А, Попов С.Б, Фурсов В.А. Технология определения восстанавливающих фильтров и обработки больших изображений // Компьютерная оптика. № 25, 2003.
3. Зимин Д.И., Фурсов В.А. Технология определения восстанавливающего фильтра и обработки цветных изображений // сб. Компьютерная оптика, № 27, 2005, с. 170-173.
4. Зимин Д.И., Фурсов В.А. Построение устойчивых алгоритмов обработки изображений путем аппроксимации фильтров с бесконечной импульсной характеристикой // сб. Компьютерная оптика, № 28, 2005, с. 124-127.
5. Dan E. Dudgeon, Russell M. Mersereau, Multidimensional digital signal processing, Prentice-Hall, Inc., Englewood Cliffs, 1984.
6. http://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0_(%D0%BE%D0%BF%D1%82%D0%B8%D0%BA%D0%B0)
7. Sergeyev, Vladislav V., Fursov, Vladimir A., & Parfyonov C. I. Information Technologi for Evaluating the Resolving Power of the Video Channel with the Use of a Model Filter with Infinite Impulse Response. Pattern recognition and image analysis, Vol. 9, No.2, 1999, p. 314-316.
8. Методы компьютерной обработки изображений / Под ред. Сойфера В.А., Москва, Физматлит, 2001.
9. Никоноров А.В., Фурсов В.А. Распределенная вычислительная среда коррекции цветных изображений. Труды XV Всероссийской научно-методической конференции "Телематика 2008", С.-Петербург, 23-26 июня, 2008, с 88-89.

Численное моделирование трехмерных течений с волнами детонации на многопроцессорной вычислительной технике*

И.В. Семенов, И.Ф. Ахмедьянов, П.С. Уткин, А.Ю. Лебедева

В работе представлены математические модели, численные методы и методика распараллеливания расчетного алгоритма для решения задач инициирования и распространения волн газовой детонации в трехмерных трубах сложной формы. Проанализированы механизмы инициирования детонации в осесимметричной трубе с параболическим сужением и коническим расширением, а также в винтовой трубе. Полученные результаты имеют как фундаментальное значение, раскрывая механизм инициирования детонации в трехмерных трубах с профилированными стенками, так представляют и практический интерес, поскольку указывают путь снижения энергозатрат на инициирование детонации в импульсных тепловых машинах.

1. Введение

Детонация – это гидродинамический волновой процесс распространения по веществу зоны экзотермической реакции со сверхзвуковой скоростью. Детонационная волна (ДВ) представляет собой самоподдерживающийся ударный разрыв (головную ударную волну), за фронтом которого непрерывно инициируется химическая реакция вследствие нагрева при адиабатическом сжатии. Другими словами детонация – это сверхзвуковой режим распространения горения.

Как известно, реальная детонация в газах сопровождается образованием сложной неодномерной и нестационарной структуры течения за ее передним фронтом [1]. В частности, с 1926 года известно явление спиновой детонации, суть которого заключается в том, что в трубах вблизи детонационных пределов наиболее яркое свечение фронта волны сосредотачивается у стенки – в «голове» спина, вращающейся по окружности одновременно с поступательным движением фронта [2]. Спиновый режим впервые наблюдался С. Кемпбеллом, Д.В. Вудхедом и А.С. Финчем как периодические неоднородности на фоторазвертках самосвечения детонации в смесях окиси углерода с кислородом. Другим примером являются экспериментально обнаруженные в 1957 – 1958 гг. ячеистые структуры детонационного фронта вдали от пределов, обусловленные наличием сильных неоднородностей – поперечных волн в зоне за лидирующим скачком [2].

С подобным многообразием проявлений природы детонационных процессов в газовых реагирующих смесях связан ряд сложностей проведения как натурных, так и вычислительных экспериментов. Для экспериментального исследования детонационных явлений требуется разработка прецизионных методов измерений различных газодинамических величин, и часто эти методы сами по себе являются предметом серьезного научного исследования.

Настоящая работа посвящена численному исследованию одного из многочисленных вопросов механики быстропротекающих процессов – возможности инициирования детонации в газовых смесях при минимальных затратах энергии на коротких расстояниях и за малое время. Основная идея оптимизации инициирования заключается в специальной профилировке стенок трубы с тем, чтобы при прохождении относительно слабой ударной волны (УВ) по трубе обеспечить возникновение локальных областей самовоспламенения взрывчатой смеси и формирование затем самоподдерживающегося детонационного режима. Конечная цель – перевести в детонацию УВ с минимальной интенсивностью.

Исследование проводится в рамках идеологии вычислительного эксперимента [3] на многопроцессорных ЭВМ с распределенной памятью с использованием разработанного авторами программного комплекса для численного исследования динамики потоков реагирующих многокомпонентных сред в многомерных областях. В [4] изложены модели, численный метод и алгоритм распараллеливания для решения двумерных осесимметричных задач, связанных с

* Работа выполнена в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» (конкурс НК-100П, контракт № П-359).

инициированием и распространением волн газовой детонации. Данная работа служит дальнейшим развитием и расширением исследования [4] для трехмерного случая.

2. Математическая модель трехмерных течений реагирующих газовых смесей

Исследование проводится с использованием системы уравнений, описывающей трехмерные нестационарные течения невязкой сжимаемой многокомпонентной реагирующей газовой смеси, которая в декартовой системе координат (x, y, z) имеет следующий вид:

$$\frac{\partial \mathbf{q}}{\partial t}+\frac{\partial \mathbf{f}_1}{\partial x}+\frac{\partial \mathbf{f}_2}{\partial y}+\frac{\partial \mathbf{f}_3}{\partial z}=\mathbf{S},$$

$$\mathbf{q}=\begin{bmatrix}\rho_1\\ \dots\\ \rho_N\\ \rho U_x\\ \rho U_y\\ \rho U_z\\ \rho E\end{bmatrix},\ \mathbf{f}_1=\begin{bmatrix}\rho_1 U_x\\ \dots\\ \rho_N U_x\\ \rho U_x^2+p\\ \rho U_y U_x\\ \rho U_z U_x\\ (\rho E+p)U_x\end{bmatrix},\ \mathbf{f}_2=\begin{bmatrix}\rho_1 U_y\\ \dots\\ \rho_N U_y\\ \rho U_x U_y\\ \rho U_y^2+p\\ \rho U_z U_y\\ (\rho E+p)U_y\end{bmatrix},\ \mathbf{f}_3=\begin{bmatrix}\rho_1 U_z\\ \dots\\ \rho_N U_z\\ \rho U_x U_z\\ \rho U_y U_z\\ \rho U_z^2+p\\ (\rho E+p)U_z\end{bmatrix},\ \mathbf{S}=\begin{bmatrix}\dot\omega_1\\ \dots\\ \dot\omega_N\\ 0\\ 0\\ 0\\ \dot q\end{bmatrix}. \tag{1}$$

Здесь t – время; U_x, U_y и U_z – составляющие скорости; ρ, p и E – плотность, давление и полная удельная энергия газовой смеси соответственно; ρ_i, $\dot\omega_i$ – плотность и скорость изменения плотности i-го компонента смеси в результате химических реакций; $\dot q$ – тепловой эффект химических реакций. Полная удельная энергия газовой смеси определяется как:

$$E=\frac{U_x^2+U_y^2+U_z^2}{2}+e, \tag{2}$$

где e – внутренняя удельная энергия газовой смеси.

В качестве термического и калорического уравнений состояния смеси, рассматриваемой как совершенный газ, используются следующие соотношения:

$$p=\sum_{i=1}^{N}\rho_i\frac{R}{\mu_i}T,\ e=\sum_{i=1}^{N}\frac{\rho_i}{\rho}e_i,\ e_i=c_{Vi}T,\ i=1,...,N, \tag{3}$$

где T – температура; R – универсальная газовая постоянная; e_i, c_{Vi} и μ_i – удельная внутренняя энергия, удельная теплоемкость при постоянном объеме и молярная масса i-го компонента смеси; N – число компонентов смеси.

Химические реакции моделируются одностадийной кинетикой горения пропана [5]:

$$\mathrm{C_3H_8}+5\mathrm{O_2}+5\cdot 3.762\mathrm{N_2}\rightarrow 3\mathrm{CO_2}+4\mathrm{H_2O}+5\cdot 3.762\mathrm{N_2}+Q, \tag{4}$$

где Q = 46.6·10^6 Дж/кг – теплота сгорания, рассчитанная на единицу массы топлива.

Таким образом, число компонентов N газовой смеси равно 5. Им приписываются индексы в соответствии со схемой: $\mathrm{C_3H_8}$ ($i=1$), $\mathrm{O_2}$ ($i=2$), $\mathrm{N_2}$ ($i=3$), $\mathrm{CO_2}$ ($i=4$), $\mathrm{H_2O}$ ($i=5$).

Скорость изменения плотности пропана $\dot\omega_1$ определяется как:

$$\dot\omega_1=\mu_1\dot\omega_1^{mole}=\mu_1\left(k\frac{\rho_1}{\mu_1}\cdot\frac{\rho_2}{\mu_2}\right), \tag{5}$$

$$k=-7\cdot 10^8\,p^{-0.2264}\exp\left(-E^*/RT\right)\ \text{м}^3/(\text{моль}\cdot\text{с}),\ E^*=190{,}3\cdot 10^3\ \text{Дж/моль},$$

где $\dot{\omega}_1^{mole}$ – скорость изменения его молярной концентрации, p – давление в атмосферах, T – температура в градусах Кельвина. Скорости изменения плотностей остальных компонентов смеси определяются через $\dot{\omega}_1^{mole}$ и стехиометрические коэффициенты в реакции (4):

$$\dot{\omega}_2 = 5\mu_2\dot{\omega}_1^{mole},\ \dot{\omega}_3 = 0,\ \dot{\omega}_4 = -3\mu_4\dot{\omega}_1^{mole},\ \dot{\omega}_5 = -4\mu_5\dot{\omega}_1^{mole}. \quad (6)$$

Тепловой эффект реакции определяется как:

$$\dot{q} = -Q\dot{\omega}_1. \quad (7)$$

3. Численный метод

Для решения задачи (1) – (3), (5) – (7) используется метод расщепления по физическим процессам. На шаге по времени n в текущей расчетной ячейке i сначала решается система уравнений газовой динамики (1) без учета члена $\mathbf{S}$ в правой части. Затем полученное решение $\mathbf{q}_i^{газ,n+1}$ корректируется с учетом источниковых членов, связанных с протекающими химическими реакциями.

Для дискретизации системы уравнений газовой динамики по пространственным переменным используется метод конечных объемов (МКО). Основными преимуществами класса подобных схем являются их вычислительная робастность, консервативность, а также возможность работы на неструктурированных сетках [6]. Используется, так называемая, центрированная схема МКО, когда конечные объемы совпадают с ячейками расчетной сетки. Второй порядок точности на гладких решениях достигается путем сочетания использования кусочно-линейной аппроксимации величин внутри ячейки с простейшим двухшаговым пересчетом по времени, который также называется предиктор-корректор [7]. Потоки вычисляются на основе точного решения задачи Римана о распаде произвольного разрыва методом типа Годунова [8]. При расчете градиента $\nabla\mathbf{q}_i^n$ вектора-решения $\mathbf{q}_i^n$ в текущей расчетной ячейке i на текущем шаге по времени n для экстраполяции значений газодинамических параметров из центра ячейки на грани применяется метод наименьших квадратов. Будем обозначать вектор экстраполированных величин на грань σ как $\mathbf{q}_{i,\sigma}^n$. Шаг интегрирования по времени Δt определяется динамически в процессе вычислений для выполнения условия устойчивости [8].

Этап предиктора может быть записан в следующем виде:

$$\tilde{\mathbf{q}}_i^{n+1} = \mathbf{q}_i^n - \frac{\Delta t}{2V_i}\sum_\sigma \tilde{\mathbf{F}}_{i,\sigma}^n \cdot s_{i,\sigma},$$

где V_i – объем текущей ячейки, суммирование ведется по всем граням σ ячейки, $s_{i,\sigma}$ – площадь соответствующей грани, $\tilde{\mathbf{F}}_{i,\sigma}^n$ – проекция потока через грань σ на ее внешнюю нормаль:

$$\tilde{\mathbf{F}}_{i,\sigma}^n = T_\sigma^{-1}\mathbf{f}_{i,\sigma}^n,\ \mathbf{f}_{i,\sigma}^n = \mathbf{f}_1\left(\mathbf{Q}_{i,\sigma}^n\right),\ \mathbf{Q}_{i,\sigma}^n = T_\sigma\mathbf{q}_{i,\sigma}^n,$$

где T_σ – матрица преобразования от лабораторной системы координат к локальной системе координат, один орт которой ориентирован по внешней нормали, а другие лежат в плоскости грани σ.

Затем следует этап корректора:

$$\mathbf{q}_i^{газ,n+1} = \mathbf{q}_i^n - \frac{\Delta t}{V_i}\sum_\sigma \mathbf{F}_{i,\sigma}^n \cdot s_{i,\sigma},$$

где

$$\mathbf{F}_{i,\sigma}^n = T_\sigma^{-1}\tilde{\mathbf{f}}_{i,\sigma}^{n+1},\ \tilde{\mathbf{f}}_{i,\sigma}^{n+1} = \mathbf{f}_1\left(\tilde{\mathbf{Q}}_{i,\sigma}^{n+1}\right),$$

а вектор $\tilde{\mathbf{Q}}_{i,\sigma}^{n+1}$ – решение задачи Римана о распаде произвольного разрыва, соответствующее начальным данным $\tilde{\mathbf{q}}_{i,\sigma}^{n+1}$ и $\tilde{\mathbf{q}}_{j,\sigma}^{n+1}$ по разные стороны от разрыва, определяемого гранью σ. Здесь $\tilde{\mathbf{q}}_{i,\sigma}^{n+1}$ обозначает вектор экстраполированных величин на грань σ, рассчитанный по значениям $\tilde{\mathbf{q}}_{i}^{n+1}$ с шага предиктора и градиенту $\nabla\mathbf{q}_{i}^{n}$, а индекс j соответствует параметрам в ячейке, граничащей по грани σ с рассматриваемой ячейкой i.

На следующем этапе производится учет источниковых членов в правой части (1), обусловленных химическими реакциями:

$$\frac{d\mathbf{q}_i^{хим}(t)}{dt} = \mathbf{S},\ t \in [t_n,\ t_{n+1}],\ \mathbf{q}_i^{хим}(t_n) = \mathbf{q}_i^{газ,n+1}. \tag{8}$$

Здесь t_n – текущее время. В качестве начального условия для системы обыкновенных дифференциальных уравнений (8) берется решение, полученное на предыдущем газодинамическом этапе. Для решения системы ОДУ используется многошаговый неявный метод Гира. Построенное решение, взятое в момент времени t_{n+1}, будет решением всей задачи на следующем шаге по времени:

$$\mathbf{q}_i^{n+1} = \mathbf{q}_i^{хим}(t_{n+1}).$$

4. Распараллеливание расчетного алгоритма и сетки

В основе распараллеливания расчетного алгоритма лежит метод декомпозиции расчетной области. В [4] был выполнен анализ эффективности распараллеливания расчетного алгоритма для решения двумерных задач в случае использования структурированных сеток, исходя из особенностей рассматриваемых задач и характеристик вычислительной техники. В частности, рассматривались различные варианты одномерной декомпозиции расчетной области вдоль одного из координатных направлений применительно к задаче распространения УВ в реагирующей смеси.

В трехмерных расчетах используются неструктурированные сетки с гексаэдрическими ячейками, которые в общем случае могут представлять собой произвольные объединения выпуклых шестигранников, пересекающихся по граням, ребрам или в узлах. Это накладывает определенные трудности при реализации декомпозиции области, поскольку ячейки сетки, находящиеся близко друг к другу в геометрическом смысле могут быть сильно разнесены в топологическом смысле и с точки зрения организации структуры данных для хранения сетки. Для осуществления декомпозиции области был использован свободно распространяемый пакет программ METIS для разбиения графов на подграфы [9]. Стоит отметить, что пакет METIS широко используется в зарубежных программных комплексах для решения задач механики сплошной среды, таких как ANSYS, для осуществления декомпозиции расчетной области. Оценки эффективности разбиения сетки с помощью METIS по сравнению с аналогами (JOSTLE [10], PARTY [11], SCOTCH [12]) оказываются намного лучше.

На Рис. 1 приведен пример используемой расчетной O-сетки и ее разбиения с помощью METIS на K = 10 частей для трубы с параболическим сужением и коническим расширением, процесс инициирования детонации в которой будет рассмотрен далее. Рассмотрим количественные характеристики качества разбиения сетки с помощью METIS. Размер тестовой сетки, представленной на Рис. 1, составляет 2 929 360 ячеек, общее число граней – 8 734 740. В результате разбиения отклонение числа ячеек во всех частях от равномерного 2 929 360 / K не превышает 2%. При этом общее количество граней, по которым граничат ячейки из разных частей, составляет 98 693. Оценим минимальное число граней, по которым будут граничить ячейки из различных частей, в случае одномерной декомпозиции и использования структурированной сетки. Число ячеек в поперечном сечении составляет примерно 10 000, значит количество граничных ячеек при разбиении на 10 частей – около 90 000, так что фактическое число граничных ячеек отличается от оценочного минимального для одномерной декомпозиции менее чем на 10%.

Для расчетов используются от 200 до 600 процессорных ядер СК МВС-100k Межведомственного суперкомпьютерного центра РАН или СК СКИФ МГУ «Чебышев». Характерное пространственное разрешение процессов составляет 0.3 мм, временное – 5 – 10 нс.

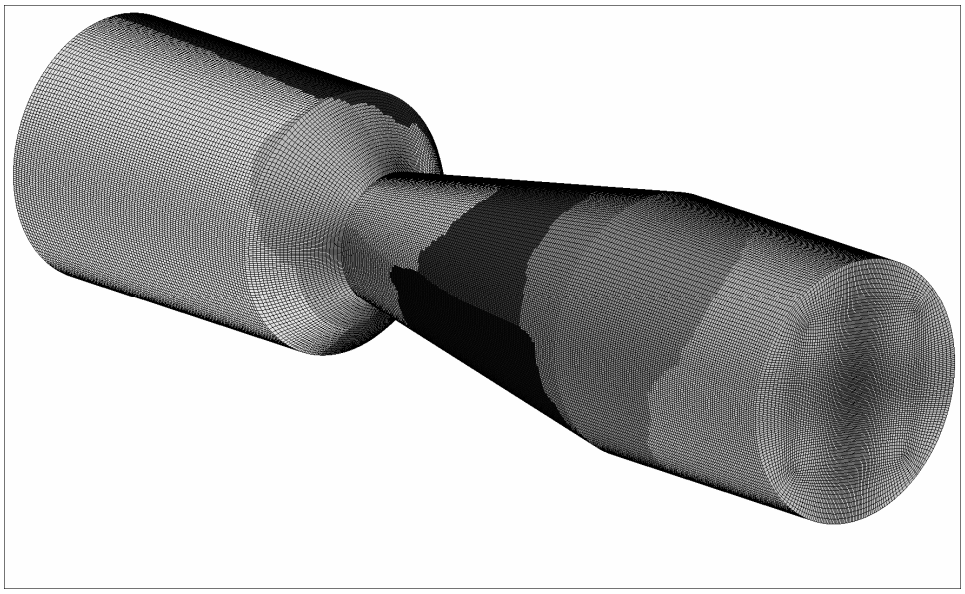


Рис. 1. Пример расчетной О-сетки и ее разбиения с помощью METIS на 10 частей.

На Рис. 2 представлена зависимость ускорения от числа используемых процессорных ядер при решении тестовой задачи на сетке, аналогичной по геометрическим параметрам изображенной на Рис. 1, но с числом ячеек порядка 16 млн. Стоит отметить, что при использовании 190 процессорных ядер ускорение составляет 94, а эффективность, соответственно, около 50%, что является приемлемым с точки зрения рационального использования вычислительных ресурсов результатом. Тем не менее, требуется дальнейшая работа по оптимизации как схемы межпроцессорных обменов, так и самого программного кода для повышения эффективности распараллеливания. Характерное время решения тестовой задачи (~ 20 000 шагов по времени) на 190 ядрах составляет примерно 15 часов.

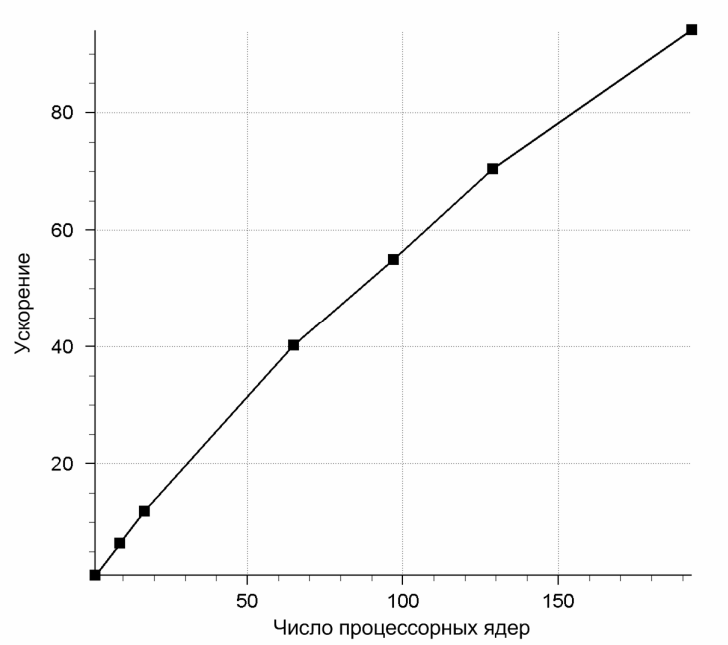


Рис. 2. Зависимость ускорения от числа процессорных ядер при решении тестовой задачи.

5. Способы инициирования детонации

Исследование механизмов формирования самоподдерживающейся детонации при различных способах инициирования является важной фундаментальной задачей, а также является неотъемлемым этапом разработки устройств, основанных на детонационном сжигании топлива.

Рассматривая вопрос об инициировании детонации, в первую очередь, необходимо проанализировать существующие принципиальные способы, с помощью которых оно может осуществляться. От этого зависит как качественная картина газодинамического течения в трубе, так и количественные характеристики процесса, такие как величина подведенной энергии, достаточной для инициирования, расстояние и время инициирования. Далее приведен краткий обзор различных подходов к проблеме инициирования детонации, а также некоторые результаты теоретических и экспериментальных исследований механизмов инициирования, реализующихся в этих подходах.

При детонации компоненты смеси вступают в реакцию в результате сжатия и нагрева газа в УВ. Поэтому естественный, наиболее простой и быстрый способ инициировать детонацию заключается в том, чтобы создать во взрывчатой смеси достаточно мощную УВ, способную вызвать практически мгновенное воспламенение смеси [13]. Мощная УВ, инициирующая интенсивные экзотермические реакции, возникает при сильном плоском, цилиндрическом или сферическом взрыве в реагирующей газовой смеси [14]. Такой способ инициирования детонации принято называть прямым инициированием или инициированием с помощью концентрированного подвода энергии. ДВ, образующаяся в результате взрыва является пересжатой. Численные и экспериментальные исследования показывают, что пересжатая ДВ переходит к самоподдерживающейся детонации Чепмена-Жуге (ЧЖ) на конечном расстоянии от места взрыва, составляющем несколько десятков поперечных размеров детонационных ячеек. Это проявляется не только в уменьшении скорости ДВ, но и в развивающейся неустойчивости фронта и появлении системы поперечных волн, типичных для самоподдерживающейся детонации в газе [1]. Важным параметром является критическая энергия взрыва, ниже которой самоподдерживающийся детонационный режим не формируется [14]. Существование критической энергии связано с конечной скоростью протекания химических реакций в сжатой и нагретой ударной волной смеси. Эти реакции включают в себя процессы диссоциации с образованием активных центров, протекающие во время, так называемого, периода индукции без выделения тепла и даже с поглощением тепловой энергии среды, и последующие экзотермические реакции рекомбинации, при которых образуются продукты детонации. Прямое экспериментальное определение критических условий инициирования и распространения детонации могло бы ответить на массу вопросов, возникающих у проектировщиков силовых установок и инженеров по технике безопасности, но это непозволительно дорогое решение проблемы. В этом заключается одна из причин, по которой для предсказания поведения горючих смесей в конкретных условиях, представляющих интерес, исследователи концентрируют усилия, в основном, на численном моделировании и полуэмпирических методах [15].

Рассмотрим теперь, по какому сценарию будет развиваться горение в трубе, если взрывчатую смесь поджечь, не создавая УВ, т.е. сравнительно медленно подводя тепло, например, посредством разогретой проволоки или слабой электрической искрой. При этом смесь начинает гореть, возникает нормальное распространение пламени, при котором от слоя к слою горение передается теплопроводностью, нагревающей несгоревший газ, и диффузией, поставляющей в него химически активные частицы из горящего слоя. Явление перехода горения в детонацию (ПГД) заключается в том, что при распространении пламени в длинной трубе скорость его постепенно увеличивается, и на некотором расстоянии от места зажигания возникает ДВ. Преддетонационное расстояние зависит не только от состояния исходной смеси, но и от гидродинамических условий, при которых происходит распространение пламени, от диаметра трубы, состояния стенок (гладкие, шероховатые) и т.д. [13]. Как отмечается многими исследователями [16], механизм ПГД – одна из основных нерешенных проблем современной теории горения и детонации. Связано это, прежде всего, с тем, что качественное предсказание ПГД в реагирующих газах – очень сложная научная задача, которая требует знаний о динамике взаимодействия пламен, ударных волн, пограничных слоев, турбулентности и т.д. При этом с точки зрения практических приложений, нежелательный, неконтролируемый ПГД имеет огромный разруши-

тельный потенциал. Классический механизм ПГД в прямой трубе включает несколько стадий, а именно: 1) вынужденное зажигание смеси с образованием ламинарного пламени, 2) прогрессирующее увеличение скорости горения вследствие проявления неустойчивости и последующей турбулизации течения перед фронтом пламени, 3) образование и усиление УВ перед ускоряющимся фронтом пламени и 4) самовоспламенение ударно-сжатой смеси в области между УВ и фронтом пламени (образование «горячих точек» и «взрыв во взрыве»), приводящее к образованию пересжатой ДВ, а затем и 5) самоподдерживающейся детонации ЧЖ [17].

Другой подход к инициированию детонации заключается в том, чтобы на конечном этапе ПГД, когда перед пламенем уже сформировалась УВ, воздействовать на эту УВ внешними факторами и перевести ее в детонационную, не ожидая, пока сработает механизм образования «горячих точек». В [18] для этих целей было предложено использовать электрические разряды принудительного зажигания, распределенные по длине трубы. Тщательная синхронизация разрядов с моментами прохождения УВ обеспечивает инициирование детонации в трубах с гладкими стенками на коротких расстояниях без дополнительной турбулизации потока. При этом суммарная энергия, выделенная в разрядах существенно ниже, чем энергия прямого инициирования с помощью одного разряда. В [19] была исследована возможность перехода ударной волны в детонационную за счет профилировки стенок канала. В численных экспериментах и натурных опытах было продемонстрировано, что регулярный параболический профиль стенок плоского канала может существенно сократить время перехода ударной волны в детонационную. В настоящей работе представлено развитие идеи инициирования детонации за счет профилировки стенок трехмерной трубы.

6. Инициирование детонации в трубе с параболическим сужением и коническим расширением

В [20] на основе вычислительного эксперимента в осесимметричной постановке с помощью программного комплекса [4] был выявлен механизм инициирования детонации в трубе с параболическим сужением и коническим расширением, а также обнаружена «оптимальная» форма параболического сужения, обеспечивающая инициирование детонации для минимального числа Маха инициирующей УВ 2.65 для блокировки трубы 0.75. Под блокировкой трубы понимается $BR = 1 - (d / D)^2$, где D – диаметр максимального по площади сечения трубы, d – минимального. В качестве топлива использовалась стехиометрическая пропано-воздушная смесь. В дальнейшем результаты этих вычислительных экспериментов были подтверждены в натурных опытах [21]. Для «оптимальной» формы параболического сужения из [20] и угла раствора конуса расширительной секции 20° было продемонстрировано, что существует критическое значение числа Маха инициирующей УВ 2.85, выше которого наблюдается переход ударной волны в детонационную, а ниже – нет. Для оптимизации процесса инициирования в экспериментальной работе [22] была существенно увеличена длина расширяющейся конической секции. При этом критическое число Маха инициирующей УВ снизилось до величины 2.0.

В настоящей работе представлены результаты трехмерного численного исследования процесса инициирования газовой детонации в трубе с параболическим сужением и коническим расширением.

Рассматривается осесимметричная труба круглого сечения, состоящая из трех секций, заполненная покоящейся стехиометрической пропано-воздушной смесью при нормальных условиях (см. Рис. 3). Профиль стенки в секции 2 характеризуется квадратичной зависимостью $z(r)$, которая строится однозначно так, чтобы получались: (а) заданная величина блокировки трубы BR; (б) заданное значение угла наклона профилированного элемента φ; (в) фокус полученной параболы лежал на оси симметрии трубы. Для исследования была выбрана «оптимальная» форма сужения $\varphi = 45°$, BR = 0.75 [20]. Предполагается, что движение в трубе возникает в результате вхождения в нее инициирующей УВ с числом Маха M и нулевым градиентом параметров непосредственно за ее фронтом.

В проведенных расчетах используются гексаэдрические сетки с числом ячеек до 20 млн.

Результаты трехмерного численного исследования в целом подтверждают основные этапы и особенности механизма инициирования детонации в трубе с параболическим сужением и ко-

ническим расширением, выявленные в расчетах в осесимметричной постановке в [20]. Рис. 4 иллюстрируют результаты трехмерного численного исследования для случая φ = 45°, BR = 0.75, M = 2.8. Для трехмерной визуализации описываемых процессов используются изоповерхности плотности пропана, а также поля давления и температуры в продольных сечениях трубы и на указанных изоповерхностях. Изозначения в каждом конкретном случае выбираются для наиболее наглядной визуализации описываемой стадии инициирования. Время отсчитывается от момента вхождения в трубу инициирующей УВ.

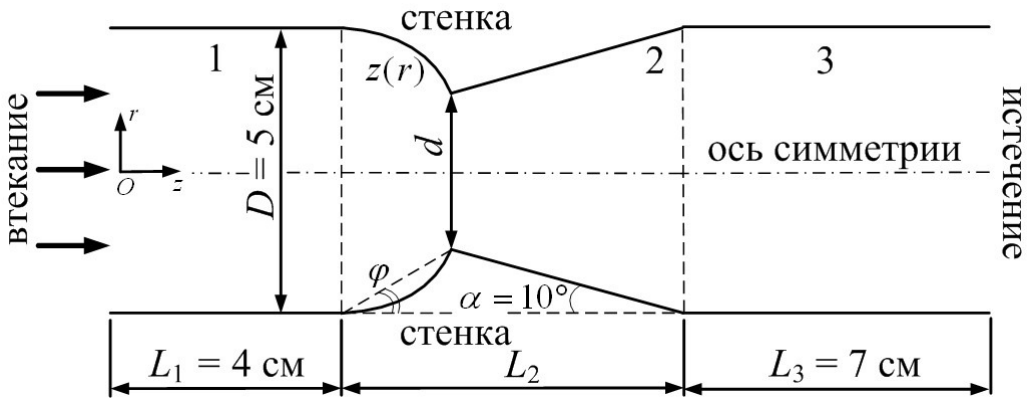
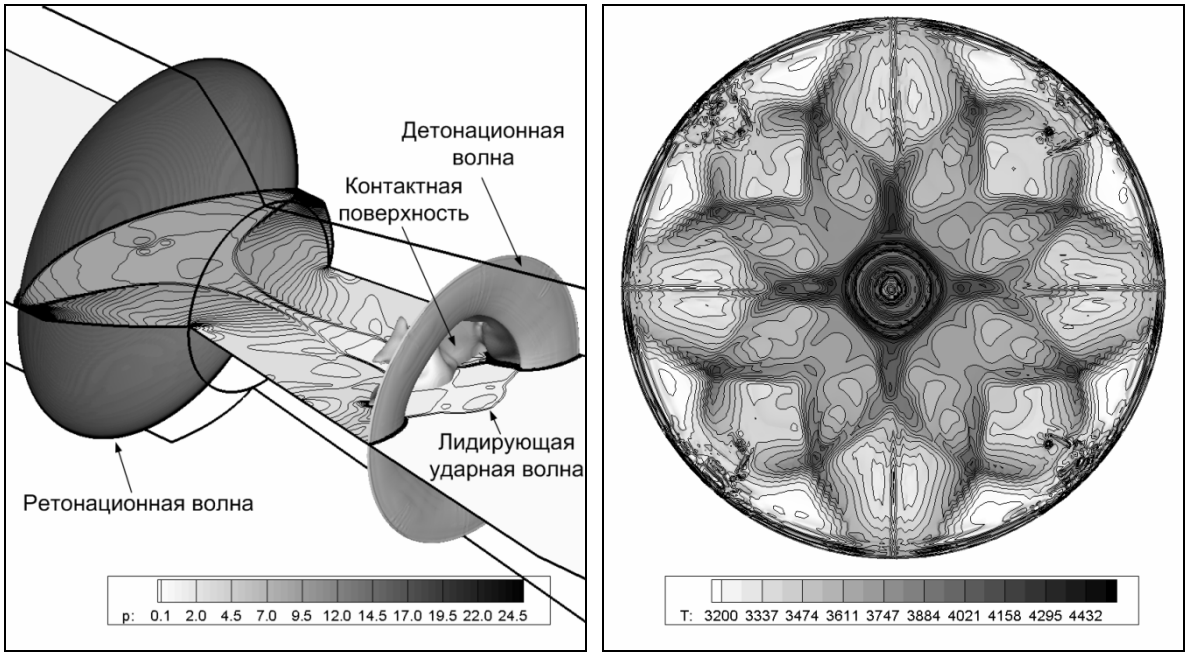


Рис. 3. Схема профилированной трубы с параболическим сужением и коническим расширением.



(а) (б)

Рис. 4. Механизм инициирования детонации: (а) стадия реинициирования детонации, изоповерхность плотности пропана в 0.05 кг/м^3 с нанесенным на нее полем давления, а также поле давления в МПа в продольном сечении, момент времени 95 мкс; (б) структура детонационной волны в выходном сечении трубы, изоповерхность плотности пропана в 0.05 кг/м^3 с нанесенным на нее полем температуры в градусах Кельвина в выходном сечении трубы, момент времени 135 мкс.

Первая стадия механизма связана с переходным двойным Маховским отражением лидирующей УВ от параболического сужения. Вторая стадия – формирование одного или двух локальных взрывов, связанных с кумуляцией волны Маха и отраженной УВ на оси симметрии трубы. Третья стадия – реинициирование детонации в результате отражения волны, вызванной локальным взрывом, от стенок конического расширения (см. Рис. 4а). Важным параметром на

этой стадии является величина угла α, которая была существенно уменьшена в [22] для оптимизации процесса инициирования.

В трехмерном расчете обнаружена потеря потоком изначально осесимметричной структуры, связанная с неустойчивостью детонации. В выходном сечении трубы наблюдается типичная трехмерная картина, зарегистрированная в многочисленных экспериментах по структуре фронта ДВ (см. Рис. 4б). Рис. 4б соответствует, так называемой, 8-ми ячеечной моде ДВ.

7. Инициирование детонации в винтовой трубе

Исследуется труба постоянного круглого сечения с диаметром 28 мм, состоящая из трех секций: входной секции S_1, винтовой секции S_2 и выходной секции S_3. Винтовая секция в свою очередь состоит из трех частей. Первая часть S_2^1 – поверхность, заметаемая окружностью при ее движении, разбиваемом на три составляющие: равномерное движение вдоль оси X, поворот с постоянной скоростью на 90° и равномерное удаление центра окружности от оси X. Вторая часть S_2^2 представляет собой цилиндрическую винтовую поверхность с оборотом на 360°, намотанную на ось X. Третья часть S_2^3 аналогична первой, только центр окружности при движении не удаляется, а равномерно возвращается к оси X (см. Рис. 5). Таким образом, поверхность винтовой секции трубы заметается окружностью в плоскости, параллельной YZ , центр которой движется по закону:

$$x = 28\cdot m,\ m\in[0;6];\ y=\begin{cases}28\cdot m\cdot\cos(90°\cdot m),\ m\in[0;1),\\ 28\cdot\cos(90°\cdot m),\ m\in[1;5),\\ 28\cdot(6-m)\cdot\cos(90°\cdot m),\ m\in[5;6];\end{cases}\quad z=\begin{cases}28\cdot m\cdot\sin(90°\cdot m),\ m\in[0;1),\\ 28\cdot\sin(90°\cdot m),\ m\in[1;5),\\ 28\cdot(6-m)\cdot\sin(90°\cdot m),\ m\in[5;6].\end{cases}$$

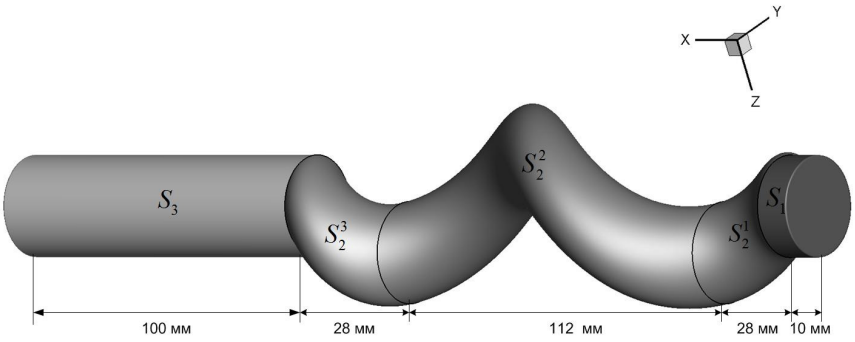


Рис. 5. Схема винтовой трубы.

Проблема построения расчетной сетки связана с тем, что сетка должна быть как можно более приближена к равномерной и ортогональной и в то же время аппроксимировать форму исследуемого объекта. Внутри сложных геометрических областей применяется многоблочный подход. В данном случае сетка разбита на блоки вдоль оси X, причем блоки сдвинуты таким образом, чтобы повторять форму трубы. Для построения ячеек в круглом сечении используется O-сетка. Полученная сетка имеет около 4.6 млн. узлов.

В начальный момент времени в секцию S_1 входит плоская инициирующая УВ с параметрами за ее фронтом, соответствующими параметрам за фронтом УВ с числом Маха M, аналогично постановке задачи в предыдущем разделе. На стенках установлено граничное условие непротекания.

Результаты расчетов показали, что в трубе описанной геометрии удается инициировать детонацию во всем сечении при числе Маха инициирующей УВ, превышающем M = 3.4. В численном эксперименте с M = 3.2 наблюдалась детонация в, так называемом, спиновом режиме, который занимает промежуточное положение между детонацией и дефлаграцией и наблюдается вблизи детонационных пределов [1]. При меньших значениях M инициирование детонации происходило в небольшом объеме, но детонационный процесс быстро затухал, наблюдался режим медленного горения.

Геометрия поверхности трубы такова, что инициирующая УВ претерпевает Маховское отражение от стенки, в результате чего на поверхности трубы образуется тройная точка. В окрестности тройной точки – зоны повышенных давления и температуры – создаются благоприятные условия для самовоспламенения смеси. Самовоспламенение смеси у стенки наблюдается во всех трех вычислительных экспериментах. Возникновение очага самовоспламенения сопровождается возникновением УВ, распространение которой по предварительно нагретой инициирующей ударной волной смеси приводит к возникновению второго очага самовоспламенения при числах Маха инициирующей УВ M = 3.4 и M = 3.2 (см. Рис. 6). Интенсивности же инициирующей УВ с M = 3.0 оказалось недостаточно для возникновения второго очага самовоспламенения.

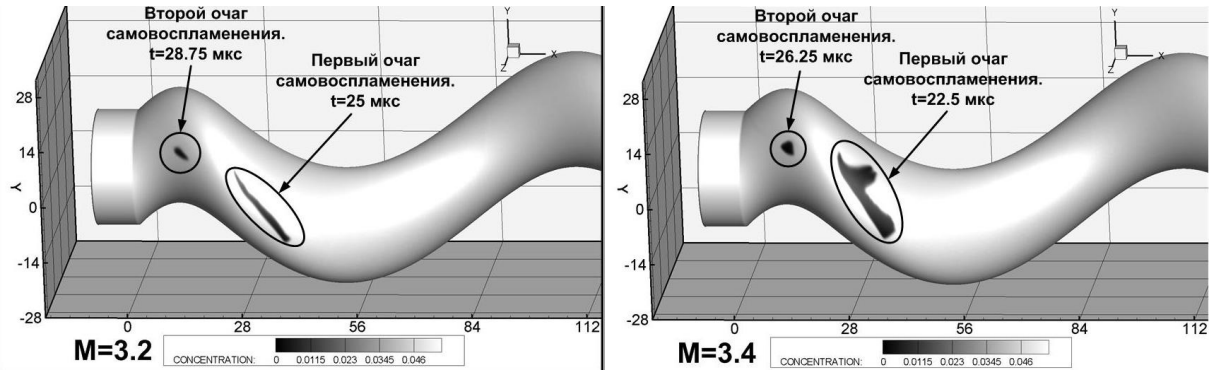


Рис. 6. Возникновение очагов самовоспламенения смеси в случаях перехода ударной волны в детонационную. Поле концентрации пропана на поверхности трубы.

В связи с увеличением площади горения при M = 3.4 и 3.2 энергоприход увеличивается, а фронт лидирующей УВ ускоряется.

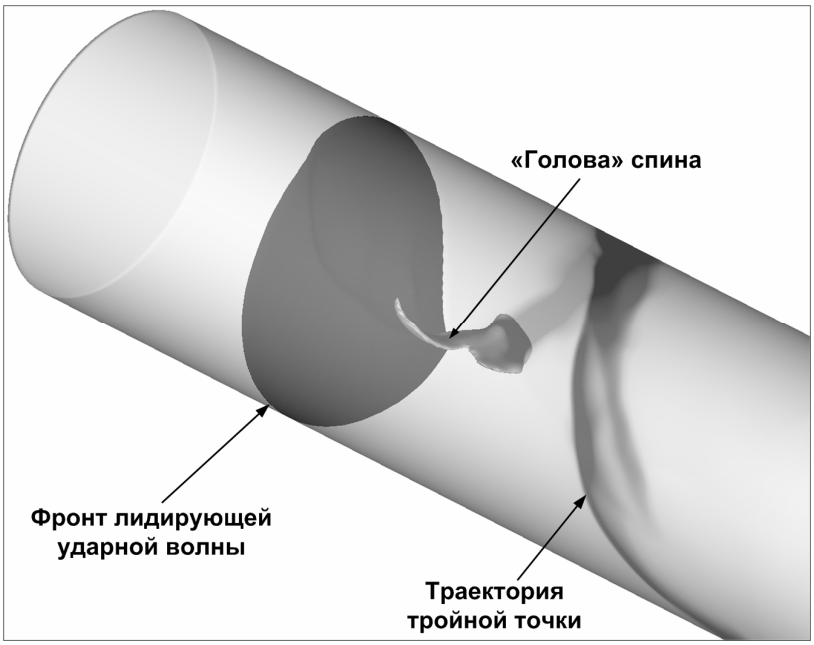


Рис. 7. Структура спиновой детонации в выходной секции винтовой трубы. Поле максимумов давления на поверхности трубы, изоповерхность температуры 298 К (фронт лидирующей УВ), изоповерхность давления 102 атм («голова» спина). Момент времени 107.5 мкс.

При выходе ДВ в выходную цилиндрическую секцию при M = 3.4 детонация занимает все сечение, при этом детонационный фронт имеет сложную форму. При выходе же детонационной волны при M = 3.2 наблюдается спиновая детонация (см. Рис. 7). Многочисленные экспериментальные исследования показывают, что траектория «головы» спина относительно трубы – спи-

раль, наклоненная под углом примерно 45° к образующей [2]. В экспериментах такую спираль регистрируют следовым методом, покрыв перед опытом стенки трубы тонким слоем сажи; тогда ДВ оставляет на саже следы, вычерчиваемые тройными точками. В численном эксперименте подобная картина наблюдается, если отобразить на поверхности трубы поле максимальных давлений, т.е. в каждой точке вывести максимальное за все время расчета значение давления в ней. Для случая $M = 3.2$ угол наклона составляет примерно 50°.

8. Заключение

В работе представлены математические модели, численный метод и методика распараллеливания расчетного алгоритма для решения задач инициирования и распространения волн газовой детонации в трехмерных трубах сложной формы. Приведены характеристики качества распараллеливания расчетного алгоритма, основанного на декомпозиции расчетной области в случае использования неструктурированных гексаэдрических сеток.

Проанализирован механизм инициирования детонации в осесимметричной трубе с параболическим сужением и коническим расширением. Результаты трехмерного численного исследования в целом подтверждают основные этапы и особенности механизма инициирования детонации в трубе указанной геометрии, выявленные ранее в расчетах в осесимметричной постановке. Вместе с тем, в трехмерном расчете обнаружена потеря потоком изначально осесимметричной структуры, связанная с неустойчивостью фронта ДВ. Основные результаты вычислительных экспериментов подтверждаются в натурных опытах.

Проанализирован механизм инициирования детонации, а также проведена классификация режимов горения, возникающих в винтовой трубе в зависимости от интенсивности инициирующей УВ. Обнаружено, что если число Маха инициирующей УВ превышает некоторое критическое значение, в выходной секции трубы наблюдается ДВ, занимающая все сечение. При ослаблении начальной УВ детонация в выходной секции распространяется в спиновом режиме. В случае дальнейшего ослабления инициирующей УВ инициирования детонации не происходит.

Практическая значимость результатов исследования связана с попытками снижения затрат энергии на инициирование детонации при разработке устройств, использующих детонационное сжигание топлива [23].

Литература

1. Васильев А.А., Митрофанов В.В., Топчиян М.Е. Детонационные волны в газах // Физика горения и взрыва. – 1987. – № 5. – С. 109 – 130.
2. Митрофанов В.В. Детонация гомогенных и гетерогенных систем. – Новосибирск: Изд-во Института гидродинамики им. М.А. Лаврентьева СО РАН, 2003.
3. Белоцерковский О.М. Численное моделирование в механике сплошных сред. – М.: Наука, 1984.
4. Семенов И.В., Уткин П.С., Марков В.В. Численное моделирование двумерных детонационных течений на многопроцессорной вычислительной технике // Вычислительные методы и программирование. – 2008. – Т. 9. – С. 119 – 128.
5. Frolov S.M., Aksenov V.S., Shamshin I.O. Detonation propagation through U-bends // Nonequilibrium processes, Vol. 1: Combustion and Detonation. Eds. G. Roy, S. Frolov, M. Starik. – 2005. – P. 348 – 364.
6. Barth T., Ohlberger M. Finite Volume Methods: Foundation and Analysis // Encyclopedia of Computational Mechanics. – 2004. – Vol. 1. – P. 439 – 470.
7. Куликовский А.Г., Погорелов Н.В., Семенов А.Ю. Математические вопросы численного решения гиперболических систем уравнений. – М.: ФИЗМАТЛИТ, 2001.

8. Годунов С.К, Забродин А.В., Иванов М.Я., Крайко А.Н., Прокопов Г.П. Численное решение многомерных задач газовой динамики. – М.: Наука, 1976.

9. Karypis G., Kumar V. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system // Technical report, Department of Computer Science, University of Minnesota, 1998: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

10. Walshaw C., Cross M., Everett M.G. A localized algorithm for optimising unstructured mesh partitions // International Journal Supercomputer Applications. – 1995. – Vol. 17, No. 4. – P. 280 – 295.

11. Preis R., Diekmann R. The party partitioning-library // Technical report TR-RSFB-96-024, Universitat-GH Paderborn, 1996.

12. Pellegrini F. PT-Scotch and LibScotch 5.1 // Technical report LaBRI, 2008: http://www.labri.fr/perso/pelegrin/scotch/.

13. Зельдович Я.Б., Компанеец А.С. Теория детонации. – М.: Госуд. изд-во технико-теорет. литры, 1955.

14. Седов Л.И., Коробейников В.П., Марков В.В. Теория распространения взрывных волн // Труды Математического института им. В.А. Стеклова РАН. – 1986. – № 175. – С. 178 – 216.

15. Борисов А.А. Инициирование детонации в газовых и двухфазных смесях / Импульсные детонационные двигатели. Под ред. д.ф.-м.н. Фролова С.М. – М.: ТОРУС ПРЕСС, 2006.

16. Oran E.S., Gamezo V.N. Origins of the Deflagration-to-detonation Transition in Gas-phase Combustion // Combustion and Flame. – 2007. – Vol. 148. – P. 4 – 47.

17. Фролов С.М. Быстрый переход горения в детонацию // Химическая физика. – 2008. – Т. 27, № 6. – С. 31 – 44.

18. Frolov S.M. Initiation of Strong Reactive Shocks and Detonation by Traveling Ignition Pulses // Journal of Loss Prevention. – 2005. – Vol. 19, № 2 – 3. – P. 238 – 244.

19. Фролов С.М., Семенов И.В., Комиссаров П.В., Уткин П.С., Марков В.В. Сокращение длины и времени перехода горения в детонацию в трубе с профилированными регулярными препятствиями // Доклады Академии наук. – 2007. – Т. 415, № 4. – С. 509 – 513.

20. Семенов И.В., Уткин П.С., Марков В.В. Численное моделирование инициирования детонации в профилированной трубе // Физика горения и взрыва. – 2009. – Т. 45, № 6. – С. 73 – 81.

21. Semenov I.V., Utkin P.S., Markov V.V., Frolov S.M., Aksenov V.S. Numerical and experimental investigation of detonation initiation in profiled tubes // 22nd ICDERS Proceedings. – 2009. – CD, Paper No. 168.

22. Фролов С.М., Аксенов В.С. Инициирование газовой детонации в трубе с профилированным препятствием // Доклады Академии наук. – 2009. – Т. 427, № 3. – С. 344 – 347.

23. Фролов С.М. Импульсное детонационное горение: новое поколение энергетических установок // Энергетика. – 2008. – Т. 3, № 41. – С. 44 – 45.

Применение параллельных алгоритмов при решении задач гидродинамики методом вихревых элементов *

И.К. Марчевский, Г.А. Щеглов

Рассматривается параллельный алгоритм расчета методом вихревых элементов пространственного обтекания тел дозвуковым потоком среды. Традиционно в задачах такого типа распараллеливается только процедура вычисления взаимных влияний вихревых элементов, аналогичная задаче N тел. В настоящей работе проанализирована трудоемкость всех операций алгоритма и показано, что их распараллеливание позволяет существенно повысить скорость счета. Исследована эффективность распараллеливания вычислений при использовании различных вычислительных систем. Приведены результаты численного моделирования.

1. Введение

В инженерной практике часто требуется исследовать взаимодействие элементов конструкций с дозвуковым потоком среды. Такие задачи возникают в авиации при проектировании летательных аппаратов, в промышленной аэродинамике при расчете ветровой нагрузки на здания и сооружения, в области экологии при исследовании распространения загрязнения и различных примесей, в электроэнергетике при проектировании ветроэнергетических установок и исследовании колебаний проводов на воздушных линиях электропередачи. В настоящее время в связи с интенсивным развитием компьютерной техники появилась возможность проведения относительно дешевого численного эксперимента методами вычислительной гидродинамики. Это позволяет сократить количество дорогостоящих экспериментов в аэродинамических трубах. Распространение суперкомпьютерных технологий, предоставляющих пользователю возможность использования многих вычислительных ядер, позволяет существенно сократить время расчета за счет использования библиотек параллельных вычислений, таких как MPI [1].

Хотя суперкомпьютерные технологии широко доступны пользователям на протяжении как минимум последнего десятилетия, их практическое использование для решения прикладных задач осложняется необходимостью создания эффективных параллельных алгоритмов. Это связано с тем, что далеко не для всех численных методов можно предложить простой способ распараллеливания. В вычислительной гидродинамике чаще всего используются сеточные методы решения уравнений движения среды. Данные методы обладают многими преимуществами и позволяют решать широкий спектр задач, однако с точки зрения затрат вычислительных ресурсов они уступают бессеточным методам, в которых вместо расчета характеристик течения в узлах сетки рассматривается движение множества частиц [2, 3]. Частица — вихревой элемент — переносит завихренность, индуцирующую в окружающем пространстве скорость по закону Био-Савара. Таким образом, каждый вихревой элемент оказывает влияние на поле скоростей во всем пространстве, а суммарное поле скоростей находится как сумма влияний всех вихревых элементов. По известным положениям вихревых элементов можно также вычислить давление в любой точке течения. Область течения, занятая завихренностью — так называемый вихревой след, — как правило, достаточно компактна, поэтому при расчете параметров течения методом вихревых элементов вычислительные ресурсы концентрируются именно в этой области, а не “распыляются” по всей сетке, которая при решении задач внешнего обтекания строится в области, намного превышающей размеры вихревого следа [4].

Целью настоящей работы является изучение возможности эффективного распараллеливания алгоритма решения задач вычислительной гидродинамики методом вихревых эле-

*Работа выполнена при частичной финансовой поддержке гранта РФФИ № 09-08-00657-а.

ментов. Для достижения поставленной цели решаются следующие задачи: оценка трудоемкостей отдельных операций алгоритма метода вихревых элементов, создание программного комплекса для проведения расчета на многопроцессорных вычислительных комплексах и оценка его производительности на различных вычислительных системах.

2. Метод вихревых элементов в задаче расчета пространственного обтекания тел

2.1. Постановка задачи

Течение несжимаемой среды ($\nabla \cdot (\rho \boldsymbol{V}) = 0$) постоянной плотности ρ описывается уравнением Навье-Стокса

$$\frac{\partial \boldsymbol{V}}{\partial t} + (\boldsymbol{V} \cdot \nabla)\boldsymbol{V} - \nu \Delta \boldsymbol{V} = -\nabla \left(\frac{p}{\rho} \right),$$

где $\boldsymbol{V}$ — скорость среды, p — давление, ν — коэффициент кинематической вязкости. На бесконечном удалении от обтекаемого тела выполняется граничное условие затухания возмущений

$$\boldsymbol{V} \to \boldsymbol{V}_\infty, \quad p \to p_\infty,$$

а на поверхности неподвижного обтекаемого тела K — условие прилипания

$$\boldsymbol{V}(\boldsymbol{r}) = 0, \quad \boldsymbol{r} \in K.$$

Для приближенного моделирования течения используется подход Прандтля — считается, что влияние вязкости существенно лишь вблизи поверхности обтекаемого тела. Вне этого слоя среда считается идеальной, и ее течение описывается уравнением Эйлера, которое можно представить в форме Гельмгольца [5]

$$\frac{D\boldsymbol{\Omega}}{Dt} = (\boldsymbol{\Omega} \cdot \nabla)\boldsymbol{V}. \tag{1}$$

Это уравнение описывает движение завихренности $\boldsymbol{\Omega} = \nabla \times \boldsymbol{V}$ в области течения, при этом скорость среды может быть восстановлена по известному полю завихренности при помощи закона Био-Савара ($d = 2$ для плоскопараллельных и $d = 3$ для пространственных течений):

$$\boldsymbol{V}(\boldsymbol{r}) = \boldsymbol{V}_\infty + \frac{1}{2(d-1)\pi} \int \frac{d\boldsymbol{\Omega}(\boldsymbol{\xi}) \times (\boldsymbol{r} - \boldsymbol{\xi})}{|\boldsymbol{r} - \boldsymbol{\xi}|^d}. \tag{2}$$

Образование новой завихренности происходит вблизи поверхности обтекаемого тела под влиянием вязкости. В простейшем случае действие вязкости можно рассматривать лишь как причину генерации завихренности у поверхности тела, в этом случае ее интенсивность находится из условия непротекания на поверхности тела

$$\boldsymbol{n} \cdot \boldsymbol{V}(\boldsymbol{r}) = 0, \quad \boldsymbol{r} \in K, \tag{3}$$

а вся образующаяся завихренность становится частью вихревого следа, что обеспечивает выполнение условия прилипания, равносильное заданию определенного потока завихренности с поверхности тела в область течения [6].

По известному полю завихренности, используя аналог интеграла Коши-Лагранжа [7], можно вычислить давление в любой точке и найти нагрузки, действующие на обтекаемое тело.

2.2. Метод вихревых элементов

При заданном начальном распределении завихренности $\boldsymbol{\Omega}$ задача моделирования обтекания тела сводится к уравнениям (1)–(3), которые могут быть эффективно решены методом вихревых частиц [3]. Под вихревой частицей понимается элементарное поле завихренности, связанное с некоторым маркером $\boldsymbol{r}_i$, имеющее интенсивность Γ_i и характеризующееся вектором параметров $\boldsymbol{\omega}_i$. Интенсивности, положения и остальные параметры вихревых частиц задаются так, чтобы аппроксимировать поле завихренности $\boldsymbol{\Omega}$. Если интеграл в (2) для каждой вихревой частицы можно представить достаточно простым аналитическим выражением, то вихревая частица называется вихревым элементом (ВЭ), а поле скоростей (2) представляется суперпозицией полей скоростей N отдельных ВЭ [3]:

$$\boldsymbol{V}(\boldsymbol{r}) = \boldsymbol{V}_\infty + \sum_{i=1}^{N} \boldsymbol{Q}(\boldsymbol{r} - \boldsymbol{r}_i, \boldsymbol{\omega}_i)\Gamma_i. \tag{4}$$

Здесь $\boldsymbol{Q}(\boldsymbol{r} - \boldsymbol{r}_i, \boldsymbol{\omega}_i)$ — влияние в точке $\boldsymbol{r}$ от i-го ВЭ.

На каждом шаге расчета вблизи поверхности обтекаемого тела происходит генерация N_0 новых ВЭ, интенсивности Γ_j^0 которых находятся из (3) с учетом (4):

$$\boldsymbol{n} \cdot \sum_{j=1}^{N_0} \boldsymbol{Q}(\boldsymbol{r} - \boldsymbol{r}_j^0, \boldsymbol{\omega}_j^0)\Gamma_j^0 = -\boldsymbol{n} \cdot \left(\boldsymbol{V}_\infty + \sum_{i=1}^{N} \boldsymbol{Q}(\boldsymbol{r} - \boldsymbol{r}_i, \boldsymbol{\omega}_i)\Gamma_i \right). \tag{5}$$

Для определения интенсивностей Γ_j^0 уравнение (5) записывается в контрольных точках $\boldsymbol{r}_{\boldsymbol{k}}$, задаваемых на поверхности тела K, что приводит к системе линейных алгебраических уравнений

$$\boldsymbol{n} \cdot \sum_{j=1}^{N_0} \boldsymbol{Q}(\boldsymbol{r}_k - \boldsymbol{r}_j^0, \boldsymbol{\omega}_j^0)\Gamma_j^0 = -\boldsymbol{n} \cdot \left(\boldsymbol{V}_\infty + \sum_{i=1}^{N} \boldsymbol{Q}(\boldsymbol{r}_k - \boldsymbol{r}_i, \boldsymbol{\omega}_i)\Gamma_i \right), \quad k = 1, \ldots, N_0. \tag{6}$$

Уравнение (1) при использовании ВЭ приводится к системе обыкновенных дифференциальных уравнений вида

$$\frac{d\boldsymbol{r}_i}{dt} = \boldsymbol{V}(\boldsymbol{r}_i), \quad \frac{d\boldsymbol{\omega}_i}{dt} = \boldsymbol{B}(\boldsymbol{r}_i, \boldsymbol{\omega}_i), \tag{7}$$

где $\boldsymbol{V}(\boldsymbol{r}_i)$ — скорость движения маркера ВЭ по траектории жидкой частицы, $\boldsymbol{B}(\boldsymbol{r}_i, \boldsymbol{\omega}_i)$ — скорость изменения параметров ВЭ.

Для реализации метода вихревых элементов необходимо выбрать тип вихревого элемента, который определяет характеристики ВЭ — $\boldsymbol{r}$, $\boldsymbol{\omega}$, Γ, и вид функций $\boldsymbol{Q}(\boldsymbol{r}, \boldsymbol{\omega})$, $\boldsymbol{B}(\boldsymbol{r}, \boldsymbol{\omega})$. В качестве ВЭ в данной работе используется симметричный вортон-отрезок [8].

2.3. Алгоритм расчета методом вихревых элементов

Алгоритм расчета течения методом вихревых элементов (МВЭ) состоит из пяти основных операций, которые выполняются на каждом шаге расчета.

1. Формирование и решение системы линейных алгебраических уравнений (6) для удовлетворения граничных условий на поверхности обтекаемого тела с последующим увеличением числа ВЭ в вихревой пелене ($\mathtt{N} := \mathtt{N} + \mathtt{N_0}$).

2. Вычисление правых частей системы (7), в частности, скоростей вихревых элементов по формуле (4)

$$\boldsymbol{V}(\boldsymbol{r}_i) = \boldsymbol{V}_\infty + \sum_{s=1}^{N} \boldsymbol{Q}(\boldsymbol{r}_i - \boldsymbol{r}_s, \boldsymbol{\omega}_s)\Gamma_s, \quad i = 1, \ldots, N.$$

3. Вычисление нагрузок, действующих на тело со стороны потока, по рассчитанному полю скоростей [7].

4. Решение системы ОДУ (7) на текущем шаге расчета, перемещение ВЭ в новые положения и предотвращение проникновения ВЭ внутрь тела вследствие погрешностей дискретизации.

5. Реструктуризация вихревой пелены, приводящая к уменьшению числа ВЭ на N_R элементов за счет объединения близкорасположенных ВЭ или исключения из расчетной схемы ВЭ, удалившихся от тела на значительное расстояние ($\mathtt{N} := \mathtt{N} - \mathtt{N_R}$).

Критерием остановки расчета является выполнение заданного числа временны́х шагов.

3. Оценка трудоемкостей различных операций алгоритма

Рассмотренный выше алгоритм имеет следующую особенность: на каждом шаге происходит изменение числа ВЭ (в операциях 1 и 5 алгоритма). Точность расчета напрямую зависит от N_0, а скорость счета — от квадрата текущего числа ВЭ (N^2). Это определяется необходимостью вычислений попарных влияний ВЭ друг на друга в операциях 2 и 5 алгоритма. Трудоемкость операций 3 и 4 — порядка $N \cdot N_0$.

Увеличение числа N вихревых элементов, моделирующих след за обтекаемым телом, при выполнении расчета на одном вычислительном ядре приводит к росту времени выполнения каждого последующего шага расчета, как показано в таблице 1. Начиная с некоторого шага расчета, процедура реструктуризации вихревой пелены позволяет стабилизировать число вихревых элементов ($N_R \approx N_0$), которое затем изменяется мало.

Таблица 1. Время выполнения одного шага расчета

Номер шага	40	90	160	300 и далее
Число ВЭ	2000	2500	3000	3500
Время выполнения шага расчета, с	8,6	10,1	12,1	13,8

Отметим, что данный расчет производился на одном ядре вычислительного кластера. Использование более мощных процессоров позволяет сократить абсолютное время счета, однако замедление расчета с увеличением числа ВЭ будет происходить с тем же темпом.

Относительные трудоемкости операций алгоритма в зависимости от числа вихревых элементов в следе представлены на рис. 1. Операция 2 является доминирующей. При увеличении числа вихревых элементов ее доля возрастает.

Рис. 1. Относительные трудоемкости операций при расчете без распараллеливания

Однако, если распараллеливать только операцию 2, а остальные операции выполнять на одном вычислительном ядре, как это часто делается при реализации вихревых методов, то получить высокую эффективность распараллеливания, особенно при использовании большого числа вычислительных ядер, не удается. Это связано с тем, что очень быстро начинают доминировать затраты на выполнение остальных операций, во время которых большинство вычислительных ядер будут простаивать. На рис. 2 для алгоритма, в котором распараллелена только операция 2, показаны диаграммы трудоемкостей операций при использовании различного числа вычислительных ядер.

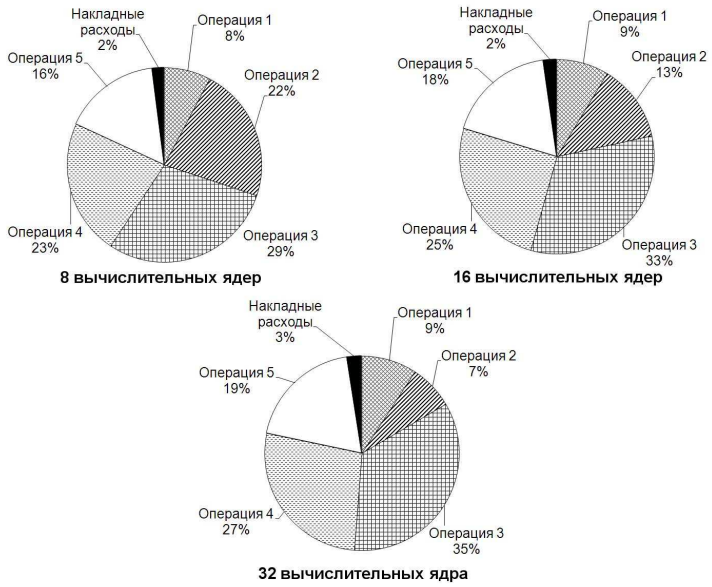


Рис. 2. Относительные трудоемкости при распараллеливании операции 2

Наибольшего эффекта от распараллеливания можно достичь при сохранении соотношения трудоёмкостей различных операций с увеличением количества вычислительных ядер. На рис. 3 приведены аналогичные диаграммы для алгоритма, в котором распараллелены все операции.

В таблице 2 приведены затраты времени при выполнении одного шага расчета параллельного алгоритма при использовании разного числа вычислительных ядер.

Таблица 2. Время выполнения одного шага расчета с использованием параллельного алгоритма

Номер шага	40	90	160	300 и далее
Число ВЭ	2000	2500	3000	3500
$T_{\text{шага}}$, с (1 ядро)	8,6	10,1	12,1	13,8
$T_{\text{шага}}$, с (2 ядра)	4,5	5,4	6,5	7,4
$T_{\text{шага}}$, с (4 ядра)	2,4	2,8	3,4	3,9
$T_{\text{шага}}$, с (8 ядер)	1,3	1,5	1,8	2,1
$T_{\text{шага}}$, с (16 ядер)	0,8	1,0	1,1	1,3
$T_{\text{шага}}$, с (24 ядра)	0,7	0,8	0,9	1,0

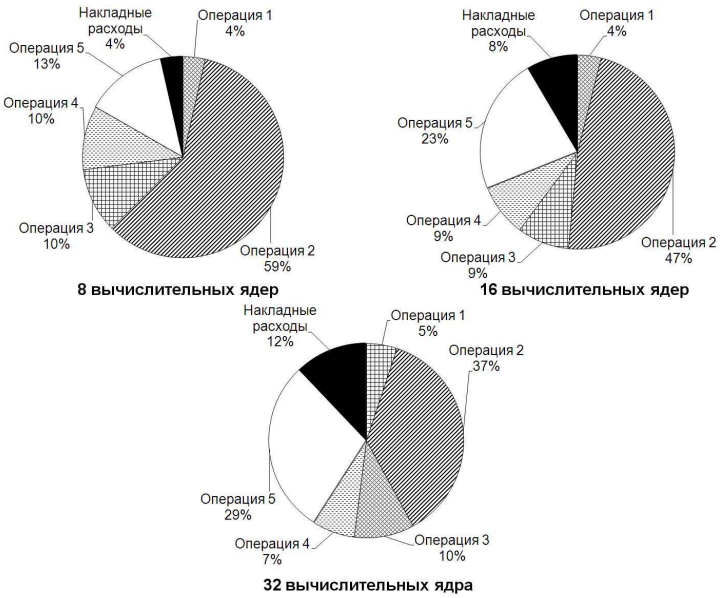


Рис. 3. Относительные трудоемкости при распараллеливании всех операций

4. Оценка эффективности различных вычислительных систем

Эффективность распараллеливания вычислений в МВЭ связана не только с совершенством параллельного алгоритма, но и с возможностями многопроцессорных вычислительных комплексов. Для оценки ускорения расчета использовались как персональные ЭВМ, объединенные в локальную сеть, так и высокопроизводительные кластерные системы, в частности МВС-100К и МВС-6К МСЦ РАН, а также кластерная система HP BLc3000 Twr CTO Enclosure на базе 2-х блейд серверов HP ProLiant 2xBL220c, полученная МГТУ им. Н.Э. Баумана в рамках программы “Университетский кластер” [9]. Сравнение ускорения счета одного и того же алгоритма МВЭ для различных вычислительных комплексов представлено на рис. 4.

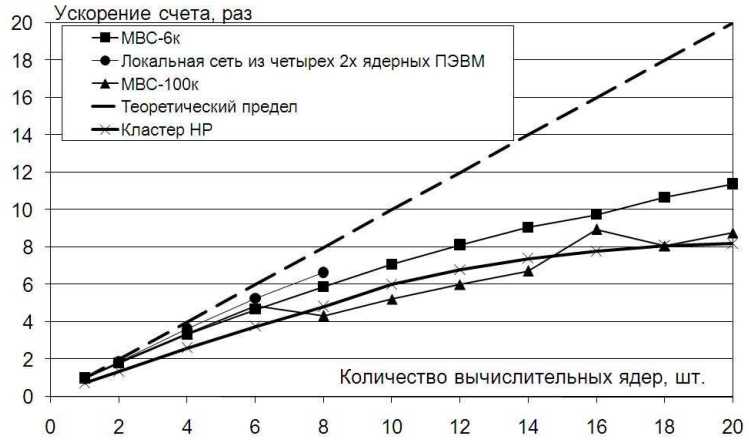


Рис. 4. Ускорение расчетов при использовании различных многопроцессорных систем

Видно, что при малом числе вычислительных ядер удается достичь практически линейного ускорения, что доказывает хорошую масштабируемость разработанного параллельного алгоритма. Кажущееся преимущество локальной вычислительной сети является относительным: абсолютное время счета на ПЭВМ превышает время счета на кластерах.

Эффективность распараллеливания при использовании большого числа ядер (более 16) снижается, это может объясняться возрастающими объемами межпроцессорного обмена и ростом доли “накладных расходов”.

5. Примеры расчетов

Описанный алгоритм реализован авторами в программном комплексе MVE3D с использованием библиотеки параллельных вычислений MPI.

На рис. 5 показан вид вихревых пелен за телами простейшей формы, образуемых симметричными вортонами-отрезками в ходе моделирования нестационарного обтекания.

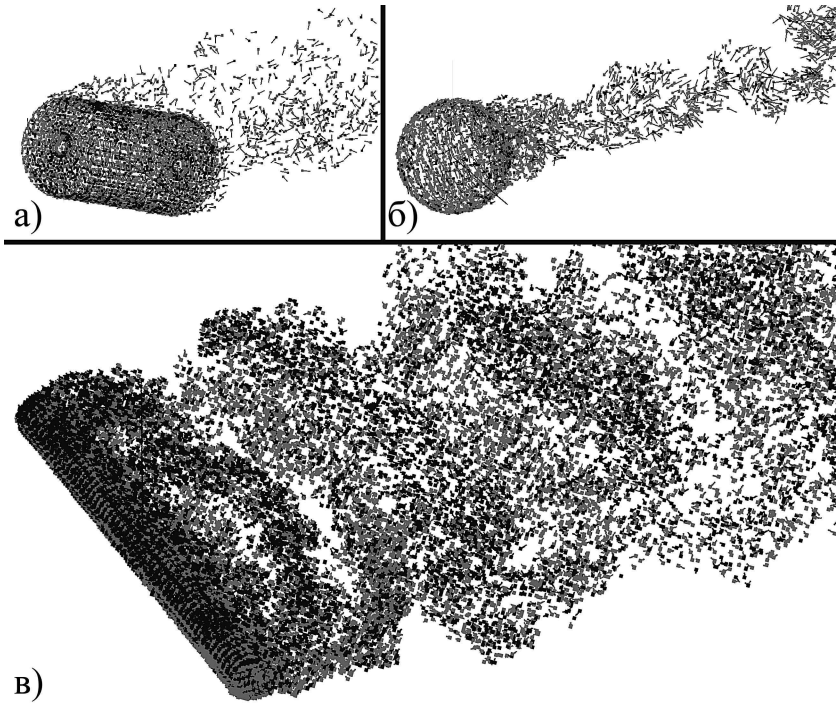


Рис. 5. Вихревые следы за телами простейшей формы: а — цилиндр с удлинением 2; б — шар; в — цилиндр с удлинением 12;

Высокая скорость расчетов, обеспечиваемая за счет использования параллельных алгоритмов, позволяет рассчитывать не только нестационарные переходные режимы обтекания, но и достаточно быстро проводить расчеты “на установление” до тех пор, пока обтекание не выходит на стационарный режим.

На рис. 6 показаны результаты расчета стационарных аэродинамических коэффициентов лобового сопротивления и подъемной силы для кругового цилиндра с удлинением, равным $l/d = 2$, в сравнении с экспериментальными данными [10].

6. Выводы

Применение параллельных алгоритмов метода вихревых элементов, реализованных с использованием библиотеки параллельных вычислений MPI, позволило значительно ускорить выполнение расчетов и эффективно использовать различные современные вычислительные системы. При проведении расчетов на многопроцессорных кластерах удалось увеличить число вихревых элементов в расчетной схеме, повысив тем самым точность вычислений, при сохранении высокой скорости счета.

Авторы благодарят Межведомственный суперкомпьютерный центр РАН за предостав-

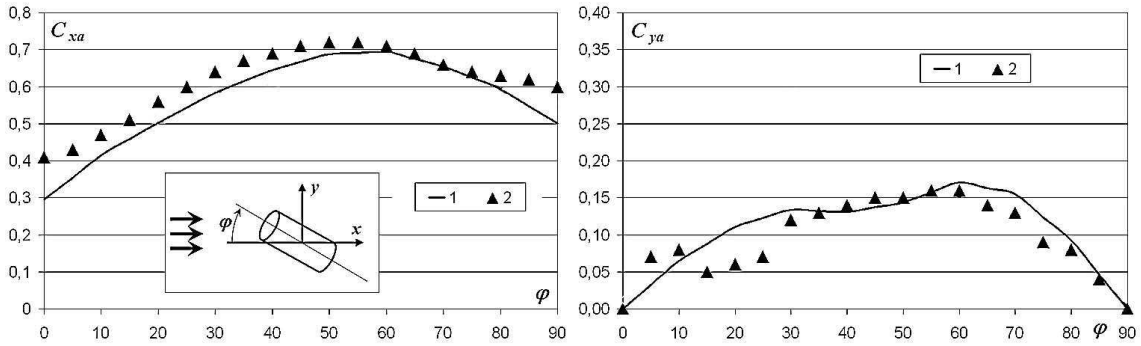


Рис. 6. Стационарные коэффициенты лобового сопротивления и подъемной силы: 1 — эксперимент [10]; 2 — расчет по программе MVE3D

ленную возможность использования кластеров МВС-100К и МВС-6000IM, а также программу “Университетский кластер”.

Литература

1. Гергель В.П. Теория и практика параллельных вычислений. — М.: БИНОМ. Лаборатория знаний, 2007. — 424 с.

2. Трехмерное отрывное обтекание тел произвольной формы / Под ред. С.М. Белоцерковского. — М.: ЦАГИ, 2000. — 265 с.

3. Cottet G.-H., Koumoutsakos P. Vortex methods: theory and practice // Cambridge University Press, 2000. — 465 с.

4. Сарпкайя Т. Вычислительные методы вихрей. Фримановская лекция (1988) // Современное машиностроение. Сер. А. — 1989. — № 10. — С. 1-60.

5. Лойцянский Л.Г. Механика жидкости и газа. — М.: Дрофа, 2003. — 840 с.

6. Lighthill M.J. Introduction // Boundary Layer Theory / J. Rosenhead, ed. — New York: Oxford University Press, 1963. — P.54–61.

7. Андронов П.Р., Гувернюк С.В., Дынникова Г.Я. Вихревые методы расчета нестационарных гидродинамических нагрузок. — М.: Изд-во Моск. ун-та, 2006. — 184 с.

8. Марчевский И.К., Щеглов Г.А. Модель симметричного вортона-отрезка для численного моделирования пространственных течений идеальной несжимаемой среды // Вестник МГТУ им. Н.Э. Баумана. Естественные науки. — 2008. — № 4. — С. 62–71.

9. Институт системного программирования РАН — Программа “Университетский кластер”: [http://www.ispras.ru/ru/unicluster/], 14.12.2009.

10. Девнин С.И. Аэрогидромеханика плохообтекаемых конструкций. — Л.: Судостроение, 1983. — 320 с.

Численное моделирование и экспериментальные исследования грязевого вулкана «Гора Карабетова» вибросейсмическими методами.*

Б.М. Глинский, Д.А. Караваев, В.В. Ковалевский, В.Н. Мартынов

Разработан алгоритм, создан комплекс параллельных программ и проведены тестовые расчеты по выбору оптимальной схемы распараллеливания на кластерах Сибирского Суперкомпьютерного Центра СО РАН. Проведено математическое моделирование распространения упругих волн от точечного источника в моделях трехмерных упругих сред, характерных для грязевых вулканов. Приводятся результаты обработки данных вибросейсмического эксперимента на грязевом вулкане "Гора Карабетова". Сравниваются данные численного и натурного экспериментов.

1. Введение

Грязевые вулканы широко распространены на земном шаре. По некоторым оценкам их более 700. Наибольшее количество грязевых вулканов наблюдается в Азербайджане (более 300). Второй по количеству таких вулканов в СНГ является Керченско-Таманская область. Грязевой вулканизм – сложное, малоизученное геологическое образование. В настоящее время установлена связь грязевого вулканизма с наличием залежей углеводородов и динамикой глубинных флюидов, однако до сих пор достоверно не известен механизм образования таких вулканов. Одним из интересных и активных вулканов в Таманской грязевулканической провинции является грязевой вулкан «Гора Карабетова» [1, 8].

Институт вычислительной математики и математической геофизики СО РАН совместно с Институтом физики Земли РАН и Кубанским Государственным Университетом, начиная с 2005 года, поставил ряд экспериментальных исследований по вибросейсмическому зондированию этих уникальных природных явлений. В частности, были поставлены эксперименты на грязевых вулканах Тамани – «Шуго», «Ахтанизовский», «Гора Карабетова» [2-5]. Наиболее детальные наблюдения были проведены на последнем вулкане и некоторые результаты этих экспериментов будут приведены в данной работе. Также впервые представлена математическая модель грязевого вулкана «Гора Карабетова», приводятся результаты численного моделирования и сравнение натурного и численного экспериментов.

2. Математическое моделирование грязевого вулкана «Гора Карабетова»

В настоящее время имеется широкий спектр численных методов применяемых для моделирования полных волновых полей в неоднородных упругих средах. Из всех известных методов численного моделирования распространения упругих волн наиболее гибкими, в случае сложно построенных 3-х мерно неоднородных упругих сред, являются разностный метод и метод конечных элементов, но их использование требует больших вычислительных затрат даже при применении кластерных СуперЭВМ.

Разработанная параллельная программа, представленная в данной работе, предназначена для численного моделирования распространения упругих волн в трехмерно неоднородных моделях упругих сред, с использованием конечно-разностного метода.

*Работа выполнена при поддержке грантов РФФИ: № 07-05-00858, 07-07-00214, 09-07-12075, проект СО РАН 16.5, ИП СО РАН № 133, № 26.

2.1 Постановка задачи

Численное моделирование распространения сейсмических волн в сложно построенных упругих неоднородных средах проводится на основе полной системы уравнений теории упругости с соответствующими начальными и граничными условиями. Данная постановка задачи представлена в терминах вектора скоростей смещений $\vec{u} = (U, V, W)^T$ и тензора напряжений $\vec{\sigma} = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \sigma_{xy}, \sigma_{xz}, \sigma_{yz})^T$:

$$\rho \frac{\partial \vec{u}}{\partial t} = [A]\vec{\sigma} + \vec{F}(t, x, y, z), \frac{\partial \vec{\sigma}}{\partial t} = [B]\vec{u},$$

$$A = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & \frac{\partial}{\partial y} & \frac{\partial}{\partial z} & 0 \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial x} & 0 & \frac{\partial}{\partial z} \\ 0 & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} & \frac{\partial}{\partial y} \end{bmatrix}, \; B = \begin{bmatrix} (\lambda + 2\mu)\frac{\partial}{\partial x} & \lambda \frac{\partial}{\partial y} & \lambda \frac{\partial}{\partial z} \\ \lambda \frac{\partial}{\partial x} & (\lambda + 2\mu)\frac{\partial}{\partial y} & \lambda \frac{\partial}{\partial z} \\ \lambda \frac{\partial}{\partial x} & \lambda \frac{\partial}{\partial y} & (\lambda + 2\mu)\frac{\partial}{\partial z} \\ \mu \frac{\partial}{\partial y} & \mu \frac{\partial}{\partial x} & 0 \\ \mu \frac{\partial}{\partial z} & 0 & \mu \frac{\partial}{\partial x} \\ 0 & \mu \frac{\partial}{\partial z} & \mu \frac{\partial}{\partial y} \end{bmatrix}$$

с начальными условиями: $U(x, y, z, t)|_{t=0} = 0, V(x, y, z, t)|_{t=0} = 0, W(x, y, z, t)|_{t=0} = 0$.

и граничными условиями: $\sigma_{xz}|_{z=0} = 0, \sigma_{yz}|_{z=0} = 0, \sigma_{zz}|_{z=0} = 0$.

В данной постановке $\lambda(x, y, z), \mu(x, y, z)$ – параметры Ламе.

Предполагается, что плотность ρ зависит от трех пространственных переменных и правая часть (массовая сила) может быть представлена в виде:

$\vec{F}(t, x, y, z) = F_x \vec{i} + F_y \vec{j} + F_z \vec{k}$.

Например, для источника типа «вертикальная сила» получим следующее представление:

$\vec{F}(t, x, y, z) = \delta(x - x_0)\delta(y - y_0)\delta(z - z_0) f(t) \vec{k}$, где (x_0, y_0, z_0) координаты источника.

2.2 Метод решения задачи

Метод решения поставленной задачи основан на использование конечноразностного метода. Алгоритм построения конечноразностной схемы предложен в статье [6]. Аналогичный подход для статических задач теории упругости, также, был развит в работах отечественных математиков [7]. Расчет сеточных коэффициентов (λ, μ, ρ, которые могут иметь разрывы) участвующих в разностной схеме проводится на основе интегральных законов сохранения.

Конечноразностная схема имеет второй порядок аппроксимации по времени и пространству [6]. Общий вид некоторых уравнений конечноразностной схемы следующий:

$$\frac{\rho_{i,j,k} + \rho_{i-1,j,k}}{2} \frac{u^{n+1}_{i-\frac{1}{2},j,k} - u^{n}_{i-\frac{1}{2},j,k}}{\tau} = \frac{(\sigma^{n+\frac{1}{2}}{}_{xxi,j,k} - \sigma^{n+\frac{1}{2}}{}_{xxi-1,j,k})}{\Delta x} + \frac{(\sigma^{n+\frac{1}{2}}{}_{xyi-\frac{1}{2},j+\frac{1}{2},k} - \sigma^{n+\frac{1}{2}}{}_{xyi-\frac{1}{2},j-\frac{1}{2},k})}{\Delta y} +$$

$$\frac{(\sigma^{n+\frac{1}{2}}{}_{xzi-\frac{1}{2},j,k+\frac{1}{2}} - \sigma^{n+\frac{1}{2}}{}_{xzi-\frac{1}{2},j,k-\frac{1}{2}})}{\Delta z} + f^{n}{}_{xi,j,k},$$

$$\frac{\sigma^{n+\frac{1}{2}}{}_{xxi,j,k} - \sigma^{n-\frac{1}{2}}{}_{xxi,j,k}}{\tau} = (\lambda+2\mu)_{i,j,k}\frac{u^{n}_{i+\frac{1}{2},j,k} - u^{n}_{i-\frac{1}{2},j,k}}{\Delta x} + \lambda_{i,j,k}\frac{v^{n}_{i,j+\frac{1}{2},k} - v^{n}_{i,j-\frac{1}{2},k}}{\Delta y} + \lambda_{i,j,k}\frac{w^{n}_{i,j,k+\frac{1}{2}} - w^{n}_{i,j,k-\frac{1}{2}}}{\Delta z},$$

$$\frac{\sigma^{n+\frac{1}{2}}{}_{xyi-\frac{1}{2},j-\frac{1}{2},k} - \sigma^{n-\frac{1}{2}}{}_{xyi-\frac{1}{2},j-\frac{1}{2},k}}{\tau} = c66_{i-\frac{1}{2},j-\frac{1}{2},k}\left(\frac{u^{n}_{i-\frac{1}{2},j,k} - u^{n}_{i-\frac{1}{2},j-1,k}}{\Delta y} + \frac{v^{n}_{i,j-\frac{1}{2},k} - v^{n}_{i-1,j-\frac{1}{2},k}}{\Delta x}\right).$$

Пример расчета взвешенного коэффициента c66 для расчета σ_{xy}:

$$c66_{i-\frac{1}{2},j-\frac{1}{2},k} = \left(\frac{1}{4}\left(\frac{1}{\mu_{i,j,k}} + \frac{1}{\mu_{i-1,j,k}} + \frac{1}{\mu_{i,j-1,k}} + \frac{1}{\mu_{i-1,j-1,k}}\right)\right)^{-1}.$$

Критерий устойчивости данной схемы [6]:

$$\tau \le \frac{1}{Vp_{\max}\sqrt{\frac{1}{\Delta x^{2}} + \frac{1}{\Delta y^{2}} + \frac{1}{\Delta z^{2}}}}.$$

Здесь $\Delta x, \Delta y, \Delta z$ – шаги дискретизации по пространственным переменным, τ – шаг дискретизации по времени; $Vp_{\max}$ – максимальная скорость распространения упругих волн.

Общая схема вычислений выглядит следующим образом. Вначале, на первом полушаге по времени определяются компоненты вектора скорости смещения, затем на втором полушаге по времени по формулам находятся нужные компоненты напряжений. Далее насчитываются новые компоненты образов вектора скорости смещения на новом полушаге по времени.

2.3 Реализация «построителя» трехмерной модели упругой среды

Для проведения численного расчета по представленной методике необходимо иметь трехмерную сеточную модель исследуемой среды. В настоящее время существуют программные средства, позволяющие задавать сеточные 3D модели упругих сред, но они, в своем большинстве находятся в составе больших систем (например, AutoCAD). В некоторых случаях их использование не возможно, по причине их отсутствия или неприспособленности к заданию специфических геометрических структур исследуемых объектов с произвольными параметрами: плотность и модули упругой среды. Поэтому для выполнения данной работы был создан специализированный построитель трехмерных моделей неоднородных упругих сред. Данный инструментарий применяется для построений различных моделей упругих сред, в том числе сред, характерных для грязевых вулканов. С помощью разработанного построителя задаются значения λ, μ, ρ в каждой точке конечноразностной схемы.

В нашем случае предполагается, что задана крупноблочная модель среды, составленная из параллелепипедов, в вершинах которых задаются параметры среды (Vp, Vs, ρ). Эти параметры являются непрерывными внутри каждого блока. Разрывы проходят только по граням соседних параллелепипедов. Далее происходит интерполяция параметров среды на более «мелкую» расчетную сетку.

После того как построена основная сеточная модель трехмерно-неоднородной упругой среды возможно дальнейшее усложнение ее геометрической структуры. В построенную модель можно «вставлять» различные геометрические объекты, которые имеют аналитическое описание (цилиндрические, конические, эллипсоидальные и др. подобласти, или их пересечение) со своими упругими параметрами среды.

Разработанный построитель модели позволяет конструировать сложные 3D модели неоднородных упругих сред, близкие к реальным объектам исследования.

3. Параллельная реализация и исследование времени работы программы

Для численного моделирования распространения упругих волн в трехмерно неоднородных упругих средах, используя приведенную выше разностную схему, проведение полномасштабных вычислительных экспериментов требует значительных вычислительных ресурсов.

Поэтому в данном случае необходимо было провести распараллеливание с целью ускорения вычислений и возможности расчета реальных моделей упругих сред. В настоящее время, в ИВМиМГ СО РАН, имеются различные многоядерные вычислительные комплексы, как вычислительные сервера с общей памятью, так и вычислительные кластера, где на одном вычислительном узле находится несколько многоядерных процессоров. Отметим кластер на базе процессора Intel Xeon E5450, 3.0 ГГц и кластер на базе процессора Itanium 2, 1.6 ГГц. Все они могут использоваться для решения больших геофизических задач, связанных с численным моделированием.

Одно из основных требований к параллельной программной реализации состояло в том, чтобы программа могла бы эффективно работать на однородных кластерах с различным количеством и типами процессоров расположенных на узле.

Необходимо отметить, что выбор способа декомпозиции расчетной области во многом обусловлен «шаблоном» вычислительной схемы. В данной работе для решения мы используем явную конечно-разностную схему, где для расчета конкретного узла необходима информация только о соседних узлах конечно-разностной схемы.

Известно несколько подходов для распараллеливания (декомпозиции области) 3D разностных схем. Для распараллеливания данной задачи нами были рассмотрены два их них. Первый состоит в разбиении исходной вычислительной модели на «кубики». Второй основан на декомпозиции области на слои вдоль направления одной из координатных осей (в нашем случае выбрано разбиение вдоль оси Z). При реализации данных схем каждый вычислительный узел рассчитывает свою сеточную область на каждом временном шаге независимо от других, за исключением точек, находящихся на границе между двумя соседними областями. Эти точки являются общими для каждой из областей и для продолжения счета необходимо производить обмен информацией об искомых величинах между «соседями». В первом случае, возможно, есть большая универсальность и некоторая гибкость метода, но в то же время его сложно реализовывать программно и необходимо производить обмены по каждой из граней между соседними «кубиками». Во втором случае, резко сокращается количество обменов, поскольку их необходимо производить только между соседними слоями, но объем передаваемой каждый раз информации больше, нежели в первом случае.

Основной причиной, по которой был выбран второй способ, является простота реализации, и сравнительно небольшое количество обменов.

На основе выбранной схемы были созданы две параллельные программы, одна, где для распараллеливания используется только MPI и вторая, где используется комбинация возможностей MPI и OpenMP. Во втором случае («гибридная параллельная схема»), предлагается проводить обмен информацией между соседними слоями через MPI, а внутри каждого слоя, расположенного в общей памяти узла, проводить параллельные вычисления, используя OpenMP. Количество слоев для «гибридной» схемы определяется количеством свободных вычислительных узлов, а количество OpenMP потоков – количеством ядер на узлах; для MPI программы – общим количеством ядер на выделенных вычислительных узлах. Также число слоев зависело и от требований к оперативной памяти для проведения необходимых расчетов. На каждом временном шаге моделирования необходимо было произвести две серии обменов информацией о волновом поле: одна – для поля скоростей смещений, вторая – для компонент тензора напряжений. Весь обмен информацией реализован через интерфейс MPI с помощью блокирующих операций получения и передачи данных.

Зависимость времени, затраченного на моделирование, от количества используемых ядер исследовалось на небольших задачах, поэтому проведенные тестовые расчеты являются своего рода "ориентиром". С целью сравнения времени расчета (рис.1, рис.2) были выбраны две тестовых трехмерных модели среды – M1, M2. Параметры расчетной области (количество узлов в

разностной сетке по координатам X,Y.Z) для каждой модели приведены в таблице 1. Также было проведено сравнение времени работы программы на различных типах вычислительных серверов hp BL2x220c G5 и hp BL2x220c G6 для представленных моделей M1 и M2 (рис.1). Расчеты также проводились на кластере НКС-30Т (ССКЦ ИВМиМГ СО РАН). Для компиляции программы на кластере использовался штатный компилятор Intel(R) Fortran, рис. 2.

Первые результаты численных экспериментов для различных расчетных моделей упругих сред показали, что реализация второго подхода (комбинация возможностей MPI и OpenMP), при использовании штатных компиляторов, оказалась неэффективной. При использовании 32 ядер (4узла) в среднем в 2.8 раза медленнее. Полученные результаты не вызывают удивления поскольку при организации параллельных вычислений средствами OpenMP, на отдельном узле нужно уметь прогнозировать распределение памяти при расчете, поскольку потоки при обращении к памяти могут использовать данные из памяти соседнего процессора. Очевидно, что в этом случае требуется индивидуальная настройка программы на свойства процессоров расположенных на узле. Поскольку MPI программы обладают некоторой универсальностью и переносимы на кластеры с любой архитектурой, то основные усилия по оптимизации программы были направлены на реализацию алгоритма с помощью библиотеки MPI.

Представленные результаты могут служить некоторой оценкой для определения времени работы параллельной программы уже на больших вычислительных задачах, требующих большего количества ресурсов.

Таблица 1. Параметры тестовых 3D моделей

Описание 3D моделей для расчетов		
Модель	**M1**	**M2**
Размеры расчетной области (кол-во узлов):		
X	201	301
Y	161	241
Z	221	331
Time (количество шагов по времени)	650	975

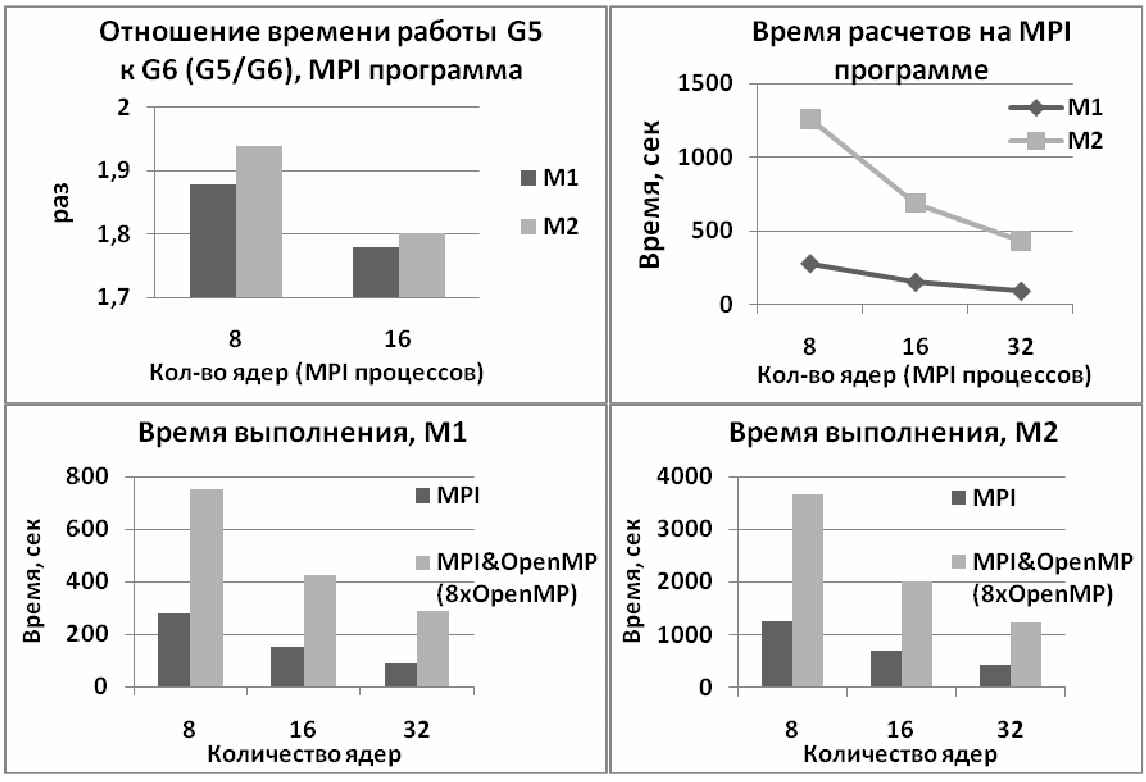


Рис.1. Результаты сравнения времени работы программ

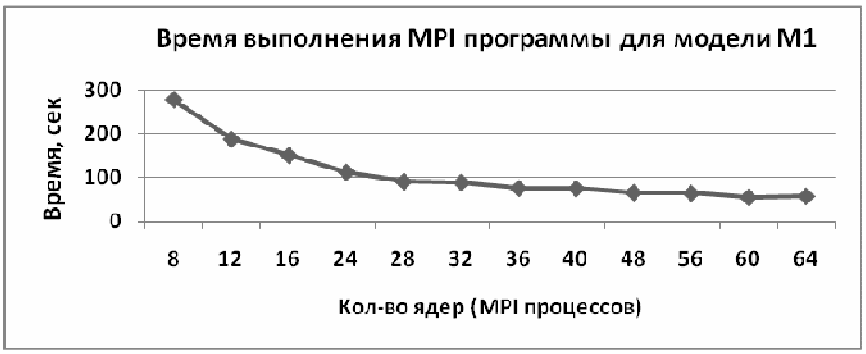


Рис.2. Зависимость времени работы MPI программы от количества ядер на кластере НКС-30Т.

4. Экспериментальные исследования грязевого вулкана «Гора Карабетова» вибросейсмическими методами.

В данном разделе кратко описывается эксперимент, проведенный на грязевом вулкане с применением вибросейсмического источника и сейсмических регистраторов RefTek и РОСА. Анализируется спектральный состав волнового поля, динамика его поведения для профиля, пересекающего вулкан.

В экспериментальных работах по активному вибросейсмическому просвечиванию грязевого вулкана «Гора Карабетова» использовался сейсмический вибрационный источник СВ-10/180 и регистрирующие комплексы RefTek-125A (40 регистраторов с вертикальными сейсмоприемниками GeoSpace GS-20DX) и РОСА (18 каналов с трехкомпонентными датчиками GeoSpace и СМЕ-3011). Общая схема зондирования вулкана приведена на рис. 3.

Для определения строения и скоростных характеристик вмещающей среды была проведена регистрация волнового поля вибраторов на профиле вне зоны вулкана. Излучение сигналов вибраторами осуществлялось в точках вдоль профиля с применением методики сейсморазведки на отраженных волнах и метода общей глубинной точки.

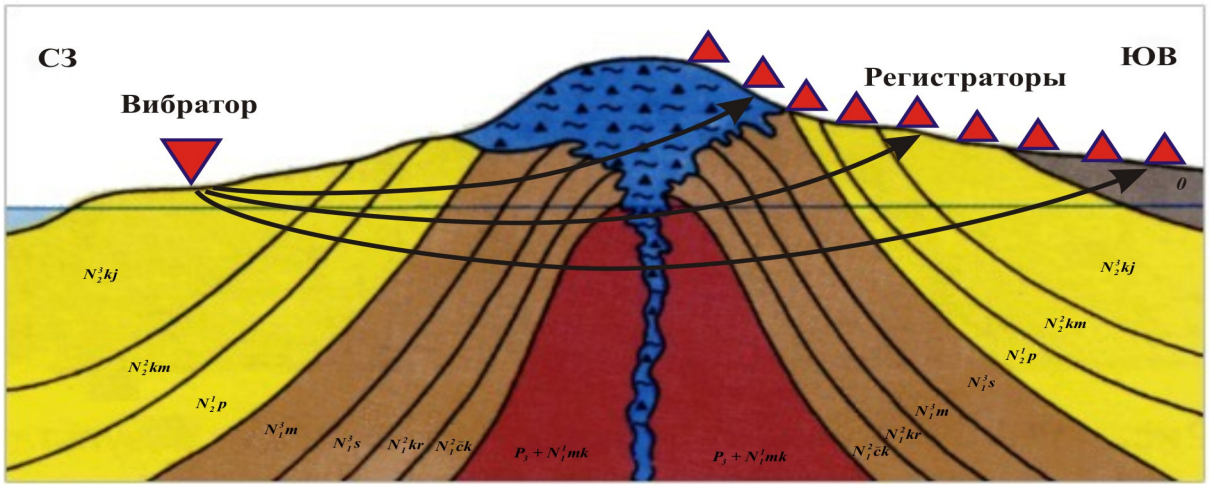


Рис.3. Геологический разрез и схема вибросейсмического зондирования грязевого вулкана «Гора Карабетова».

С этим же расположением точек излучения сигналов вибратором был отработан поперечный профиль регистрации длиной 3.2 км пересекающий вулкан. Поперечный профиль через вулкан отрабатывался в два приема, с использованием 37 и 31 регистраторов RefTek. Именно этот поперечный профиль с точками излучения на концах профиля будет анализироваться в данной работе.

В качестве зондирующих сигналов вибратора использовались свип-сигналы в диапазоне частот 10 – 64 Гц (сигнал с линейной разверткой частоты в данном диапазоне с длительностью

60 с). Количество зондирований в одной точке составляло 5, 10 и 20 для различных точек. Это позволило поднять помехоустойчивость вибрационных сейсмограмм (получаемых путем корреляционной свертки излучаемого и принятого сигналов) в условиях регистрации при повышенных сейсмических шумах. Регистрация излучаемого сигнала велась в режиме непрерывного времени с записью файлов волновых форм длинной 3300 с. Синхронизация по времени регистрирующих систем и вибратора осуществлялась при помощи GPS-приемников. Одним из сейсмоприемников RefTek осуществлялась регистрация сигнала от генератора сигналов вибратора СВ-10/100. Запись формы излучаемого вибратором сигнала велась с целью использования последнего в качестве опорного при вычислении вибрационных сейсмограмм в режиме зондирования широкополосными сигналами, что позволило избежать ошибок временной синхронизации. Второй регистратор RefTek использовался для записи колебаний грунта в непосредственной близости от вибратора.

Полученный полевой материал обрабатывался с помощью специальных программ. Были получены корреляционные сейсмограммы, пример приведен на рис. 4 и проекции спектрально-временных функций, пример рис. 5.

Временная структура волнового поля характеризуется усложнением по мере приближения к телу вулкана. Одиночные волновые формы на начальном участке зондирования разрастаются до нескольких цугов волн в районе вулкана и далее за ним. Соответственно их длительность при этом возрастает примерно на порядок. Усложнение структуры волнового поля во временной области сопровождается изменениями в спектральной области.

Спектральный анализ вибрационных сейсмограмм (коррелограмм) на этом профиле показывает довольно сложную картину прохождения сейсмических волн через тело вулкана.

На расстояниях до 1800 метров от источника отчетливо просматривается поверхностная волна со скоростью около 340 м/с, имеющая спектральные пики на частотах 12–15 Гц.

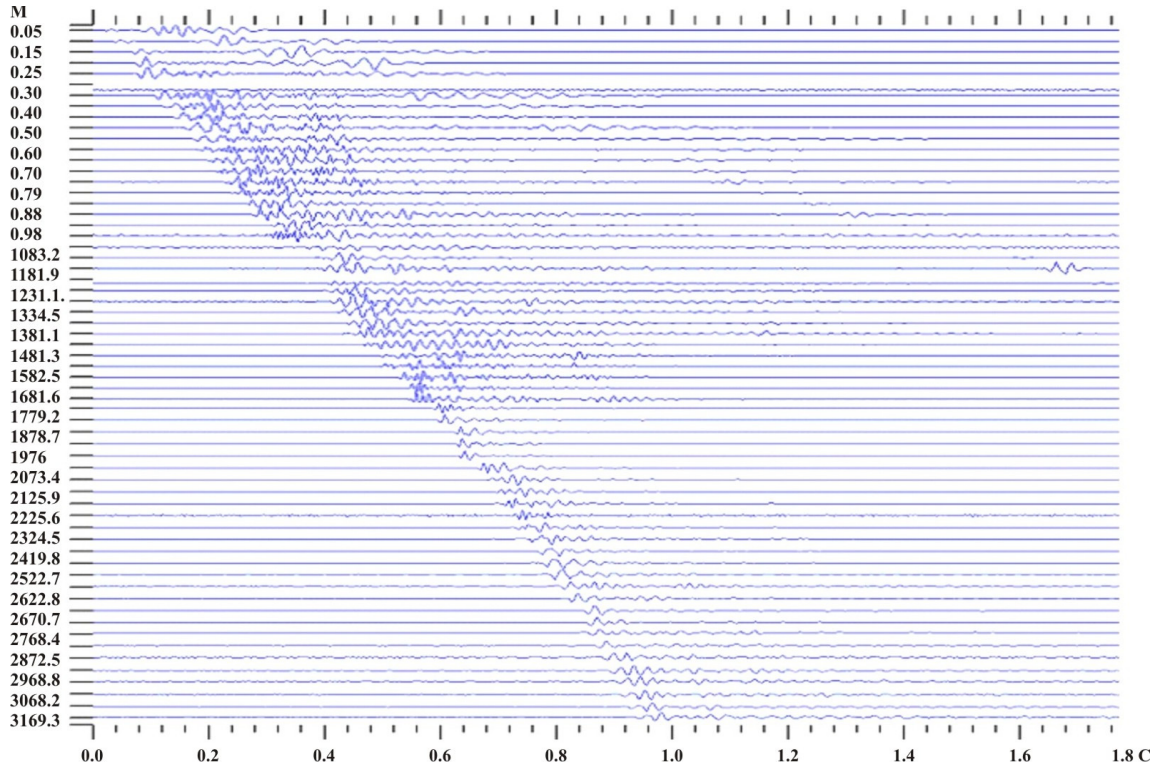


Рис.4. Пример вибрационных сейсмограмм, полученных на профиле, пересекающем вулкан. По вертикали расстояние регистратора от источника в метрах, по горизонтали время регистрации в секундах.

По мере приближения к центру вулкана спектральная картина начинает меняться. Появляются более высокие частоты, спектр существенно расширяется (рис. 5), на некоторых расстояниях появляются узкополосные пики, возможно, связанные с резонансными свойствами скопления ранее действовавших сальз и грифонов (выходящие на поверхность каналы в виде конусов – грифоны или небольших углублений - сальзы, через которые извергаются грязевые и га-

зообразные фракции вулкана). Однако, в центре вулкана, на расстоянии 1480 метров от источника появляется узкополосный спектральный пик на частоте 25–28 Гц (рис. 6), который может быть связан с геометрией центрального канала трубки вулкана и с резонансными свойствами этого выводящего канала. Для подтверждения этой гипотезы было проведено численное моделирование, по предложенной вычислительной схеме, ориентированное на изучение геометрических параметров вулкана «Гора Карабетова», результаты которого приведены в следующем разделе.

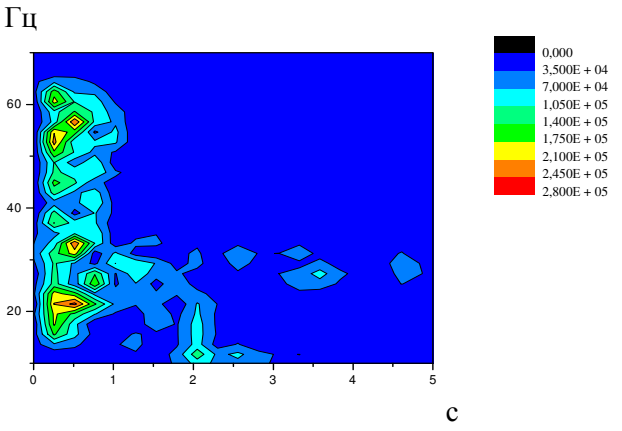


Рис.5. Проекция СВФ на удалении 740 м от источника (район скопления грифонов)

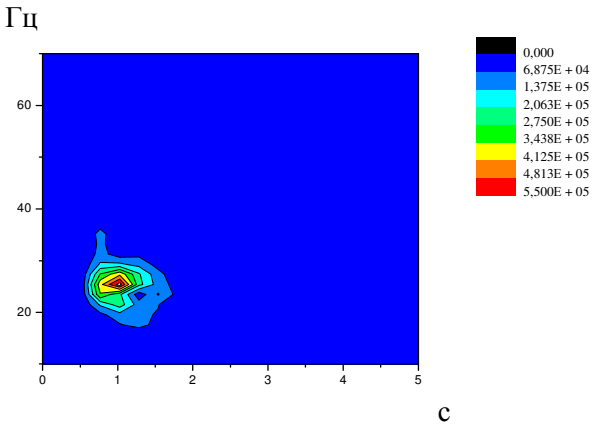


Рис.6. Проекция СВФ на удалении 1480 м от источника (центр вулкана)

На этих рисунках по горизонтали обозначено время в секундах, по вертикали частота в Герцах, цветовая шкала отображает амплитуду спектра в относительных единицах.

5. Результаты численного эксперимента на модели «Гора Карабетова»

Результаты полевых экспериментов показали, что структура грязевых вулканов имеет сложную геометрию и неоднородное строение среды, в которой содержится жидкость, пузыри газа, неоднородные включения и т.д. Соответственно все это оказывает влияние на структуру волнового поля наблюдаемого в экспериментах.

Карабетовская антиклинальная зона, в пределах которой размещены грязевые вулканы Карабетовой горы, горы Чиркова, Северо-Нефтяной и Дубовый Рынок. В строении антиклинали принимали участие отложения майкопской серии, чокракского, караганского, сарматского, мэотического, понтического, киммерийского и куяльницкого ярусов. Майкопская толща 3 и более километров [1].

Строение характерно для большинства диапировых антиклиналей Таманского полуострова. Ядро складок прорвано сильно перемятыми пластичными глинами майкопского периода. Более молодые слои вблизи ядра залегают обычно круто, по мере удаления от оси складок они выполаживаются, а мощность их увеличивается. Однако в своде собственно Карабетовской складки приповерхностная часть ядра не прорвана майкопскими глинами. Глубина залегания майкопского диапирового ядра около 1км, северо-северо-восточное простирание антиклинали.

Предполагается, что майкопская серия является решающей в формировании геологических структур, ответственных за возникновение грязевых вулканов. [8]. По результатам микросейсмического зондирования (там же) под вулканом выделена относительно узкая, вертикальная низкоскоростная зона, ассоциируемая с насыщенным флюидоми подводящим каналам. Область питания для подводящего канала по данным эксперимента находится на глубине 4,5–9 км и возможно продолжается до глубины более 15 км. Следует отметить, что кристаллический фундамент находится также на глубине 15 км.

В данной работе мы проводим численное моделирование, цель которого представить влияние геометрии модели на структуру волнового поля, т. е. не учитываются особенности связанные с учетом присутствия разнородного строения среды с включением различных неоднородностей, хотя программа позволяет проводить расчеты такого вида. В этой связи, на первом эта-

пе, мы рассматриваем простые модели, которые позволяют исследовать основные особенности волнового поля. Для исследования структуры волнового поля, получаемой при проведении полевых экспериментов на грязевом вулкане «Гора Карабетова» были проведены тестовые расчеты для различных моделей сред, одна из которых представлена далее.

Моделируется трехслойная среда с цилиндрическим включением (рис. 7, слева) со следующими параметрами:

Слой 1 – Vp = 2.0 км/с, Vs = 1.0 км/с, ρ = 2.55 г/см^3

Слой 2 – Vp = 2.7 км/с, Vs = 1.35 км/с, ρ = 2.75 г/см^3

Слой 3 – Vp = 2.5 км/с, Vs = 1.25 км/с, ρ = 2.55 г/см^3

Цилиндр (4) – Vp = 1.7 км/с, Vs = 0.85 км/с, ρ = 0.75 г/см^3

Где Vp и Vs – скорости распространения продольных и поперечных волн соответственно, ρ – плотность упругой среды.

Источник – «центр давления», несущая частота 25 Гц, расположен вблизи свободной поверхности (белый треугольник, рис. 7,слева).

Система наблюдения – четыре линии сейсмоприемников на свободной поверхности (рис. 7, справа).

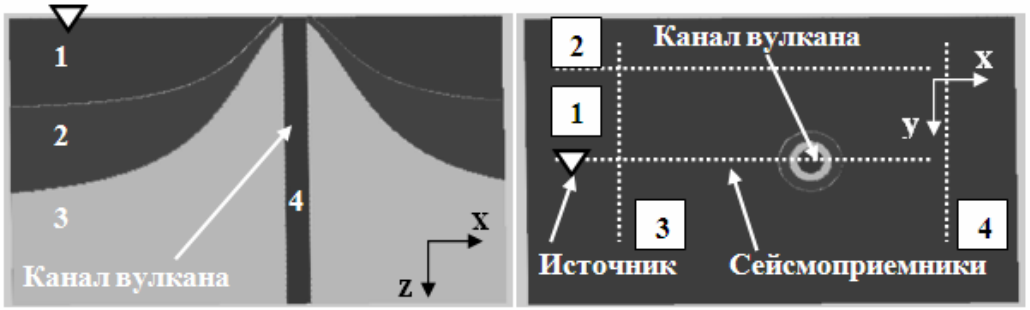


Рис.7. Геометрическая структура 3D модели упругой среды в различных плоскостях сечений

Численное моделирование для 3D модели (рис.7) проведено на вычислительных блэйд-серверах hp ProLiant BL2x220c G5, находящихся в составе НКС-30Т (ССКЦ ИВМиМГ СО РАН). Некоторые параметры расчета приведены в табл.2.

Таблица 2. Параметры расчета

Кол-во узлов расчетной сетки	
по оси X	1677
по оси Y	1059
по оси Z	971
по времени	10313
Кол-во используемых ядер (MPI процессов)	160
Время, затраченное на моделирование	31ч15м17с (112517сек)

Результаты моделирования в виде теоретических сейсмограмм и мгновенных снимков волнового поля представлены на рис.8 и рис.9.

Проведенные расчеты показывают, что при прохождении сейсмической волны через область трубки на профиле 1 (рис. 9), пересекающем ее, отчетливо видна вертикальная полоса соответствующая дилатансной зоне грязевого вулкана. Медленное затухание поля, показанное на рис. 8 (t_4), связанно с резонансными проявлениями, обусловленное геометрией изучаемого объекта, что полностью согласуется с экспериментальными данными.

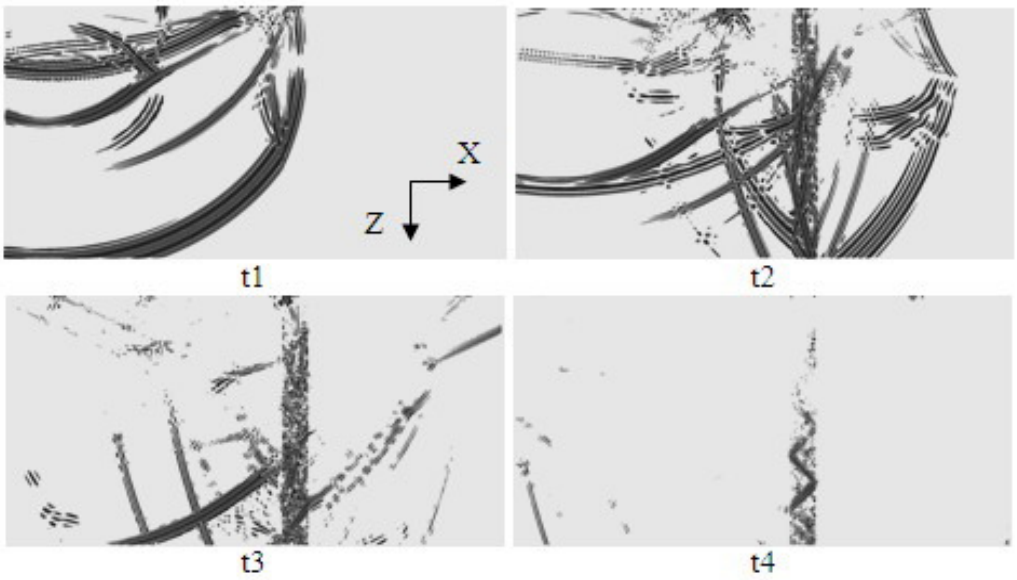


Рис.8. Снимки W компоненты волнового поля в различные промежутки времени для профиля 1.

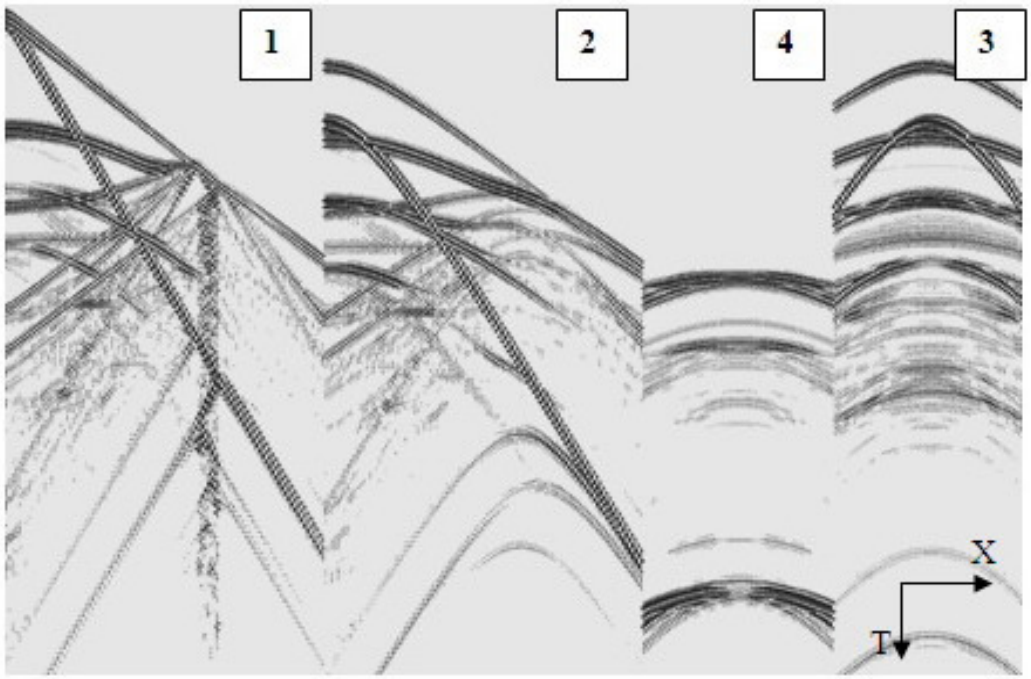


Рис.9. Теоретические сейсмограммы, W компонента волнового поля для профилей 1,2,3,4.

Заключение

Анализ СПФ по профилю, пересекающему зону вулкана «Гора Карабетова» и численное моделирование, основанное на геометрических особенностях этого вулкана, показало следующее:

- в спектрах вибрационных сейсмограмм на фоне их широкополосной части появляются узкополосные составляющие, характеризующие селективные свойства среды с выраженной неоднородностью, вероятно, это связанно с мелкими подводящими каналами, питающими действующие сальзы и грифоны и резонансными свойствами этих каналов на соответствующих частотах. Особенно выделяется узкополосный пик над центральной частью вулкана на частотах 25–28 Гц. Проведенные чис-

ленные расчеты также подтверждают, что этот пик может быть связан с геометрией центрального канала вулкана.

- спектры сейсмограмм непосредственно в области вулкана характеризуются двумя особенностями – с одной стороны смещением узкополосных пиков в область более высоких частот на компонентах Z, а также расширением спектров колебаний в область высоких частот. Такие явления связаны с трансформацией спектра на трещиноватых и флюидонасыщенных структурах тела вулкана, обусловленное нелинейными процессами распространения волн в зонах разрушения горных пород.

Для моделирования структур, характерных для грязевых вулканов создан специализированный построитель трехмерных моделей неоднородных упругих сред. Разработанный построитель модели позволяет конструировать сложные 3D модели неоднородных упругих сред, близкие к реальным объектам исследования.

С применением данного построителя был проведен численный эксперимент для заданных параметров геофизической модели вулкана, расчет с помощью параллельной программы проведен на вычислительном кластере НКС-30Т.Описан способ распараллеливания вычислений для представленной задачи.

Приведены результаты тестовых расчетов, показывающие время работы на различном количестве используемых ядер для различных параллельных реализаций программы.

Литература

1. Грязевые вулканы Керченско-Таманского региона /Шнюков Е.Ф., Шереметьев В.М., Маслаков В.А. и др. – Краснодор: ГлавМедиа, 2006 .– С. 176.
2. Б.М. Глинский, А.Л. Собисевич, А.Г. Фатьянов, М.С. Хайретдинов. Математическое моделирование и экспериментальные исследования грязевого вулкана Шуго. // Вулканология и сейсмология, 2008 №4, С. 1-9.
3. А.С. Алексеев, Б.М. Глинский, А.Л. Собисевич, В.В. Ковалевский, М.С. Хайретдинов и др. Активная сейсмология с мощными вибрационными источниками: Отв. ред. Г.М. Цибульчик. – Новосибирск: ИВМиМГ СО РАН, Филиал "Гео" Издательства СО РАН, 2004. – С. 387.
4. Глинский Б.М., Собисевич А.Л., Хайретдинов М.С. Опыт вибросейсмического зондирования сложно построенных геологических структур (на примере грязевого вулкана Шуго). // Докл. РАН. - 2007. - Т.413, №3. - С.398-402.
5. Алексеев А.С., Глинский Б.М., Имомназаров Х.Х., Ковалевский В.В., Собисевич Л.Е., Хайретдинов М.С., Цибульчик Г.М. Мониторинг геометрии и физических свойств «поверхностной» и «очаговой» дилатансных зон методом вибросейсмического просвечивания сейсмоопасных участков земной коры. // Коллективная монография «Изменение окружающей среды и климата. Природные и связанные с ним техногенные катастрофы». – М.: ИФЗ РАН, 2008. – Т.1: Сейсмические процессы и катастрофы, Ч.2. – С. 179-223.
6. Караваев Д.А. Параллельная реализация метода численного моделирования волновых полей в трехмерных моделях неоднородных сред. //Параллельные вычислительные технологии (ПаВТ’2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). – Челябинск: Изд. ЮУрГУ, 2009. , С. 196-204
7. Коновалов А.Н. Сопряженно-факторизованные модели в задачах математической физики // Сиб.журн.вычисл.математики / РАН. Сиб.отд-ние.- Новосибирск, 1998. Т.1, № 1. С.25-57
8. Горбатиков А.В. и др. Технология глубинного зондирования земной коры с использованием естественного низкочастотного микросейсмического поля. // Коллективная монография «Изменение окружающей среды и климата. Природные и связанные с ним техногенные катастрофы». – М.: ИФЗ РАН, 2008. – Т.1: Сейсмические процессы и катастрофы, Ч.2. – С.221-236.

Отечественная коммуникационная сеть 3D-тор с поддержкой глобально адресуемой памяти для суперкомпьютеров транспетафлопсного уровня производительности*

А. А. Корж, Д. В. Макагон, А. А. Бородин, И. А. Жабин, Е. Р. Куштанов, Е. Л. Сыромятников, Е. В. Черемушкина

Межузловая коммуникационная сеть является одной из важнейших частей суперкомпьютера, определяющей его производительность и масштабируемость. В статье рассматриваются детали реализации и первые результаты макетирования разработанной в НИЦЭВТ межузловой коммуникационной сети с топологией 3D-тор. Данная сеть может эффективно применяться как в вычислительных кластерах небольшого и среднего размера, так и в суперкомпьютерах транспетафлопсного уровня производительности. Основными особенностями сети являются аппаратная поддержка глобально адресуемой памяти и устойчивость к сбоям. В статье описывается архитектура интерфейса маршрутизатора с вычислительным узлом и проблемы, которые были решены при его разработке. Программисту аппаратная поддержка глобально адресуемой памяти предоставляется посредством библиотеки параллельного программирования SHMEM, реализации которой уделено особое внимание. Приводятся результаты, полученные на кластере из шести узлов макета второго поколения M2 (2008–2009 гг.), и первые результаты макета третьего поколения M3 (2009–2010 гг.).

1. Введение

Межузловая коммуникационная сеть является одной из важнейших частей суперкомпьютера, определяющей его производительность и масштабируемость. Разрыв тактовых частот процессора и подсистемы локальной и удалённой памяти постоянно увеличивается, поэтому именно время выполнения обращений в память удалённого узла становится «узким местом» при распараллеливании большинства задач на кластерах и суперкомпьютерах.

Коммуникационные сети современных суперкомпьютеров можно разделить на два класса: коммерческие (Ethernet, Infiniband, Quadrics, Myrinet) и заказные (IBM BlueGene, Cray XT, SGI Altix UV).

Коммерческие сети разрабатываются для широкого спектра применений, использование их в качестве коммуникационной сети суперкомпьютеров — лишь одно из них. В связи с этим в жертву универсальности приносится производительность в ряде режимов, в частности, в коммерческих сетях не оптимизирована передача коротких сообщений, нет адаптивной передачи, аппаратно не поддержана отказоустойчивость при отказах линков.

Для систем среднего уровня (до 1000 узлов), а также для вычислительных кластеров, используемых в режиме счета нескольких небольших задач, умещающихся по потреблению памяти в одной стойке, чаще всего именно фактор цены оказывается важнее возможности эффективно считать одну задачу на всем суперкомпьютере. Как правило, для таких систем применяются коммерчески доступные коммуникационные сети, среди которых в последнее время с огромным отрывом лидируют сети стандарта Infiniband.

Для суперкомпьютеров с большим числом узлов (10–100 тысяч), достигших в настоящее время петафлопсного уровня производительности, вопрос сбалансированности производительности сети и процессоров является критическим, поэтому применение коммерческих технологий, таких как Infiniband, оказывается неприемлемым.

В связи с этим, крупнейшие производители суперкомпьютеров, такие как Cray, IBM и SGI, используют коммуникационные сети собственной разработки: Cray Seastar2+ и будущая сеть машин серии Cray Baker — Gemini High-Speed Network, IBM Bluegene Torus Network, SGI NU-

* Работа выполнена при поддержке РФФИ, грант № 09-07-13596-офи_ц.

MAlink. Такие сети недоступны для коммерческой продажи и используются исключительно в машинах высшего эшелона производительности.

В таких системах в настоящее время используются преимущественно сети с топологией 3D-тор [4]. Основные причины этого — трёхмерность коммуникационного шаблона ряда физических задач, простая расширяемость системы, регулярность упаковки сети по стойкам с учетом экономии на межстоечных кабелях, хорошая масштабируемость и т. д.

При всех достоинствах топологии 3D-тор, она становится малоэффективной, когда число вычислительных узлов измеряется многими десятками тысяч (не говоря уже о сотнях тысяч). Переход к торам большей размерности хотя и увеличивает производительность (вместе со стоимостью), однако требует заметно более сложного способа упаковки и плохо соотносится с коммуникационными шаблонами физических задач.

В связи с этим, в последнее время широко исследуются альтернативные топологии, в частности, различные варианты модифицированных деревьев (fat-tree) и сетей Клоса. Так, в числе последних разработок фирмы Cray — коммутатор YARC (для суперкомпьютера Cray X2), в ближайшем будущем планируется выпуск коммутатора Cray Aries (предположительно, с топологией dragonfly [5]).

Важным аспектом, определяющим применение заказных сетей, является поддержка глобально адресуемой памяти как основного режима программирования стратегических суперкомпьютеров. Данная парадигма может эффективно применяться при использовании обычных коммерческих микропроцессоров [1].

На отечественном рынке на данный момент присутствуют только зарубежные коммерческие разработки, не подходящие для суперкомпьютеров высшего эшелона производительности. Более того, каналы импорта компонентов зарубежных коммерческих сетей сравнительно легко могут быть перекрыты, чего с процессорами или памятью сделать практически невозможно. В связи с этим в последние несколько лет НИЦЭВТ ведет разработку заказной высокоскоростной коммуникационной сети в рамках проекта разработки суперкомпьютера стратегического назначения (СКСН).

Разрабатываемой коммуникационной сети уделяется особое внимание, как одному из критических компонентов СКСН. На данный момент изготовлен, налажен и находится в опытной эксплуатации шестиузловой вычислительный кластер, построенный на основе второго поколения макетных образцов маршрутизаторов высокоскоростной коммуникационной сети с топологией 2D-тор на основе ПЛИС. Его тестирование специалистами ряда организаций показало хорошие результаты, сопоставимые с образцами коммерческой сети Infiniband DDR. При этом в наиболее тяжёлом режиме работы с интенсивным обменом короткими пакетами сообщений достигнутые характеристики существенно превосходили сеть Infiniband, в том числе и последнего поколения Infiniband QDR 4x.

В настоящее время изготовлен и находится в стадии отладки макетный образец третьего поколения, построенный на основе наиболее современных ПЛИС фирмы Xilinx; его характеристики в 1.8 раз лучше, достигаемый темп передачи сообщений — 14.9 миллионов пакетов в секунду.

В маршрутизаторе на аппаратном уровне поддерживаются четыре операции с удалённой памятью: асинхронное чтение (get), асинхронная запись (put), атомарное сложение (add), атомарное исключающее ИЛИ (xor), а также различные методы синхронизации и контроль возврата выданных запросов на асинхронное чтение.

Реализована стандартная библиотека Cray SHMEM с дополнениями, позволяющими более эффективно выполнять типовые операции, и библиотека MPI на основе MPICH.

Далее в статье приводится описание разработанного макетного образца маршрутизатора и методология написания параллельных программ на языках C/C++ и Fortran с использованием аппаратно поддерживаемой в макете глобально адресуемой памяти. В заключении приведены основные полученные на текущий момент результаты и сформулированы планы на будущее.

2. Архитектура маршрутизатора

Разработка высокоскоростной коммуникационной сети в НИЦЭВТ ведётся уже несколько лет в рамках проекта создания СКСН. В 2007-м году был изготовлен полнофункциональный

макет M2, состоявший из шести узлов, соединённых в топологию 2D-тор. Пропускная способность межузловых линков — 6.25 Гбит/с в каждую сторону, интерфейс с процессором — PCI-Express x4 (10 Гбит/с).

Обобщённая структурная схема разработанного маршрутизатора приведена на рис. 1. В его состав входят 4 линка (Link) — по два на каждое измерение тора, кроссбар (Crossbar), интерфейс с процессором вычислительного узла (NI+PCI) и сервисный процессор (Service PowerPC).

Каждый линк содержит в ПЛИС блок сериализатора-десериализатора и состоит из входной и выходной частей. Входная часть включает блоки виртуальных каналов и арбитры.

В маршрутизаторе реализован алгоритм бездедлоковой детерминированной маршрутизации, основанный на правилах «пузырька» и «порядка направлений» [3].

С помощью виртуальных каналов реализованы две независимые подсети для запросов и ответов, чтобы избежать дедлоков «процессор — сеть» (запросы не должны задерживать доставку ответов). Аппаратно поддерживается обход отказавших узлов (алгоритм «нестандартного первого и последнего шага») [2].

Выходная часть реализует надёжный протокол доставки сообщений, обеспечивающий непрерывную передачу пакетов и гарантирующий повтор в случае ошибок на линии.

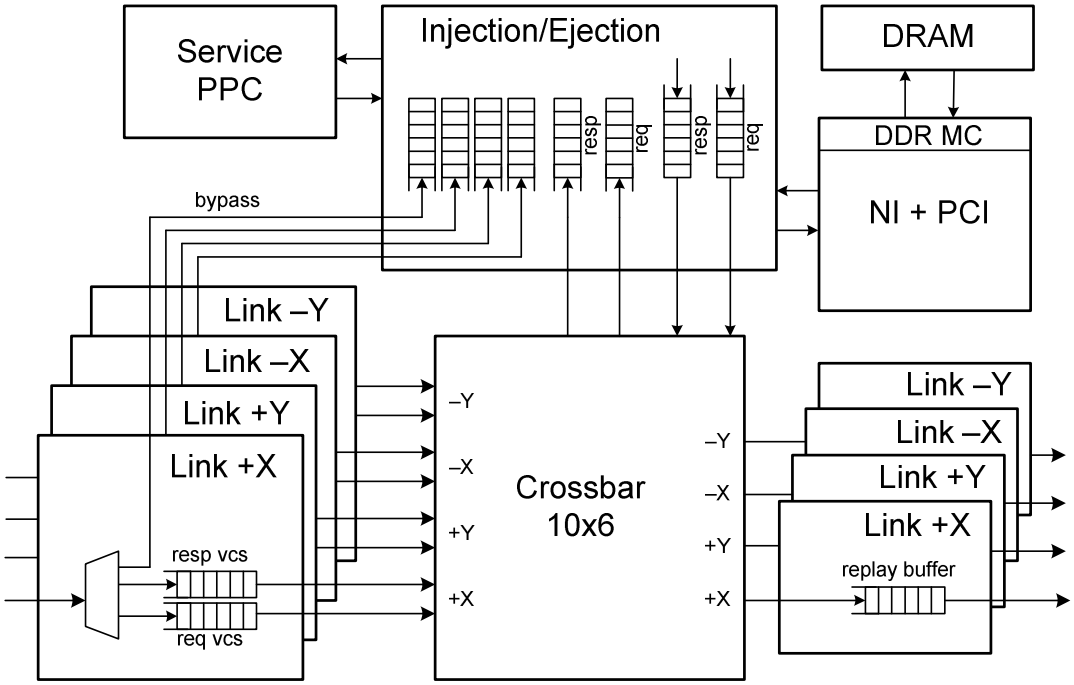


Рис. 1. Обобщённая структурная схема разрабатываемого в НИЦЭВТ маршрутизатора.

Управление потоком данных осуществляется с помощью передачи кредитной информации (кредитов). Пакет передаётся следующему узлу, только если в принимающем буфере соответствующего виртуального канала гарантировано наличие необходимого свободного места. Такая гарантия предоставляется с помощью отсылки «кредитов»: как только в некотором входном буфере маршрутизатора освобождается место (один пакет передаётся в кроссбар и далее — соседнему узлу), по линку, связывающему данный входной буфер с выходным буфером другого маршрутизатора, отсылается «кредит» — информация об освободившемся месте. Основываясь на этой информации, в каждом маршрутизаторе определяется возможность передачи пакетов по линкам в каждый момент времени. Кредиты гарантируют наличие определённого количества свободного места (флитов), при этом за время передачи кредита свободное место может увеличиться (но гарантированно не уменьшится).

Предотвращение сбоев при передаче пакетов по линку обеспечивается собственным протоколом надёжной передачи, реализованным поверх Xilinx Aurora link-layer protocol [6]. В случае несовпадения контрольной суммы при передаче пакета от одного маршрутизатора другому, пакет запрашивается повторно. В случае долговременного выхода из строя ряда линков, воз-

можна перенастройка таблиц маршрутизации сети и последующая перемаршрутизация пакетов. В сетевом интерфейсе содержится настраиваемая сервисной подсистемой таблица маршрутизации, позволяющая задавать различные маршруты между узлами.

Интерфейс с процессором вычислительного узла включает IP-ядро стандартного интерфейса PCI Express или HyperTransport, в зависимости от варианта исполнения маршрутизатора. Основной функцией интерфейса является инжекция пакетов в сеть. Для каждого пакета в автоматическом режиме на основании данных из встроенной таблицы маршрутизации определяется физический адрес и направление передачи. При приёме пакетов из сети интерфейс с процессором извлекает из полученных пакетов данные и записывает их напрямую в оперативную память вычислительного узла с использованием технологии прямого доступа в память (DMA). В некоторых случаях для повышения эффективности пакеты сначала накапливаются в памяти маршрутизатора (режим Write combining), а уже затем записываются в память узла.

Для реализации подсистемы отказоустойчивости, позволяющей корректно обрабатывать отказы отдельных линков и даже вычислительных узлов, использовано встроенное в ПЛИС процессорное ядро, выполняющее функции коммуникационного процессора, следящего за состоянием сети и, при отказах, осуществляющего изменение данных в таблице маршрутизации.

Макетный образец третьего поколения, изготовленный в 2009 году, имеет пропускную способность линка 13 Гбит/с, интерфейс с процессором PCI Express x8. Поддерживается топология трёхмерный тор (используются 6 линков), адаптивная маршрутизация и агрегация коротких сообщений от разных узлов. Задержка между соседними узлами не превышает 1.2 мкс.

На аппаратном уровне поддерживаются следующие типы операций с удалённой памятью: асинхронное чтение (get), асинхронное запись (put), атомарное сложение (add) и атомарное исключающее ИЛИ (xor), а также аппаратный контроль возврата выданных запросов на асинхронное чтение.

Для эффективной реализации операций с удалённой памятью потребовалось снизить накладные расходы, вносимые трансляцией адресов для поддержки виртуальной памяти. Было решено статически выделить область физической памяти, с которой может работать как маршрутизатор, так и пользовательская задача. Это позволило упростить интерфейс маршрутизатора, убрав громоздкие и неэффективные схемы трансляции адресов, присутствующие в коммерческих коммуникационных сетях, таких как Infiniband.

Одной из главных задач, поставленных перед разработчиками маршрутизатора для СКСН, было получение высокой пропускной способности на коротких пакетах. Для этого были использованы несколько виртуальных каналов; при передаче через PCI Express был применён режим Write Combining при посылке пакетов. При приёме пакетов была добавлена возможность агрегации коротких сообщений во встроенной памяти на плате маршрутизатора (через шину PCI данные сбрасываются большими блоками). На задачах типа умножения разреженной матрицы на вектор (SpMV) данные решения оказались довольно эффективными.

3. Программное обеспечение

Коммуникационная сеть, разработанная в НИЦЭВТ, имеет развитую программную поддержку, включающую реализацию стандартной библиотеки SHMEM (версии фирмы Cray с дополнениями, позволяющими более эффективно выполнять типовые операции) для языков C/C++ и Fortran. Также реализована версия библиотеки MPI 1.1 (MPICH), отлажена базовая реализация MPI-2, проведено функциональное тестирование с помощью пакета Intel MPI Benchmarks 3.2 (с проверкой на корректность доставки сообщений).

Поддерживается написание параллельных программ, использующих одновременно и MPI, и OpenMP, и SHMEM. Рекомендуется гибридный режим программирования SHMEM+OpenMP и SHMEM+OpenMP+MPI.

3.1 Библиотека SHMEM

Модели параллельного программирования можно разделить на два класса, в зависимости от того, на коммуникациях какого типа они основаны: двусторонних (передача сообщений) либо односторонних (обращения к удалённой памяти).

В двусторонних коммуникациях активное участие принимают две стороны: одна отправляет записываемое слово, а вторая — ждёт прихода слова, после чего копирует его из буфера приёма в нужную ячейку памяти. Адрес, куда записать слово в памяти второго узла, указывается самим получателем.

В односторонних коммуникациях активное участие принимает лишь инициатор: при записи он отсылает записываемое слово, а то, достигнув по сети адресата, напрямую записывается в память второго узла (при этом процессорное время на ожидание и запись вторым узлом не тратится). Адрес, куда записать слово в памяти второго узла, указывается отправителем.

Программы на MPI используют двусторонние коммуникации Send/Recv; переход к односторонним коммуникациям Put/Get (поддерживаемым не только интерфейсом SHMEM, но и стандартом MPI-2) потенциально способен повысить продуктивность разработки и сопровождения параллельных программ и эффективность их выполнения.

Такому переходу мешает в основном тот факт, что большая часть имеющихся библиотек написаны на MPI, большая часть разработанных параллельных алгоритмов созданы для двухсторонней парадигмы Send/Recv, большая часть программистов и математиков, занимающиеся распараллеливанием научных задач, привыкли думать именно в парадигме Send/Recv и не хотят переходить на другие парадигмы, не видя существенных преимуществ.

Система программирования SHMEM (от shared memory — общая память) была разработана фирмой Cray более 15 лет назад, как интерфейс односторонних коммуникаций, способный стать эффективной альтернативой и дополнением к MPI и PVM. Интерфейс SHMEM поддерживается всеми MPP-системами фирмы Cray (Cray T3E, Cray XT5), Silicon Graphics (SGI Altix), интерконнектами Quadrics (QsNetIII). Также библиотека SHMEM (с некоторыми дополнениями) была реализована в системе МВС-Экспресс (под руководством А. О. Лациса).

По сути SHMEM реализует простейший вариант программирования в стиле PGAS (partitioned global address space). У каждого узла есть локальная память; каждому узлу также доступна удалённая память: узел может напрямую обращаться к локальной памяти любого узла системы. Поскольку обращения к удалённой памяти происходят через коммуникационную сеть, время их выполнения заметно больше, а темп — меньше, чем у обращений к локальной памяти. Ожидать выполнения каждой одиночной операции крайне дорого, поэтому требуется, чтобы программист явно выделял удалённые обращения.

В отличие от других PGAS-языков (например, UPC) SHMEM *заставляет* программиста явно выделять внешние обращения с помощью функций shmem_put, shmem_get, при этом дальнейшая группировка обращений и оптимизация выполняются аппаратно.

Большинство прикладных задач, использующих MPI, основано на простом коммуникационном шаблоне: preposted MPI_Recv + MPI_Send; однако такой шаблон гораздо естественнее записывается с использованием SHMEM, как асинхронная запись (shmem_put) с подтверждением прихода сообщений. Подобным образом авторами статьи были переписаны на SHMEM тесты NPB CG и NPB UA.

Основу SHMEM составляют две операции: shmem_put — запись в память удалённого узла и shmem_get — чтение из памяти удалённого узла. Синхронизация происходит с помощью встроенной функции shmem_barrier. Возможность напрямую обращаться к удалённой памяти даёт большинство преимуществ работы с «общей памятью», не накладывая никаких дополнительных ограничений на то, как память распределена физически.

Главная проблема односторонних операций — необходимость при отправке указывать адрес в памяти другого узла. Пересылать адреса между узлами — неэффективно, поэтому было предложено следующее решение: на всех узлах использовать симметричные массивы (симметричные указатели), имеющие одинаковые адреса на всех узлах. Если в программе объявлен симметричный массив, то у каждого процесса будут свои локальные данные, однако адреса элементов и размер массива у всех процессов будут одинаковые. Благодаря этому процессы могут обращаться к массивам других узлов, используя локально известные адреса. Симметричное выделение памяти происходит с помощью функции shmem_alloc.

Помимо операций записи (shmem_put) и чтения (shmem_get), реализованы атомарные операции сложения (shmem_add) и исключающее ИЛИ (shmem_xor).

В макетах M2/M3 реализован SHMEM с поддержкой эффективной передачи коротких сообщений, атомарных операций и быстрого барьера, поэтому большинство алгоритмов (CG, FFT, UA) на SHMEM не только проще записываются, но и эффективнее выполняются.

3.2. Порядок сообщений и синхронизация узлов

При односторонних коммуникациях особое значение приобретают функции синхронизации и порядок сообщений.

В макете M2 используется модель программирования, основанная на односторонних коммуникациях без подтверждений; все пакеты при этом передаются детерминировано.

Данная модель предполагает стиль программирования, в котором каждая итерация алгоритма делится на две фазы: подкачка и счёт. Например: асинхронная подкачка данных для (i+1)-ой итерации; счёт i-ой итерации; ожидание завершения подкачки для (i+1)-ой итерации; асинхронная подкачка данных для (i+2)-ой итерации; счёт (i+1)-ой итерации и т.д.

Обычно подкачка выполняется с помощью операций чтения, однако стандартным приёмом программирования в стиле PGAS является замена подкачки с помощью чтений на посылку узлом-владельцем элементов с помощью операций записи. На множестве реальных задач (CG, UA) этот приём даёт значительный выигрыш, так как снижает нагрузку на сеть, потому как операция чтения посылает по сети два пакета (запрос и ответ), а операция записи реализуется посылкой пакета только в одну сторону.

Ожидание завершения подкачки (с помощью операций shmem_put, shmem_get, shmem_add и shmem_xor) может осуществляться стандартным методом — с помощью вызова общего барьера (shmem_barrier). При этом также реализованы несколько специальных методов, функционально заменяющих барьер для контроля выполнения выданных асинхронных записей и чтений, позволяющих достичь намного большей производительности.

Контроль возврата выданных запросов на асинхронное чтение реализуется блокирующей функцией shmem_syncreads (используется аппаратно реализованный счётчик в сетевом интерфейсе: при отправке запроса на чтение он увеличивается на единицу, при получении ответа — уменьшается на единицу).

Контроль порядка выданных асинхронных записей реализуется функцией shmem_fence: гарантируется, что все отосланные до shmem_fence сообщения узлом A узлу B достигнут адресата строго до сообщений, отосланных после shmem_fence. Таким образом, порядок гарантируется только для пакетов, имеющих одинаковых отправителей и получателей. При детерминированной маршрутизации такой порядок соблюдается по определению и не требует какой-либо синхронизации узлов.

Контроль выполнения выданных асинхронных записей осуществляется с помощью блокирующей функцией shmem_quiet. Стандартный метод реализации — потребовать подтверждений для всех операций. Если пакеты были отосланы детерминировано, то достаточно запросить подтверждения только последней отправленной операции для каждого узла. Часто именно с помощью этой функции гарантируется глобальный порядок сообщений: ни одно сообщение, отосланное до shmem_quiet, не должно обогнать сообщение, отосланное после shmem_quiet.

Функция shmem_barrier гарантирует, что каждый процесс продолжит работу только после того, как все узлы дойдут до места вызова барьера в коде программы. Данная функция может быть реализована в две фазы: shmem_barrier_notify (начало барьера — неблокирующая) и shmem_barrier_wait (ожидание окончания барьера — блокирующая).

В большинстве реализаций SHMEM барьер выполняет важную упорядочивающую функцию: узлы выйдут из барьера только тогда, когда каждый узел получит все адресованные ему сообщения.

Однако данное условие является слишком строгим и во множестве (даже в большинстве) задач совсем не требуется. Вполне достаточно, чтобы узел выходил из барьера, как только он получит все отосланные ему до барьера сообщения. При таком условии каждый узел может продолжить счёт сразу, как только получит все адресованные ему данные, не дожидаясь остальных узлов (вплоть до следующего барьера). Такой «быстрый» барьер в среднем, как минимум, в два раза быстрее; именно он реализован в функции shmem_barrier библиотеки SHMEM

для макета M2/M3. Обычный «медленный» барьер реализуется выполнением двух «быстрых» барьеров подряд.

Помимо глобального барьера, в котором участвуют все процессы данной задачи, поддерживаются барьеры по заданной группе, где явно указывается (с помощью маски, шага) множество узлов, участвующих в барьере.

4. Результаты тестирования, производительность

4.1. Intel MPI Benchmarks

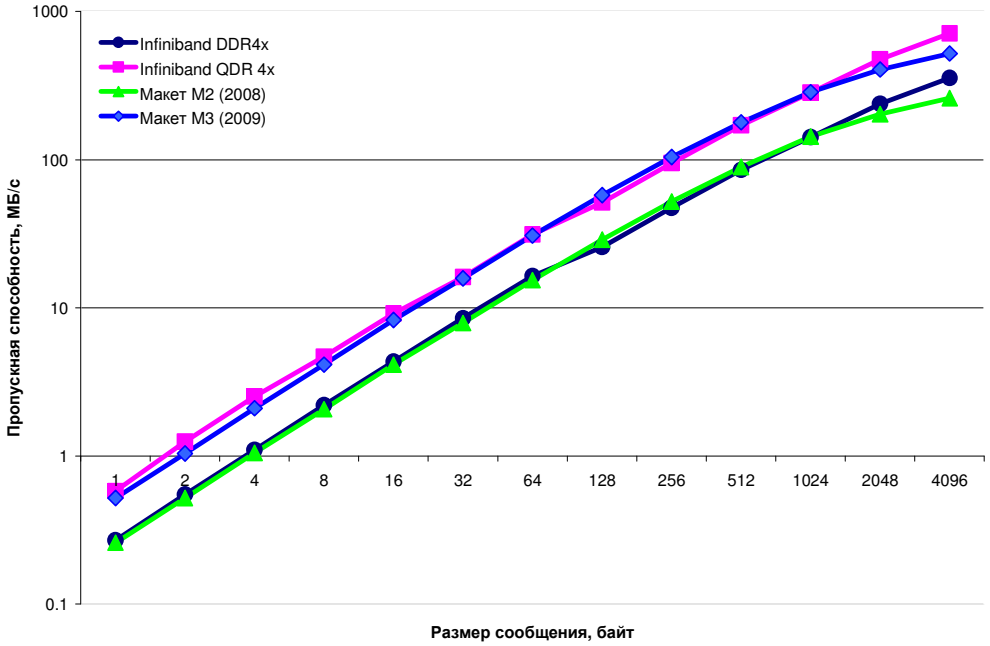


Рис. 2. Сравнение коммуникационной сети, разработанной НИЦЭВТ, с сетью Infiniband на тесте IMB PingPong.

Intel MPI Benchmarks (IMB) 3.2 — свободно распространяемый набор тестов, предназначенный для оценки эффективности выполнения операций MPI (с проверкой на корректность доставки сообщений) [7]. Ранее данный набор тестов был известен как Pallas MPI Benchmarks (PMB).

На макете M2 на тесте PingPong на сообщениях размером до 2 килобайт пропускная способность находится на уровне Infiniband DDR 4x. На больших сообщениях из-за более медленного линка M2 отстает. Макет M3 превосходит Infiniband QDR 4x на коротких сообщениях и несколько уступает на больших сообщениях (рис. 2).

На тесте Bcast коммуникационная задержка при сообщениях размером до 1 килобайт на макете M2 на 70% меньше, чем на Infiniband DDR 4x. На больших сообщениях задержка на M2 превосходит уровень Infiniband (рис. 3).

По характеристикам (на всех тестах из пакета IMB) на пакетах до 2КБ реализация MPI на M2 не отстает от реализации MPI на сети Infiniband DDR 4x. На больших пакетах производительность ниже по двум причинам: у M2 существенно меньшая пропускная способность сетевых линков, а также отсутствует поддержка передачи больших сообщений в режиме DMA.

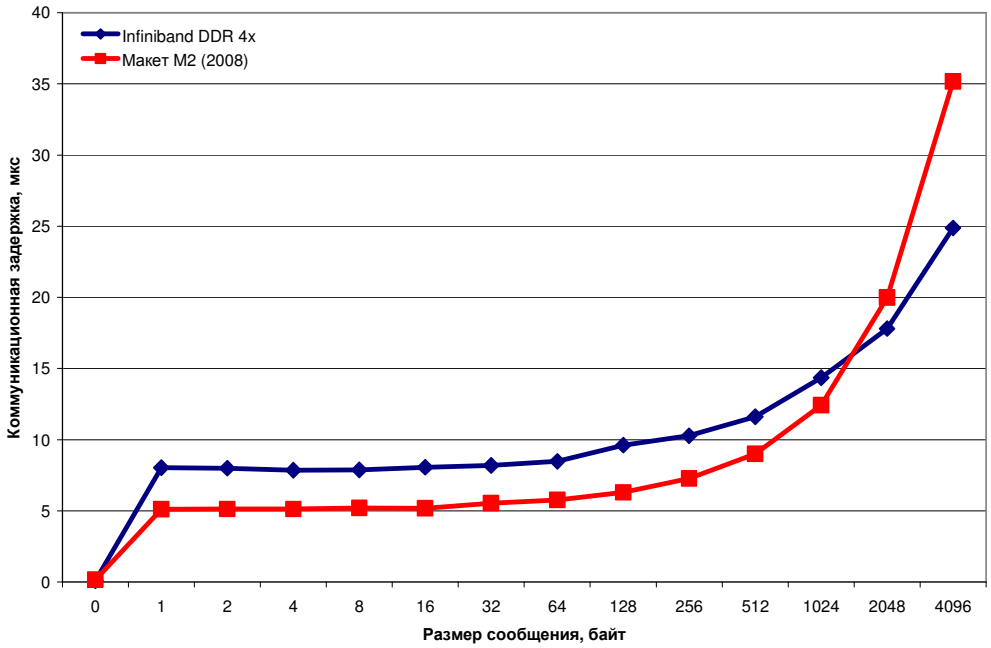


Рис. 3. Сравнение коммуникационной сети, разработанной НИЦЭВТ, с сетью Infiniband на тесте IMB Bcast.

4.2. HPCC RandomAccess

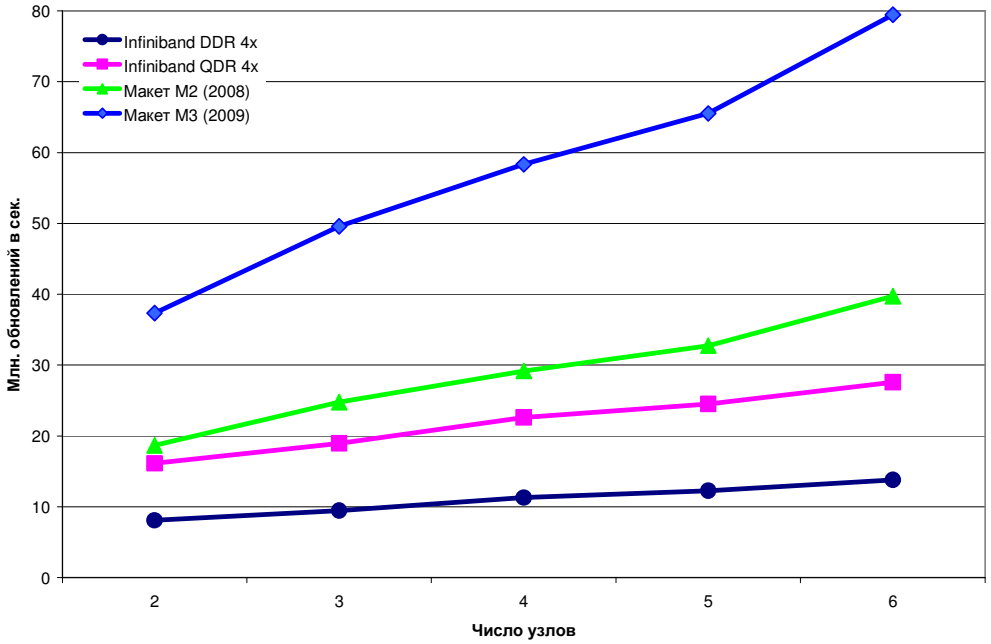


Рис. 4. Сравнение коммуникационной сети, разработанной НИЦЭВТ, с сетью Infiniband на тесте HPCC RandomAccess.

RandomAccess — один из семи тестов из набора HPC Challenge Benchmark [7]. Тест характеризует производительность вычислительной системы при наиболее неблагоприятном режиме доступа к памяти.

Тест имеет коммуникационный шаблон все-всем при минимальном размере пакета 8 байт; результатом теста является достигаемое число обновлений ячеек памяти по случайным адресам в секунду.

На макете M2 результаты более чем в 2 раза превышают результаты Infiniband QDR 4x в случае использования API нижнего уровня Infiniband Verbs, и более чем в 10 раз в случае использования MPI (рис. 4).

4.3. Барьерная синхронизация

Барьерная синхронизация — наиболее важная синхронизационная функция, используемая в MPI и SHMEM.

На макетах M2 и M3 до 4-х узлов барьерная синхронизация работает на уровне Infiniband QDR 4x, на большем числе узлов Infiniband QDR 4x значительно проигрывает — выполняется в несколько раз медленнее (рис. 5).

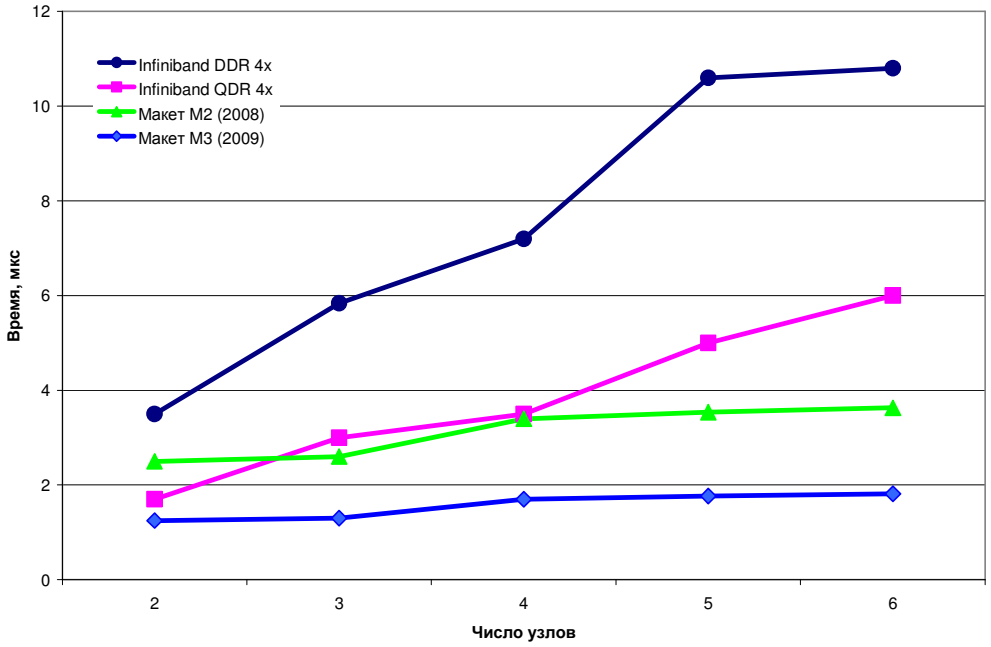


Рис. 5. Сравнение коммуникационной сети, разработанной НИЦЭВТ, с сетью Infiniband на тесте времени выполнения барьерной синхронизации.

4.4. NAS Parallel Benchmarks

NAS Parallel Benchmarks (NPB) — набор вычислительных задач, предназначенный для измерения производительности суперкомпьютеров в различных актуальных научных приложениях [8]. Набор включает тесты Conjugate Gradient (CG), Fast Fourier Transform (FT), MultiGrid (MG), Integer Sort (IS), Unstructured Adaptive (UA), Data Traffic (DT) и т. д.

Все тесты написаны на языке Fortran 77; существуют параллельные реализации NPB, использующие MPI, OpenMP, HPF, однако версий, использующих односторонние коммуникации в парадигме PGAS — нет.

Авторами статьи были реализованы с использованием SHMEM два теста — CG и UA. Ядром теста CG [8] является итеративное умножение разреженной матрицы на вектор. Поскольку реализованное в NPB блочно-циклическое разбиение матрицы специально нацелено на увеличение доли нерегулярных коммуникаций, на малом числе узлов производительность версии на SHMEM оказалась сравнима с версией на MPI. Оптимизированная версия, в которой используется блочное разбиения матрицы по строкам, позволяет более эффективно реализовать алгоритм на SHMEM, что даёт прирост в среднем на 36%.

В тесте UA [9] решается задача Дирихле уравнения теплопереноса в трехмерной кубической области на нерегулярной декартовой сетке. Источник тепла представляет собой шар, движущийся с постоянной скоростью. Для решения применяется нерегулярная сетка, причём каж-

дые несколько шагов происходит её адаптация: в областях с большим градиентом температуры сетка измельчается, с малым — укрупняется. Таким образом, тест нацелен на измерение производительности при нерегулярном динамически изменяемом шаблоне доступа к памяти.

Тест UA, появившийся в NPB 3.1, до этого распараллеливался лишь с использованием OpenMP, распараллеливание в модели MPI для данной задачи никем успешно не произведено до сих пор (т. е. невозможно на данный момент получить результаты об ускорении на системах с Infiniband), поэтому полученные результаты для SHMEM представляют особый интерес.

Ускорение на макете M2 для версии на SHMEM на 6-ти узлах (ядрах) составляет 5.4 раза (рис. 6); ускорение OpenMP-версии на других системах для 6-ти ядер не превосходит 2.8 раз; для 16-ти ядер — 4.4 раз. Ускорение измерялось для класса C относительно последовательного времени выполнения теста, которое для макета M2 составляло 1527 с, для MVS-Express — 1292 с, для SGI Altix 3700 — 1585 с (OpenMP версия).

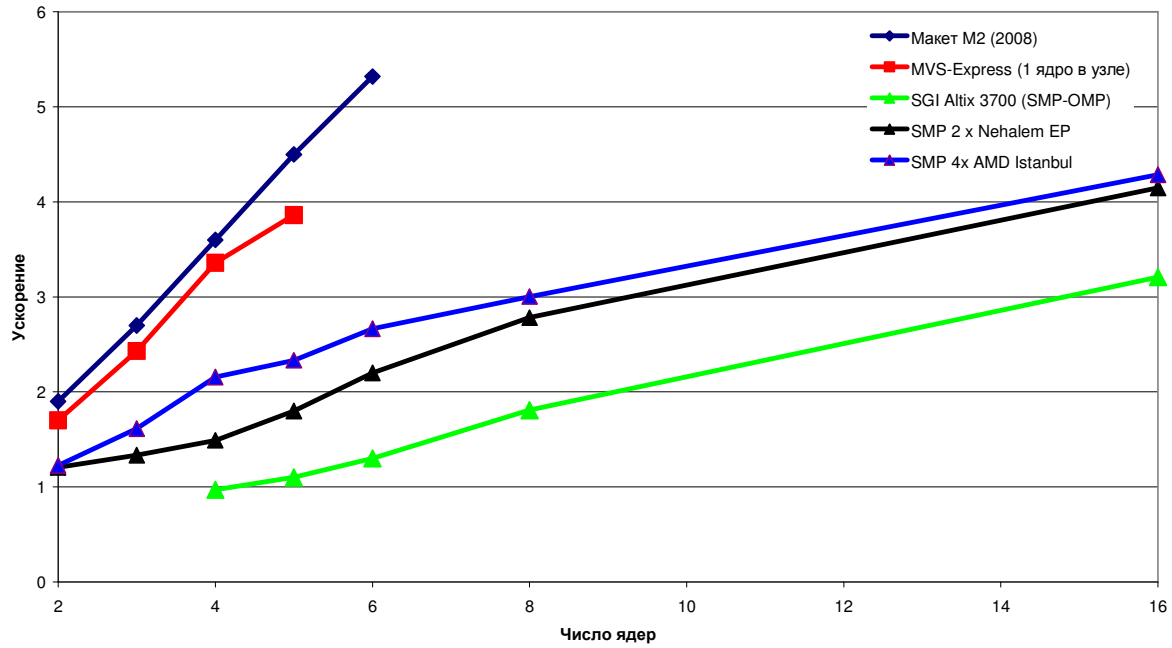


Рис. 6. Ускорение на тесте NPB UA от числа используемых ядер.

5. Заключение

Разработанная коммуникационная сеть 3D-тор с поддержкой глобально адресуемой памяти может быть использована как в кластерах, так и в суперкомпьютерах транспетафлопсного уровня производительности.

Коллективом программистов были адаптированы библиотеки MPI 1.1, SHMEM и MPI-2, а также тесты IMB, NetPIPE. Благодаря аппаратной поддержке односторонних коммуникаций, авторам удалось эффективно реализовать тесты NPB CG и UA, используя модель программирования SHMEM.

Модель программирования SHMEM, будучи простейшим представителем семейства языков в парадигме PGAS, является более продуктивной и эффективной, чем модель MPI, при этом она не содержит серьёзных недостатков более сложных языков PGAS, таких как UPC и Co-array Fortran. Данная тема будет подробно рассмотрена в следующей статье. Модель программирования SHMEM обладает достаточной гибкостью и функциональностью, чтобы реализовать все остальные языки PGAS.

В настоящее время успешно используется макетный образец сети второго поколения (M2), заканчивается тестирование макета третьего поколения (M3) и уже ведутся работы по созданию макета четвёртого поколения (M4).

Макет третьего поколения включает в себя эффективную поддержку коллективных операций broadcast и allreduce (по кольцам и измерениям тора), кольцевые буфера в памяти маршрутизатора, поддержку «активных сообщений», RPC и подтверждений.

Макет четвёртого поколения сможет адекватно конкурировать с последними разработками как отечественных фирм, так и зарубежных компаний (в том числе с Infiniband EDR и Cray Gemini). Он будет включать отдельные сети для глобальной синхронизации и коллективных операций (с древовидной топологией), также будет произведена замена интерфейса PCI Express на более производительный HyperTransport. Параллельно с этим ведутся работы по разработке блейдов под процессоры AMD Opteron, на которые будет интегрирован кристалл коммуникационной сети и мультитредовый ускоритель. Разрабатывается специализированная ОС суперкомпьютера стратегического назначения на базе ядра ОС Linux.

Коллектив разработчиков выражает признательность Л. К. Эйсымонту — за формирование первоначального внимания и интереса к тематике высокоскоростных коммуникационных сетей.

Литература

1. Keith D. Underwood , Michael J. Levenhagen , Ron Brightwell, Evaluating NIC hardware requirements to achieve high message rate PGAS support on multi-core processors, Proceedings of the 2007 ACM/IEEE conference on Supercomputing, November 10-16, 2007, Reno, Nevada.
2. Steven L. Scott, Synchronization and communication in the T3E multiprocessor, Proceedings of the seventh international conference on Architectural support for programming languages and operating systems, p.26-36, October 01-04, 1996, Cambridge, Massachusetts, United States.
3. J. Duato, S. Yalamanchili, L. Ni, Interconnection Networks, An Engineering Approach, IEEE Computer Society Press, 1997.
4. Корж А.А., Джосан О.В., Организация коммуникационной сети для транспетафлопсных суперкомпьютеров // журнал «Труды ИСА РАН: динамика неоднородных систем», том 32(3), 2008, сс. 267-274.
5. J. Kim, W. J. Dally, S. Scott, D. Abts, Technology-Driven, Highly-Scalable Dragonfly Topology, 35th International Symposium on Computer Architecture, 2008. 21-25 June 2008, pp. 77–88.
6. Aurora 8B/10B Protocol Specification SP002 (v2.1) June 24, 2009. http://www.xilinx.com/products/design_resources/conn_central/grouping/aurora.htm
7. S. Saini et al., Performance evaluation of supercomputers using HPCC and IMB Benchmarks. J. Comput. Syst. Sci. 74, 6 (Sep. 2008), 965-982.
8. H. Jin, M. Frumkin, J. Yan, The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance, NAS Technical Report NAS-99-011 October 1999.
9. M. Field, H. Feng, R.F. Van der Wijngaart, R. Biswas, Unstructured adaptive (UA) NAS Parallel Benchmark, NASA Ames Research Center, CA, 2004.

Суперкомпьютерные технологии России: объективные потребности и реальные возможности

С.М. Абрамов

В статье рассматривается природа суперкомпьютерных технологий, их роль в обеспечении конкурентоспособности различных отраслей экономики и страны в целом. Анализируются киберинфраструктуры развитых стран (США, объединенная Европа) — грид-системы национальных и региональных суперкомпьютерных центров. Дается оценка объективных потребностей России в суперЭВМ различного уровня производительности. Рассматривается имеющийся в России потенциал и научный задел для создания суперкомпьютерных технологий и оснащения России отечественными средствами высокопроизводительных вычислений в необходимом объеме. Анализируется возможность использования результатов суперкомпьютерной программы «СКИФ-ГРИД» для: эффективного достижения запланированных показателей (от транспетафлопсных систем до компактных суперЭВМ) сегодняшних российских суперкомпьютерных инициатив: обеспечения в перспективе выхода на экзафлопсную производительность (на рубеже 2018–2020 годов).

1. Суперкомпьютерные технологии

1.1. Роль и место суперкомпьютерных технологий

Сегодня критические (прорывные) технологии в государствах, строящих экономику, основанную на знаниях, исследуются и разрабатываются на базе широкого использования высокопроизводительных вычислений на суперЭВМ. И другого пути — нет. Без серьезной суперкомпьютерной инфраструктуры:

- невозможно создать современные изделия высокой (аэрокосмическая техника, суда, энергетические блоки электростанций различных типов) и даже средней сложности (автомобили, конкурентоспособная бытовая техника и т.п.);
- невозможно быстрее конкурентов разрабатывать новые лекарства и материалы с заданными свойствами;
- невозможно развивать перспективные технологии (биотехнологии, нанотехнологии, решения для энергетики будущего и т.п.).

Сегодня суперкомпьютерные технологии (СКТ) по праву считаются важнейшим фактором обеспечения конкурентоспособности экономики страны, а *единственным* способом победить конкурентов объявляют возможность обогнать их в расчетах. Здесь характерны слова Президента Совета по конкурентоспособности США*: «Технологии, таланты и деньги доступны многим странам. Поэтому США стоит перед лицом непредсказуемых зарубежных экономических конкурентов.* ***Страна, желающая победить в конкуренции, должна победить в вычислениях»***[1].

Отметим два обстоятельства в данном высказывании: (1) речь идет об экономике в целом, обо всех секторах экономики — сказанное верно для добывающих и перерабатывающих секторов экономики, и особенно это верно при разработке новых технологий; (2) для победы в конкуренции требуется ***победа*** в вычислениях — мало быть способным проводить вычисления, надо иметь ***самые мощные*** суперЭВМ, ***самые мощные*** прикладные пакеты и ***уметь использовать*** эти ресурсы в интересах экономики.

Тем самым, краткое определение сегодняшней роли суперкомпьютерных технологий может быть таким: это ключевая критическая технология, ***единственный инструмент***, дающий возможность победить в конкурентной борьбе.

[1] *"With technology, talent and capital now available globally, the U.S is facing unprecedented economic competition from abroad. The country that wants to out compete must out-compute"* — Deborah Wince-Smith, President of the Council on Competitiveness.

1.2. Киберинфраструктура страны — забота государства

Каждая эпоха развития экономики требовала создания соответствующей инфраструктуры страны. В разные периоды это были национальные сети железных дорог или автомагистралей, национальные энергетические системы, системы газо- и нефтепроводов и т.п. Чаще всего инфраструктура страны (как «общественное благо», необходимое всем отраслям экономики, всем слоям населения) создается либо исключительно государством и только за счет бюджета страны, либо при значительной доле участия государства.

Сегодня, исходя из роли суперкомпьютерных технологий, в развитых странах мира для перехода к экономике знаний создается новая инфраструктура государства — государственная система из мощных национальных суперкомпьютерных центров (СКЦ), объединенных сверхбыстрыми каналами связи в грид-систему. Для такой системы часто используют термин *киберинфраструктура* [1]. В этих странах на создание национальной киберинфраструктуры выделяются большие финансы из государственных бюджетов: в 2005–2008 гг. США тратили на эти цели от 2 до 4 млрд. долларов в год.

1.3. Природа экономической эффективности киберинфраструктуры

В развитых странах[1] государство вкладывает бюджетные деньги в создание национальных суперкомпьютерных центров, в их объединение в грид-систему, в ее содержание: оплату электроэнергии, ремонтов, зарплаты персонала, поддержки сервисов. Созданные и поддержанные за бюджетные средства ресурсы киберинфраструктуры предоставляются не только научным и образовательным организациям, но и коммерческим компаниям, например таким как: IBM, General Electric, Pratt & Whitney. Причем, ресурсы предоставляются бесплатно, но на конкурсной основе: чья идея, требующая суперкомпьютерных расчетов, сулит больший эффект. Получив по конкурсу необходимые ресурсы, компании выполняют свои НИОКР, результаты которых являются интеллектуальной собственностью данных компаний. В рамках таких НИОКР компании разрабатывают принципиально новые изделия (материалы, технологии и т.п.), обладающие подавляющими конкурентными преимуществами. Это позволяет потеснить конкурентов, расширить свое присутствие на рынке, свои объемы продаж. Как следствие, такие компании платят больше налогов. И только в этот момент, через уплаченные в большем объеме налоги, в бюджет возвращаются средства, ранее вложенные в киберинфраструктуру.

Подчеркнем, в большинстве национальных суперкомпьютерных центров в мире никто не пытается продавать машинное время, оказывать платные вычислительные услуги. Для них применяется механизм бюджетного финансирования и извлечения экономического эффекта за счет бюджетной эффективности (за счет увеличения налоговых поступлений), а не за счет прямой коммерческой эффективности (прямая продажа услуг или ресурсов СКЦ). Более того, постепенно такой подход к делу реализуется и на региональном уровне: появились первые региональные[2] суперкомпьютерные центры, созданные и содержащиеся за счет местных бюджетов, работающие на принципе бюджетной эффективности.

Нам в России, привыкшим за последние годы преувеличивать роль рыночных механизмов, предстоит перенимать опыт ведущих стран:

- с одной стороны, рассчитывать на получение отдачи от национальной киберинфраструктуры не за счет прямых продаж, а за счет ее бюджетной эффективности;
- с другой стороны, создавать условия для успешной работы механизма экономической эффективности. Что, среди прочего, включает усилия по подготовке и переподготовке кадров, повышению мотивации предприятий в области инноваций и, как следствие, повышение их потребности в использовании суперкомпьютерных технологий и их готовности к этому.

[1] В первую очередь речь пойдет о США и объединенной Европе.

[2] Суперкомпьютерные центры штатов Техас и Нью-Мексико, см. места 8 и 17 в рейтинге Топ500 за июнь 2009 года. СуперЭВМ в каждом из этих региональных суперкомпьютерных центров мощнее, чем самая мощная суперЭВМ в России.

Иначе бюджетные средства, вложенные в киберинфраструктуру России, окажутся «закопанными в землю».

1.4. Многогранные суперкомпьютерные технологии

Развитие суперкомпьютерных технологий в России включает целый комплекс задач, охватывающий следующие сферы:

- **разработка, реализация и производство аппаратных средств суперЭВМ**, что включает элементную базу, печатные платы и конструктивы, различные модули суперЭВМ и вычислительные системы целиком;
- **разработка, реализация и производство базового системного программного обеспечения (ПО) суперЭВМ,** что включает операционные системы, параллельные файловые системы, базовые библиотеки поддержки параллельных вычислений, обработчики очередей и планировщики заданий, системы мониторинга и управления суперЭВМ и т.п.;
- **разработка, реализация и производство программного обеспечения поддержки разработки параллельных приложений,** что включает различные языки и системы параллельного программирования, другие средства параллельного программирования (библиотеки шаблонов, параллельные реализации библиотек подпрограмм, высокоуровневые библиотеки поддержки параллельного программирования и т.п.), инструментальные системы поддержки создания параллельных приложений, вспомогательные средства (отладчики, трассировщики и визуализаторы трасс, профилировщики, средства оптимизации и т.п.);
- **разработка, реализация и производство прикладного программного обеспечения,** что включает различные пакеты для параллельных вычислений, визуализации результатов вычислений, поддержки ввода-вывода и хранения данных и т.п.,— для различных прикладных областей;
- **создание и эксплуатация СКЦ, объединение их в грид-систему, формирование служб и сервисов на их основе:** предоставление вычислительной мощности, поддержка пользователей готовых прикладных пакетов, консультации, разработка заказного ПО под новые задачи, проведение расчетов «под ключ» — от интуитивной постановки задачи, через выработку математической модели, выбора расчетной схемы, ее программной параллельной реализации, отладки, выполнения расчета и передачи результатов заказчику в пригодной для него форме;
- **подготовка и переподготовка кадров для суперкомпьютерной отрасли.** Речь идет о всех сторонах поддержки образовательного процесса: разработка учебно-методических материалов, учебной базы, организация учебного процесса и т.п. Необходима подготовка специалистов для решения всех задач, перечисленных выше: разработчики (аппаратных средств и различных классов ПО), специалисты по производству, монтажу и запуску суперЭВМ, персонал поддержки эксплуатации — операторы, инженеры по эксплуатации, системные администраторы,— сотрудники для поддержки сервисов СКЦ — от консультантов до специалистов, способных по начальной постановке задачи построить математическую модель, ее программную параллельную реализацию и выполнить расчеты.

Практически каждая упомянутая выше разработка требует проведения фундаментальных поисковых исследований. И только затем возможен выход на уровень НИР, НИОКР и ОКР.

1.5. Уровни суперкомпьютерных технологий

В каждый момент времени, если посмотреть уровень развития суперкомпьютерной отрасли, то можно выделить два слоя:

- **Технологии уровня «N».** Это суперкомпьютерные технологии будущего, которые еще не вполне освоены, а только-только разрабатываются. Инновационные, совершенно новые технические решения, недоступные на рынке. На их базе создают суперкомпьютеры, которые сильно вырываются вперед. Как правило, это машины, соответствующие первым 5–10 местам списка Top500 [2]. Эти суперЭВМ обладают мощностью, которая радикально отличает их от всех других машин. И на платформе таких суперЭВМ можно выполнить расчеты, которые невозможно повторить (ни за какое разумное время) на суперЭВМ более низ-

кого класса. На базе таких расчетов можно создать в разных отраслях принципиально новые материалы, новые технологические решения, новые изделия, которые позволят обладающей ими стороне быть вне конкуренции и существенно оторваться от других игроков в соответствующей отрасли.

– **Технологии уровня «N-1».** Технологии более низкого уровня, отработанные решения, широкодоступные на рынке. СуперЭВМ на их базе доступны (и даже могут быть воспроизведены) во многих странах. Соответственно, расчеты, выполняемые на таких машинах, могут быть воспроизведены многими. На базе таких расчетов можно создать в разных отраслях конкурентоспособные материалы, технологические решения, изделия,— достичь нормального качества, заурядной конкурентоспособности. С такими изделиями можно выходить на мировой рынок, но на нем придется вести изнурительную конкурентную борьбу с десятком подобных товаров, созданных на базе подобных расчетов.

Надо отметить, что все разработанные в предыдущие годы отечественные суперЭВМ и суперкомпьютерные технологии относились к технологическому уровню N-1. И России, для того, чтобы победить в вычислениях, предстоит создать свои собственные технологии уровня N. График отставания России от технологического уровня N приведен на рисунке 1 (построен на основе анализа данных мирового рейтинга Top500).

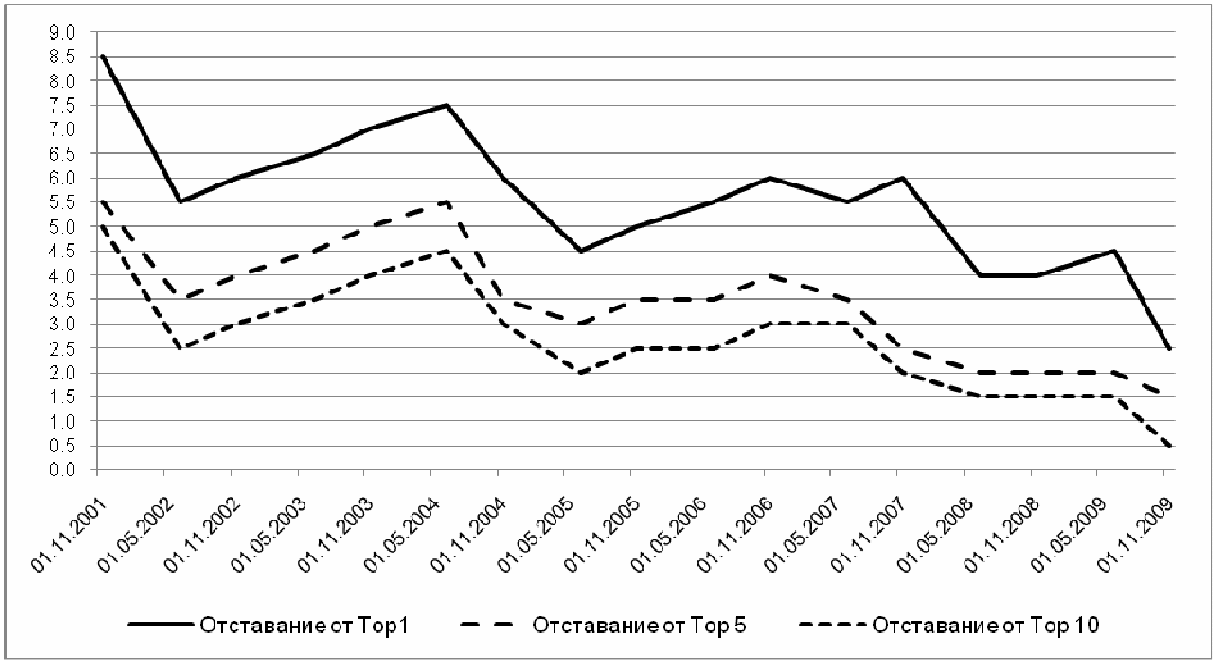


Рис 1. Отставание (в годах) уровня самых мощных суперЭВМ в России от передового мирового технологического уровня

1.6. Иерархия киберинфраструктуры

В каждый момент времени в развитых странах (в первую очередь — в США и объединенной Европе) в части их оснащения суперЭВМ прослеживается наличие «пирамиды» — для обеспечения сбалансированной киберинфраструктуры страны выпускаются суперЭВМ различных уровней производительности:

– **суперЭВМ в крупнейших национальных центрах** — единичные установки в стране, соответствующие местам 1–20 в мировом рейтинге Top500;
– **суперЭВМ в крупнейших региональных и отраслевых центрах** — два–четыре десятка установок в стране, соответствующих местам 21–100 в мировом рейтинге Top500;
– **суперЭВМ в крупных региональных и корпоративных центрах** — от четырех десятков до сотни установок в стране, соответствующих местам 101–250 в мировом рейтинге Top500;

– **суперЭВМ предприятий и научных учреждений** — одна–три сотни установок в стране, соответствующих местам 251–500 в мировом рейтинге Top500;
– **суперЭВМ небольших исследовательских компаний, отдельных лабораторий и научных подразделений** — сотни установок в стране, составляющих самый нижний уровень пирамиды киберинфраструктуры и не входящих в мировой рейтинг Top500 или вышедших из него в силу быстрого развития отрасли.

В таблице 1 приведены подготовленные на основе анализа текущего (ноябрь 2009 г.) мирового рейтинга Top500 сведения о количестве таких суперЭВМ и об их суммарной Linpack-производительности (сумма считалась отдельно по каждому классу суперЭВМ) — колонки с (1) по (4) для США, Единой Европы, Китая и России. В таблице 2 приведен уровень сегодняшнего отставания России в оснащенности суперЭВМ от США, Единой Европы и Китая.

Таблица 1. Состояние (на июнь 2009 года) оснащенности суперЭВМ различных стран и оценка потребностей России на ближайший период (до 2012 года)

	(1) США	**(2) Единая Европа**	**(3) Китай**	**(4) Россия**	**(5) Потребности России**
Top1–20: СуперЭВМ в крупнейших национальных центрах	12 шт. 7 062 Tflops[1]	3 шт. 1 274 Tflops	2 шт. 743 Tflops	1 шт. 350 Tflops	2 шт.
Top21–100: СуперЭВМ в крупнейших региональных и отраслевых центрах	34 шт. 2 911 Tflops	29 шт. 2 458 Tflops	1 шт. 102 Tflops	1 шт. 107 Tflops	15 шт.
Top101–250: СуперЭВМ в крупных региональных и корпоративных центрах	83 шт. 2 856 Tflops	37 шт. 1 313 Tflops	11 шт. 366 Tflops	3 шт. 117 Tflops	30 шт.
Top251–500: СуперЭВМ предприятий и научных учреждений	148 шт. 3 586 Tflops	75 шт. 1 777 Tflops	7 шт. 167 Tflops	3 шт. 71 Tflops	60 шт.
Top1–500: ВСЕГО	277 шт. 16 416 Tflops	144 шт. 6 822 Tflops	21 шт. 1 379 Tflops	8 шт. 646 Tflops	107 шт.

Таблица 2. Уровень отставания России в оснащенности суперЭВМ (на июнь 2009 года, разы)

	От США	**От Единой Европы**	**От Китая**
По числу СуперЭВМ Top1–500	34,6	18,0	2,6
По суммарной производительности СуперЭВМ Top1–500	25,4	10,6	2,1

2. Оценка объема объективных потребностей России в суперкомпьютерных технологиях

Конечно, оценка потребностей России в суперкомпьютерных технологиях должна выполняться с учетом всей многогранности (раздел 1.4) данного понятия. Однако это невозможно сделать в рамках короткой статьи. Ограничимся оценкой объективной потребности в парке суперЭВМ, рассчитывая на то, что из этого может быть оценен хотя бы порядок необходимого

[1] 1 Gflops — миллиард (10^9) операций с плавающей точкой в секунду. 1 Tflops — триллион (10^{12}) операций с плавающей точкой в секунду; 1 Pflops — квинталион (10^{15}) операций с плавающей точкой в секунду.

уровня развития всех остальных аспектов (программное обеспечение, сервисы, приложения, кадры и т.п.).

Полагая, что для России необходимо обеспечение паритета в оснащенности разными классами суперЭВМ — по крайней мере, на уровне, сравнимом с Единой Европой (или отстающем не более 2–3 раз), получаем оценку объективной потребности России, указанную в колонке (5) таблицы 1.

Следует учитывать, что производительность суперЭВМ, соответствующая тому или иному месту в мировом рейтинге Top500, очень быстро меняется. Поэтому в оценке потребности России на период 2010–2012 годов должны быть учтены тенденции и прогноз развития мирового состояния отрасли: какие (по производительности) к тому времени суперЭВМ будут в странах-конкурентах в национальных центрах (места 1–20 в мировом рейтинге); какие — в крупнейших региональных и отраслевых центрах (места 21–100 в мировом рейтинге) и т.д. Данные сведения приведены в таблице 3.

Тем самым, объективно необходимый уровень обеспеченности России суперЭВМ, оцененный из соображений паритета или хотя бы сравнимости с Единой Европой, на период до 2012 года диктуют следующие потребности:

- создание двух или более суперЭВМ для крупнейших федеральных или региональных центров с производительностью 1 Pflops в 2010 году, с их расширением до 5–10 Pflops в 2012 году — **за счет федерального бюджета;**
- создание 15 и более суперЭВМ для крупных региональных и отраслевых центров с производительностью 200–500 Tflops в 2010 году, с их расширением до 500–1600 Tflops в 2012 году — **за счет федеральных, региональных и отраслевых средств;**
- создание 30 и более суперЭВМ для региональных и корпоративных центров с производительностью 100 Tflops в 2010 году, с их расширением до 450–500 Tflops в 2012 году — **за счет региональных, отраслевых и корпоративных средств;**
- создание 60 и более суперЭВМ для ведущих высокотехнологичных предприятий, научных учреждений и инновационных университетов с производительностью до 100 Tflops в 2010 году, с их расширением до 350–450 Tflops в 2012 году — **за счет отраслевых и внебюджетных средств;**
- создание в 2010–2012 годах «фундамента отечественной пирамиды» суперЭВМ за счет производства сотен компактных суперкомпьютеров для оснащения высокопроизводительными вычислительными ресурсами отдельных научных подразделений высокотехнологичных предприятий, исследовательских лабораторий и подразделений инновационной направленности, с обеспечением следующего уровня производительности компактных суперЭВМ: 2010 год — 2–4 Tflops, 2011 год — 6–12 Tflops, 2012 год — 12–25 Tflops.

Таблица 3. Оценка потребностей России в оснащенности суперЭВМ на ближайший период (до 2012 года) с учетом изменения уровня производительности суперЭВМ в различных частях рейтинга Top500

	Количество	**2010**	**2011**	**2012**
Top1–20: СуперЭВМ в крупнейших национальных центрах	2 шт.	до 4 Pflops	до 7 Pflops	до 15 Pflops
Top21–100: СуперЭВМ в крупнейших региональных и отраслевых центрах	15 шт.	до 430 Tflops	до 850 Tflops	до 1,600 Tflops
Top101–250: СуперЭВМ в крупных и региональных и корпоративных центрах	30 шт.	до 130 Tflops	до 250 Tflops	до 490 Tflops
Top251–500: СуперЭВМ предприятий и научных учреждений	60 шт.	90–120 Tflops	180–240 Tflops	350–470 Tflops

Любые другие сценарии развития отечественной отрасли суперЭВМ неизбежно приведут к накоплению степени отставания в технологическом и экономическом развитии России от стран — мировых лидеров.

3. Возможности России в развитии СКТ

В России еще в советский период и последующие годы участниками отечественной отрасли высокопроизводительных систем были такие организации как: Институт точной механики и вычислительной техники имени С.А. Лебедева АН СССР (ИТМиВТ, ведущие разработчики академики В.С. Бурцев, В.А. Мельников), головной центр разработок программы «ЕС ЭВМ» — Научно-исследовательский центр электронной вычислительной техники (сейчас — ОАО «НИЦЭВТ»), ФГУП НИИ «Квант» (академик В.К. Левин), Всероссийский НИИ экспериментальной физики — Институт теоретической и математической физики (ВНИИЭФ-ИТМФ, г. Саров), Научно-исследовательский институт многопроцессорных вычислительных систем (НИИ МВС, академик А.В. Каляев), Межведомственный суперкомпьютерный центр РАН (МСЦ РАН, академик Г.И. Савин), головной исполнитель от России суперкомпьютерных программ «СКИФ» и «СКИФ-ГРИД» — Институт программных систем РАН (ИПС имени А.К. Айламазяна РАН, чл.-корр. С.М. Абрамов), Научно-исследовательский институт системных исследований РАН (НИИСИ РАН, академик В.Б. Бетелин), группа разработчиков систем на базе архитектуры микропроцессора серии «Эльбрус» (сейчас — ЗАО «Московский центр спарк технологий», «МЦСТ») и многие другие организации. За счет выполнения специальных и оборонных заказов, эти и многие другие организации сохранили высокий уровень научно-технической экспертизы, кадровый потенциал.

В настоящее время основные отечественные разработчики суперкомпьютерных технологий и систем остались прежними, однако время, политические и экономические преобразования наложили существенный отпечаток на состояние и научный потенциал этого суперкомпьютерного сообщества.

В последние 6–8 лет потребности рынка серверов и высокопроизводительных систем выдвинули в этот сектор информационных технологий ряд новых игроков — системных интеграторов, специализирующихся на поставках в Россию готовых решений крупных зарубежных компаний, таких как IBM, HP, Sun и других, и организацию сборки кластерных систем из доступных на зарубежном рынке компонентов и сетевых плат с помощью «отверточной технологии». Некоторые компании из этой группы, накопив достаточный опыт, перешли к собственным разработкам отдельных узлов высокопроизводительных систем, однако называть эти устройства законченными отечественными разработками несколько преждевременно, так как значительная их часть производится за рубежом (Китай, Тайвань, Гон-Конг), а решения заимствованы или на их репликацию получена лицензия, что само по себе уже определяет отставание от лучших зарубежных образцов. К таким компаниям в первую очередь относятся «Открытые технологии», «Т-платформы», «Эр-Стайл», «Арбайт», «Крафтвэй», «Крок» и другие.

Следует отметить, что все представленные на российском рынке отечественные суперкомпьютеры, кроме специализированных, созданы с использованием элементной базы зарубежных производителей, поскольку сегодняшние отечественные микропроцессорные решения значительно уступают по своим параметрам решениям ведущих мировых производителей (Intel, AMD, IBM). С учетом этого разумно сочетать:

– развитие отечественных суперкомпьютерных технологий (аппаратных средств и программного обеспечения) на базе зарубежных микросхем;
– создание отечественной элементной базы для современных суперЭВМ.

Это позволит избежать периода ожидания достижения российской микропроцессорной техникой нужного уровня развития и быть всегда в готовности для немедленного использования отечественных микропроцессоров по мере их появления.

В последние годы государством проводился ряд мер, направленных на ускоренное развитие в области высокопроизводительных систем. Сюда стоит отнести научно-технические программы «СКИФ» (2000–2004) [3] и «СКИФ-ГРИД» (2007–2010), государственный заказчик-координатор от России — Роснаука, головной исполнитель от России — ИПС имени А.К. Айламазяна РАН. В следующем разделе рассмотрены более подробно результаты данных программ и возможности их использования в сегодняшних планах развития суперкомпьютерных технологий в России.

4. Суперкомпьютеры семейства «СКИФ»: реальный ответ на объективные потребности

4.1. Ранее полученные результаты по программам «СКИФ» и «СКИФ-ГРИД»

К настоящему времени в рамках программам «СКИФ» и «СКИФ-ГРИД» создано три поколения (Ряды 1, 2 и 3) семейства отечественных суперкомпьютеров «СКИФ» [4]. Подготовлена конструкторская и программная документация с литерой O_1. Выпущено 18 опытных образцов. Как результат программ «СКИФ» и «СКИФ-ГРИД»:

Суперкомпьютерные программы «СКИФ» и «СКИФ-ГРИД» внесли серьезный вклад в развитие суперкомпьютерной отрасли и суперкомпьютерного рынка России. По данным национального рейтинга 50 самых мощных суперкомпьютеров в СНГ [5], в последние годы 75–80% суперкомпьютеров отечественной разработки обеспечиваются суперЭВМ семейства СКИФ и установками с использованием технологических решений семейства СКИФ. За время выполнения программ «СКИФ» и «СКИФ-ГРИД» шесть систем семейства «СКИФ» 14 раз вошли в престижный всемирный рейтинг суперЭВМ Top500 (с максимально высокой позицией № 36 в июне 2008 г.):

- **СКИФ-Аврора ЮУрГУ**, 21.8/24[1] Tflops — 11'2009 № 450;
- **СКИФ МГУ «Чебышёв»**, 47.17/60 Tflops — 06’2008 № 36, 11’2008 № 54, 06'2009 № 82, 11'2009 № 103;
- **СКИФ Урал**, 12.2/15.94 Tflops — 06’2008 № 283;
- **СКИФ Cyberia**, 9.01/12 Tflops — 06’2007 № 105, 11’2007 № 200, 06’2008 № 485;
- **СКИФ К-1000**, 2.032/2.534 Tflops — 11’2003 № 98, 06’2005 № 182, 11’2005 № 331, 06’2006 № 489;
- **СКИФ К-500**, 0.424/0.717 Tflops — 11’2003 № 406.

Отметим: за всю историю только восемь машин, разработанных в России, входили в мировой рейтинг Top500, шесть из них (75%) — суперкомпьютеры семейства «СКИФ».

Создано базовое, системное, инструментальное и прикладное программное обеспечение (ПО) в самых разных областях применения суперЭВМ «СКИФ». Разработанные технологии используются в науке, образовании и реальных отраслях экономики России. Результаты программ «СКИФ» и «СКИФ-ГРИД» неоднократно докладывались и демонстрировались на ведущих суперкомпьютерных конференциях и выставках (трижды на самой известной международной суперкомпьютерной конференции и выставке ISC — ноябрь 2006 г. в г. Тампа, США, июнь 2009 г. в г. Гамбург, Германия, ноябрь 2009 в г. Портленд, США).

В рамках программы «СКИФ-ГРИД» традиционные направления разработки («суперЭВМ семейства СКИФ и ПО для них», «прикладные системы», «информационная безопасность») дополнены разработками GRID-технологий — отечественное ПО промежуточного уровня для территориально-распределенных грид-систем, с поддержкой интеграции вычислительных ресурсов (вычислительный грид), метакомпьютинга, распределенного хранения данных и их обработки, управления пользователями и ресурсами грид-сети, и других грид-технологий.

4.2. СуперЭВМ ряда 4 семейства «СКИФ»: отечественные решения от компактных суперЭВМ до транстпетафлопсных

На сегодняшний день важнейшим проектом является разработка и выпуск первых моделей суперкомпьютеров ряда 4 семейства «СКИФ» [6] — для них так же используется название «СКИФ-4» и СКИФ-Аврора.

Состояние разработки позволило 23–25 июня 2009 года на международной суперкомпьютерной конференции ISC’09 в Гамбурге представить работоспособные модули суперкомпьютера СКИФ-Аврора, рисунок 2. Демонстрировались разобранный вычислительный узел и работающее полушасси с водяным охлаждением, на котором считались реальные задачи (например, расчет прогноза погоды), функционировала подсистема мониторинга и управления установкой.

[1] Указана производительность на тесте Linpack и, через дробь, пиковая производительность.

Экспозиция получила высокую оценку от ведущих специалистов из России, среди которых были академик Г.И. Савин, Б.М. Шабанов (МСЦ РАН), чл.-корр. РАН В.В. Воеводин, А.В. Тихонравов (НИВЦ МГУ), Г.С. Елизаров (НИИ «Квант») и специалисты других стран. На полушасси СКИФ-Авроры был получен сертификат совместимости *Intel Cluster Ready*, что свидетельствует о гарантированной работоспособности на данной платформе большинства прикладных пакетов программ.

Рис 2. Представление суперЭВМ СКИФ 4/Н на конференции ISC'09, Гамбург, 23–25 июня 2009. Слева направо: общий вид экспозиции; работающее полушасси с водяным охлаждением; расчет прогноза погоды и панель подсистемы мониторинга СКИФ-Авроры.

Полушасси — основные модули СКИФ-Авроры с производительностью 1,5 Tflops,— могут устанавливаться в монтажный шкаф — по 8 штук с двух сторон, рисунок 3. Таким образом, в монтажном шкафу может быть собран суперкомпьютер с производительностью 24 Tflops. Сегодня в мире это высшая производительность на один шкаф.

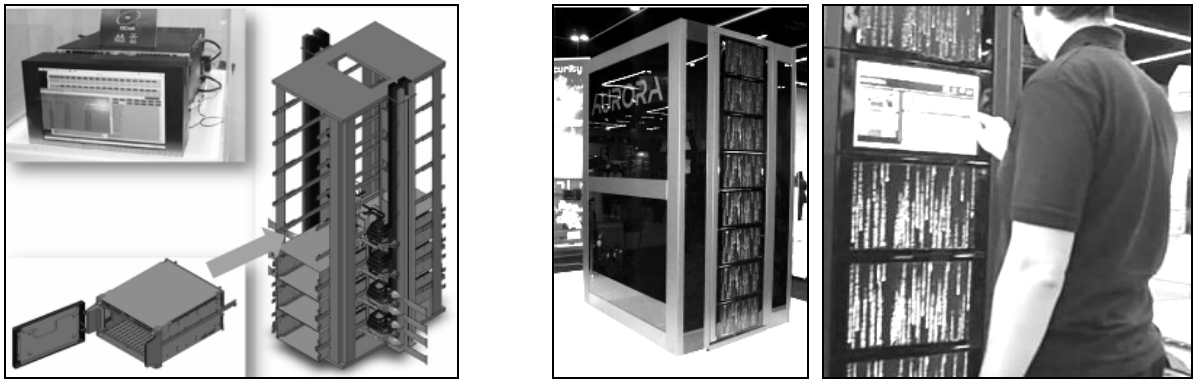


Рис 3. Слева схема размещения 16 полушасси в монтажном шкафу. Справа демонстрация установка СКИФ-Аврора на конференции SC'09 (Портленд, США) в конфигурации одного шкафа.

В ноябре на конференции SC'09 (Портленд, США) демонстрировалась установка СКИФ-Аврора в конфигурации одного шкафа, рисунок 3; 17 ноября 2009 года СКИФ-Аврора была включена в рейтинг 500 самых мощных машин мира — 450 место с показателями производительности 21.8 Tflops на тесте Linpack, 24 Tflops пиковой производительности, КПД = 21.8/24 = 90.8%. Посетители отмечали высокое эстетическое качество исполнения установки и ее уникальную эргономику:

– во-первых, шкаф охлаждается водой, не содержит подвижных частей, является абсолютно бесшумным;
– во-вторых, каждое полушасси закрыто как крышкой сенсорным жидкокристаллическим экраном, который является панелью управления суперкомпьютером при помощи прикосновения пальцев.

Построение крупной системы СКИФ-Аврора необходимой производительности обеспечивается расположением шкафов вдоль непрерывной линии и связью шкафов между собой отечественной системной сетью с топологией 3D-тор. Данная сеть обладает повышенной масштабируемостью и обеспечивает построение систем рекордной производительности. В 2009 году можно было таким образом построить вычислитель в 500 Tflops, который бы занимал объем всего 21 монтажного шкафа, не содержал подвижных частей, был бы надежным и беззвучным.

Подчеркнем, что водяное охлаждение позволяет экономить место и деньги на внутрирядные кондиционеры и на организацию «горячих коридоров».

Суперкомпьютеры СКИФ-Аврора по нескольким параметрам обладают преимуществом перед всеми известными на сегодня разработками (по многим из них речь идет о существенном превышении от мирового уровня: не на несколько процентов, а в 1,5–2 раза):

– в 1,5 раза лучше эффективность использования системой электроэнергии;
– в 2 раза плотнее упаковка вычислительной мощности;
– в 1,5 раза выше пропускная способность системной сети (российская разработка);
– повышенная эффективность реализации массовых операций в системной сети;
– повышенная надежность суперкомпьютера: нет подвижных частей, N+1 резервирование, тройное резервирование в системе управления и мониторинга (российская разработка);
– улучшенная система электропитания;
– улучшенные эргономические и эстетические показатели: сенсорный мультиэкран для управления, вся система — беззвучная.

За счет российской реализации по четырем технологиям удается преодолеть ограничения экспортного контроля (поправка Джексона–Вейнике) — мы сами разработали технологические решения уровня N — то, что запрещено ввозить в Россию, да и не только в Россию:

– тесносвязанный гибридный вычислительный узел: высокая совместимость с существующим программным обеспечением в комбинации с возможностью использования FPGA-ускорителей (как это было сделано в линии Cray XD-x — недоступны в России);
– улучшенная масштабируемость системной сети за счет топологии 3D-тор;
– повышенная эффективность реализации синхронизации за счет отдельной аппаратной сети синхронизации;
– возможность аппаратной поддержки в системной сети не только MPI, но и новых перспективных подходов к реализации параллельных вычислений.

Разработка суперЭВМ ряда 4 (СКИФ-Аврора) семейства СКИФ ведется широкой кооперацией организаций, разработчиков суперкомпьютерных технологий. Непосредственно в разработке участвуют группы из семи организаций: ИПС имени А.К. Айламазяна РАН (головной), ИПМ имени М.В. Келдыша РАН, ОАО «НИЦЭВТ», ЮУрГУ, ООО «Альт Линукс Технолоджи», ЗАО «РСК СКИФ», ОИПИ НАН Беларуси. В создании, адаптации и оптимизации системного и прикладного программного обеспечения для суперЭВМ ряда 4 (СКИФ-Аврора) семейства СКИФ участвуют двадцать российских организаций. Эта кооперация разрабатывает в России большинство ключевых решений СКИФ-Аврора, в том числе:

– отечественную системную сеть с топологией 3D-тор и соответствующее программное обеспечение для нее;
– поддержку совместного использования в счете стандартных процессоров и FPGA-ускорителей;
– средства оптимизации синхронизации и массовых операций при помощи аппаратуры системной сети и сети синхронизации;
– средства мониторинга и управления суперЭВМ СКИФ-Аврора;
– перспективные подходы к реализации параллельных вычислений.

В разработке суперЭВМ ряда 4 (СКИФ-Аврора) семейства СКИФ используется равноправное сотрудничество с западными партнерами — речь идет об альянсе с компанией Евротех (Италия). Это обеспечило получение доступа к передовым западным технологиям, основанным на опыте создания встроенных решений, и позволило серьезно улучшить такие показатели проекта как стоимость, сроки и качество разработки.

Все интересы и права России при этом учтены и защищены надлежащим образом. Российская сторона имеет право:

– изготовлять все печатные платы, все узлы и модули, суперЭВМ в целом;
– поставлять без всяких ограничений и согласований созданные суперЭВМ заказчикам;
– вносить модификации в конструкторскую документацию, создавать на ее базе новые суперЭВМ, в том числе и постепенно заменять импортные микросхемы на отечественную элементную базу — по мере ее появления.

В контексте данного международного сотрудничества российской стороной были приобретены и освоены многие технологические решения, ранее отсутствовавшие в российской инду-

стрии. При этом не только все ключевые программные, но и многие инфраструктурные решения, относящиеся к системам электропитания и охлаждения суперкомпьютера, были целиком разработаны в России с учетом требований российских технологических нормативов и ГОСТов.

Тем самым, обеспечена применимость использования технологий «СКИФ» ряда 4 (СКИФ-Аврора) для создания суперЭВМ в интересах всех отраслей использования, включая стратегические.

Планы развития суперкомпьютеров ряда 4 семейства «СКИФ» тщательно проработаны сегодня до 2012 года. Каждый год в период до 2012 года обеспечивается существенное улучшение ключевых показателей: энергоэффективность системы — отношение производительности к потребляемой электрической мощности,— и производительность модуля (полушасси) и шкафа системы — конечно, при сохранении их физических размеров.

Таблица 4. Линейки моделей суперЭВМ ряда 4 семейства «СКИФ»

Линейка моделей	**СКИФ 4/Н 2009**	**СКИФ 4/В 2010**	**СКИФ 4/С 2011**	**СКИФ 4/П 2012**
Общие сведения о модельной линейке				
Начало — конец НИОКР	2 кв. 2008 — 4 кв. 2009	2 кв. 2009 — 4 кв. 2010	2 кв. 2010 — 2 кв. 2012	2 кв. 2010 — 4 кв. 2012
Поставка суперЭВМ	с 2 кв. 2009	с 3 кв. 2010	3 кв. 2011 — 1 кв. 2012	3–4 кв. 2012
Эффективность вычислителя	0,25 Tflops/КВатт	0,36 Tflops/КВатт	1,0 Tflops/КВатт	1,37 Tflops/КВатт
Производительность 1 шкафа	24 Tflops	40 Tflops	100 Tflops	200 Tflops
Показатели суперЭВМ с производительностью 1 Pflops				
Размер вычислителя	42 шкафа	25 шкафов	10 шкафов	5 шкафов
Потребление вычислителя	4,03 МВатт	2,84 МВатт	1,0 МВатт	0,73 МВатт
Полное потребление СКЦ	6,05 МВатт	4,15 МВатт	1,5 МВатт	1,09 МВатт
Рекомендуемый предел расширения суперЭВМ (50 шкафов)				
Производительность	1,2 Pflops	2 Pflops	5 Pflops	10 Pflops
Компактные суперЭВМ (возимые, герметичные, бесшумные)				
Мини «0,5» 28×50×80 см	1,5 Tflops 6 КВатт	2,55 Tflops 7 КВатт	6,25 Tflops 7 КВатт	12,5 Tflops 17 КВатт
Мини «1,0» 56×50×80 см	3 Tflops 12 КВатт	5,1 Tflops 14 КВатт	12,5 Tflops 14 КВатт	25 Tflops 35 КВатт

По сути, каждый год предполагается разработка новой модельной линейки СКИФ-Аврора (таблица 4). При этом предусмотрено глубокое повторное использование всей предыдущей конструкторской документации и, конечно, совместимость всего ранее разработанного программного обеспечения. Каждый год в период до 2012 года в рамках указанных модельных линеек суперкомпьютеров СКИФ-Аврора обеспечивается выпуск совместимых по программному обеспечению современных суперкомпьютеров в широком спектре производительности:

– для небольших суперкомпьютерных центров и для специального применения — компактные суперЭВМ (возимые, герметичные, бесшумные) от 1.5 до 25 Tflops;
– для региональных и отраслевых суперкомпьютерных центров — суперкомпьютеров с производительностью в десятки и сотни Tflops — установки из одного или несколько шкафов;

– для национальных суперкомпьютерных центров — суперкомпьютеров с высшей производительностью — от 1 Pflops (и выше), и с разумными характеристиками (стоимость, размер помещения, электропотребление и т.п.). На данном слайде показаны оценки показателей суперкомпьютера с производительностью 1 Pflops и оценки производительности системы с вычислителем в 50 шкафов — это рекомендуемый предел расширения систем СКИФ-Аврора.

Заметим, что во всех случаях производительность указывалась только с учетом стандартных процессоров — без учета возможностей FPGA-ускорителей. А их использование в некоторых приложениях может обеспечить двух-трехкратный прирост производительности.

4.3. СКИФ ряда 4 как начало реального пути России к экзафлопсному рубежу

Дальнейшее развитие суперкомпьютерных технологий уже в среднесрочной перспективе (до 2020 г.) ставит проблемы создания и эффективного использования суперкомпьютеров экзафлопсного класса (10^{18} операций в секунду). Общие темпы движения к экзафлопсному рубежу во всем мире представлены в таблице 5.

Таблица 5. Планы зарубежных разработок экзафлопсных суперкомпьютеров

Интервал лет	Разработка необходимых технологий и создание суперкомпьютеров с производительностью:
2008–2012	1–10 Pflops
2012–2016	10–100 Pflops
2016–2019 (±1)	100–1000 Pflops

Первый этап по году завершения (2012) и по планируемому результату — разработка технологий, необходимых для создания суперкомпьютеров с производительностью до 10 Pflops и соответствующего программного обеспечения (системного, инструментального и прикладного) — хорошо согласуется с точно просчитанными планами работ (таблица 4) по созданию суперкомпьютеров СКИФ-Аврора. Последующие два этапа (2012–2016 — 10–100 Pflops и 2016–2019 — 100–1000 Pflops) связаны с качественным решением следующих основных проблем:

– достижение высокой плотности компоновки вычислителя суперкомпьютера, сокращение физической длины соединений — сокращение задержки передачи сигнала;
– снижение удельного потребления электроэнергии (КВатт/Tflops);
– разработка новых подходов к охлаждению вычислителя, обеспечение эффективного и надежного отвода тепла;
– разработка новых подходов к системной сети передачи данных (система обменов между вычислительными узлами), для обеспечения низкой задержки передачи данных, высокой пропускной способности и возможности интеграции большого числа (~106) вычислительных узлов — то есть, без видимых пределов масштабирования.
– разработка системы мониторинга и управления всех технических средств вычислителя с большим числом (~10^6) вычислительных узлов, реализация средств компенсации в реальном режиме времени отказа части оборудования с обеспечением свойств устойчивости установки в целом к отказам части оборудования;
– разработка новых архитектурных решений для суперкомпьютеров с многими миллионами процессорных ядер, в том числе с поддержкой использования неоднородных ядер и специализированных ускорителей в составе вычислительных узлов таких суперкомпьютеров;
– разработка новых подходов к организации параллельного выполнения программ для суперкомпьютеров с многими миллионами процессорных ядер; в том числе, разработка системных программных средств обеспечения автоматизации распределения вычислительной нагрузки по ядрам суперЭВМ, устойчивости прикладных программ к отказу части аппаратных средств суперЭВМ.

Начальные удачные шаги по решению данных проблем частично сделаны в рамках суперкомпьютерных программ «СКИФ» и «СКИФ-ГРИД», в том числе и в рамках создания суперкомпьютеров СКИФ-Аврора.

Заключение

Россия, стремящаяся стать развитой страной с инновационной экономикой, основанной на знаниях, нуждается в серьезном развитии собственной суперкомпьютерной отрасли. Объективно необходимо обеспечить страну суперкомпьютерной киберинфраструктурой, которая по количественным показателям в 10–20 раз больше всех сегодняшних суперкомпьютерных ресурсов России. Объективно необходимо, развив собственную суперкомпьютерную отрасль, достичь уровня, позволяющего самостоятельно выполнять разработку и производство суперкомпьютерных решений на базе собственных технологий уровня N.

У России есть команды разработчиков и необходимый задел для решения этих серьезных задач. Одной из таких команд является кооперация исполнителей суперкомпьютерных программ «СКИФ» и «СКИФ-ГРИД» Союзного государства. В рамках исполнения данных программ ранее был внесен серьезный вклад в создание отечественных суперкомпьютерных технологий и ресурсов, создан задел, позволяющий смело браться за создание суперкомпьютерных технологий транспетафлопсного, а затем и экзафлопсного уровня. Поэтому, в целях технологической модернизации и повышения конкурентоспособности высокотехнологичных отраслей промышленности России представляется важным широко использовать кооперацию исполнителей программы «СКИФ-ГРИД» (головной исполнитель — ИПС имени А.К. Айламазяна РАН), как готовую команду для выполнения работ по развитию суперкомпьютерной отрасли России, и развивать перспективную суперкомпьютерную платформу ряда 4 суперкомпьютеров семейства «СКИФ», технологии ее создания и другие результаты программы «СКИФ-ГРИД».

Автор благодарен своим коллегам, помогавшим в подготовке различных материалов, использованных в данной статье: А.А. Московскому, В.Ф. Заднепровскому, А.В. Суворинову и многим другим. Данная работа выполнялась в рамках суперкомпьютерной программы «СКИФ-ГРИД» Союзного государства и проектов по программе фундаментальных исследований Президиума РАН «Проблемы создания национальной научной распределенной информационно-вычислительной среды на основе развития GRID-технологий и современных телекоммуникационных сетей» и программе фундаментальных научных исследований ОНИТ РАН «Архитектура, системные решения, программное обеспечение, стандартизация и информационная безопасность информационно-вычислительных комплексов новых поколений».

Литература

1. Абрамов С.М., Заднепровский В.Ф., Московский А.А.. Отечественные СуперЭВМ и грид-системы. Проблемы развития национальной киберинфраструктуры в России // XII научно-практическая конференция Университета города Переславля. Программные системы: теория и приложения. Переславль-Залесский: Изд-во «Университет города Переславля», 2008, Том 1, с. 9–35. ISBN 978-5-901795-11-8.

2. Мировой рейтинг пятисот самых мощных суперкомпьютеров // Электронный ресурс — http://www.top500.org/

3. Абрамов С.М. Итоги суперкомпьютерной программы «СКИФ» Союзного государства и перспективы ее развития // В книге «Пути ученого. Е.П. Велихов». Под общей редакцией академика РАН В.П. Смирнова. М.: РНЦ «Курчатовский институт» — стр. 325–333 IBSN 978-5-9900996-1-6.

4. Абламейко С.В., Абрамов С.М., Анищенко В.В., Парамонов Н.Н., Чиж О.П. Суперкомпьютерные конфигурации СКИФ. Минск: ОИПИ НАН Беларуси, 2005, цв. ил. — 170 с. — ISBN 985-6744-19-9.

5. Национальный рейтинг пятидесяти самых мощных суперкомпьютеров СНГ // Электронный ресурс — http://www.supercomputers.ru/

6. С.М. Абрамов, В.Ф. Заднепровский, А.Б. Шмелев, А.А. Московский. 2009. Супер ЭВМ ряда 4 семейства СКИФ: штурм вершины суперкомпьютерных технологий. // Труды Международной научной конференции «Параллельные вычислительные технологии (ПаВТ'2009)», Нижний Новгород, 30 марта–3 апреля 2009 г., изд. Нижегородского государственного университета имени Н.И. Лобачевского, с. 5–16. ISBN 978-5-696-03854-4.

Опыт использования суперкомпьютера «СКИФ Аврора» для решения научно-технических задач*

А.А. Московский, М.П. Перминов, Л.Б. Соколинский,
В.В. Черепенников, А.В. Шамакина

В работе проведено сравнительное исследование производительности ряда приложений численного моделирования на суперЭВМ «СКИФ»: «СКИФ Аврора» и «СКИФ Урал», установленных в Южно-Уральском государственном университете (Челябинск), а также на кластере «Endeavor» компании Intel (DuPont, США). В качестве приложений были выбраны задача газовой динамики, задачи конечно-элементного анализа и задача конденсации наночастиц. В результате анализа результатов показано, что в большинстве случаев суперЭВМ «СКИФ Аврора» демонстрирует наилучшую производительность, в особенности в задачах, требовательных к пропускной способности подсистемы памяти.

1. Введение

Сравнительное исследование производительности и масштабируемости различных приложений крайне важно для суперкомпьютерных центров, как с точки зрения оптимизации нагрузки на существующие машины, так и с точки зрения политики закупки новых платформ. В Южно-Уральском государственном университете установлены две машины семейства «СКИФ»: «СКИФ Урал» (2008 г.) и «СКИФ Аврора» (2010 г.) В качестве задач были выбраны не стандартные наборы тестов производительности, а несколько приложений пользователей суперкомпьютерного центра ЮУрГУ. Такой выбор позволяет получить более адекватную оценку возможностей вычислительных систем. Дополнительно, при помощи специализированных инструментальных средств, нами проведен анализ особенностей приложений, обуславливающих характеристики производительности приложений.

2. Архитектура суперкомпьютера «СКИФ Аврора»

Платформа «СКИФ Аврора» изначально разрабатывалась как основа для высокопроизводительных систем большого масштаба. Целый ряд технических решений, использованных в «СКИФ» ряда 4, сдвигает баланс свойств в сторону специализации для применения именно в суперкомпьютерах. Подробно характеристики решения рассмотрены в работе [1]. Установка «СКИФ Аврора» в Южно-Уральском государственном университете является первым пилотным проектом по развертыванию системы такого класса. В данном разделе кратко описываются особенности «СКИФ Аврора» с учетом ее конфигурации в ЮУрГУ.

Проект системы, включая ресурсы систем охлаждения и бесперебойного электропитания, позволяет установить до 8 вычислительных шасси «СКИФ Аврора». Каждое шасси включает 64 двухпроцессорных узла с четырехядерными процессорами Intel Xeon X5570 (Nehalem), с рабочей частотой 2,93 ГГц. Таким образом, в рамках одного монтажного шкафа удалось собрать 2048 процессорных ядер. Максимальная теоретическая производительность системы, состоящей из одного шкафа, составляет 24 ТФлопс.

* Работа выполнена при финансовой поддержке Программы СКИФ-ГРИД (контракт № 2009-СГ-03), ФЦП «Научные и научно-педагогические кадры инновационной России» (контракт № П2036) и РФФИ (проекты 10-07-96001-р_урал_а и 10-07-96007-р_урал_а).

Рис. 1. Шкаф вычислителя «СКИФ Аврора»

2.1 Вычислительная часть

Высокая плотность упаковки процессоров в вычислителе диктует необходимость использования жидкостной системы охлаждения. Вычислительные узлы выполнены в виде печатных плат, с интегрированными на материнской плате коммуникационными, сервисными микросхемами, модулями памяти. Тестирование плат проводится на заводе-изготовителе, что уменьшает число отказов компонент при инсталляции и первичной настройке системы. Каждый узел-плата накрыт плотно прилегающей пластиной охлаждения. Пластины охлаждения оснащены быстроразъемными муфтами, что позволяет демонтировать отдельный вычислительный узел без демонтажа системы охлаждения корзины (шасси) в целом.

Каждый узел оснащен твердотельным накопителем объемом 80 Гбайт. Использование твердотельных накопителей также направлено на повышение надежности вычислителя – отказы шпиндельных дисковых накопителей составляют львиную долю причин отказов узлов в кластерных установках и вычислительных фермах.

2.2 Коммуникационные сети

Ключевым компонентом любого суперкомпьютера является его коммуникационная среда. Узлы «СКИФ Аврора» обладают суммарным каналом пропускания до 100 Гбит/с, учитывая как системную и вспомогательную коммуникационные сети. Если во вспомогательной сети используются стандартные решения Infiniband QDR, то системная сеть является оригинальной разработкой.

Системная сеть имеет топологию трехмерного тора, маршрутизаторы сети реализованы на уровне адаптеров. Суммарная пропускная способность сети в пересчете на один узел составляет 60 Гбит/с. Сеть позволяет обойтись без использования дополнительного оборудования (маршрутизаторов) и задействовать при монтаже кабели одинаковой длины, вне зависимости от размера системы. Соединения на уровне половины шасси (корзины) выполнены на соединительной плате. Трехмерная организация сети позволяет легче распределить задачи между узлами кластера при моделировании объектов реального мира (трехмерных) и распараллеливании методом декомпозиции области. Для системной сети создана реализация MPI на основе MPICH2, удовлетворяющая спецификации версии MPI 2.0.

Вспомогательная сеть – сеть Infiniband QDR (40 Гбит/с) с полной бисекционной пропускной способностью. Адаптеры сети интегрированы на платах-узлах. Маршрутизаторы первого уровня интегрированы на уровне корзин (шасси) на так называемых «корневых платах». Соединения между узлами и маршрутизатором первого уровня выполнены на соединительной плате (backplane), что существенно уменьшает количество кабелей Infiniband, подключаемых вручную при установке системы. Поскольку маршутизаторы первого уровня уже присутствуют

в системе, на втором уровне сети можно использовать относительно недорогие 36-портовые маршрутизаторы – количество Infiniband кабелей и их длина от этого не меняется.

2.3 Подсистема мониторинга и управления

Подсистема мониторинга и управления обеспечивает надежное выполнение всех функций по удаленному обслуживанию установки, за исключением функций, требующих физических манипуляций. Подсистема использует как возможности стандартных IPMI средств мониторинга, так и оригинальную разработку – сеть Servnet. Компоненты Servnet присутствуют во всех основных модулях «СКИФ Аврора»:

1. на уровне узлов интегрированы контроллер и датчики температуры и влажности;
2. на уровне «корневой» платы интегрированы датчики и контроллер управления;
3. на плате блока питания интегрированы датчики и контроллер управления питанием;
4. соединительная плата обеспечивает связь сети Servnet на уровне половины шасси.

Отличительной особенностью Servnet является возможность осуществления мониторинга даже в случае полного отключения электропитания всех основных систем – питание Servnet осуществляется независимо.

«Корневые» платы играют важную роль в системе управления установкой. Именно программное обеспечение, работающее на корневой плате, позволяет отключать и включать электропитание отдельных узлов, осуществлять мониторинг характеристик системы во время работы. Программное обеспечение «корневой платы» осуществляет вывод информации на сенсорные дисплеи, установленные в торцах шасси.

Программное обеспечение мониторинга интегрирует информацию из различных источников, включая подсистемы электропитания, охлаждения, хранения данных, отображает и хранит архив данных. Поскольку установка «СКИФ Аврора» носит экспериментальный характер, под нужды управления и мониторинга выделен отдельный сервер.

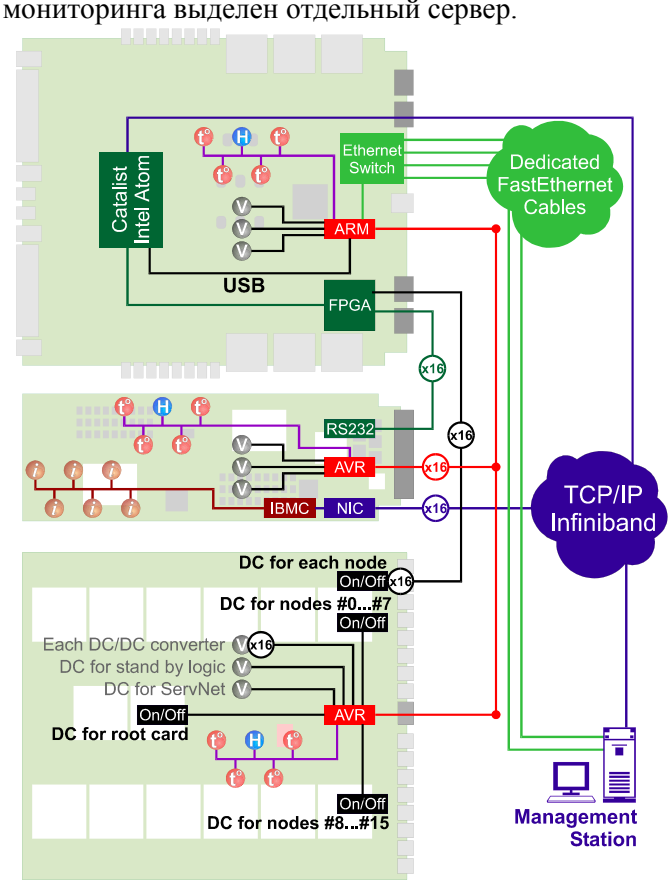


Рис. 2. Сети системы управления и мониторинга

2.4 Подсистема электропитания

Подсистема электропитания вычислителя «СКИФ Аврора» осуществляется постоянным током с напряжением 48В. За счет использования постоянного тока подсистема бесперебойного электроснабжения оказывается проще – содержит лишь выпрямитель и аккумуляторные батареи. Преобразователь постоянного тока в переменный оказывается не нужен.

Бесперебойное питание сервера мониторинга дополнительно резервировано – для обеспечения автономной работы в течение полутора часов. Таким образом, система мониторинга вполне в состоянии исполнять роль «черного ящика» вычислительной системы.

Подсистема хранения данных реализована на основе параллельной файловой системы Lustre. Общий объем подсистемы – более 50 терабайт. Теоретически подсистема должна обеспечивать производительность более 4000 операций ввода-вывода (IOPS) и пропускную способность более 500 Мбайт/сек. Узлы вычислителя имеют доступ к хранилищу данных по вспомогательной сети – Infiniband QDR.

3. Описание задач

Для исследования эффективности выполнения приложений на вычислителе «СКИФ Аврора» группой сотрудников ЮУрГУ были отобраны несколько задач. Данные задачи анализировались с точки зрения их масштабирования и оптимизации группой специалистов компании Интел (Нижний Новгород), работающей с HPC проектами. Среди приложений можно выделить задачи инженерного проектирования и анализа, решаемые с использованием стандартных инженерных пакетов, а также программный комплекс, реализованный на языке Фортран с использованием библиотеки MPI для решения задачи моделирования процессов формирования металлических наночастиц методом газофазной конденсации. Ниже представлены описания приложений.

3.1 Задача вычислительной гидродинамики тонких турбулентных слоев в щелевых уплотнениях питательных насосов электрических станций

Надежность питательного насоса определяется его вибрационным состоянием. Основным источником вибрации является неуравновешенный ротор, динамика которого в значительной мере зависит от упругих, демпфирующих и инерционных свойств турбулентной жидкости, дросселируемой в щелевых уплотнениях [2].

Щелевые уплотнения характеризуются малым зазором (0,1-0,5 мм) по сравнению с линейными размерами (для цилиндрических уплотнений – длина ~ 200 мм, диаметр ~200 мм, для радиальных – внутренний радиус ~ 140 мм, длина ~40мм), а также наличием перекоса и эксцентриситета.

Традиционно при расчетах гидродинамики тонких турбулентных слоев в щелевых уплотнениях используется укороченные уравнения Навье-Стокса (уравнения тонкого слоя), которые принципиально не позволяют определить падение давления на входном участке тонкой щели.

Использование численных методов расчета полных уравнений Навье-Стокса с Рейнольдсовым осреднением позволяет в общем виде решить задачу формирования тонкого слоя на начальном участке и течения жидкости в щели при нестационарном движении твердой стенки. Определение гидродинамических сил в тонких слоях щелевых уплотнений мощных питательных насосов требует решения системы уравнений с числом неизвестных 50-100 млн. Решение подобных задач возможно только с использованием высокопроизводительных вычислительных систем и мощных пакетов CFD.

3.2 Деформирование и разрушение тканевых бронежилетов при локальных ударах

Основной задачей при проектировании бронежилетов является минимизация их массы при сохранении заданного уровня защиты. Проверка качества бронежилета не находящегося в контакте с защищаемым объектом проводится с определением баллистического предела. А если бронежилет контактирует с защищаемым объектом (тело человека), то в этом случае существу-

ет критерий определения тупой травмы, который применяется для сравнения бронежилетов различных классов [3].

В экспериментах в качестве тела человека используют либо технический пластилин (при этом довольно сложно оценить степень травмирования тела человека), либо дорогостоящие экспериментальные модели грудной клетки. Экспериментально-аналитический путь оптимизации конструкции многослойных тканевых преград позволяет достаточно быстро определить оптимальное соотношение параметров для фиксированного воздействия (конкретных формы индентора и скорости нагружения), однако этот метод весьма затратный.

Чисто аналитических моделей, точно описывающих процесс динамического взаимодействия пули и бронежилета с учетом разрушения, на данный момент не существует и, очевидно, их получение невозможно из-за сложности физических явлений, происходящих в этом процессе: большие перемещения, скольжение, фрикционные контакты, повышение температуры. Для того чтобы учесть эти сложные физические явления, необходимо учитывать структуру баллистической ткани.

Вычислительные возможности кластеров позволяют решать сложные контактные задачи, в которых нельзя использовать механику сплошной среды. Полученные результаты и методы исследования сопротивления тканевых преград ударам огнестрельного оружия используются при разработке новых средств защиты тела человека, значительно сокращая этап предварительной оценки служебных свойств такого рода изделий.

3.3 Моделирование механического поведения грудной клетки человека при локальных ударах

При разработке персональной защитной брони минимальной массы необходимо иметь представление о механизме повреждений, которые возникают в теле человека при локальных ударных воздействиях. Поэтому учеными разных стран ведется работа по созданию теоретических и экспериментальных моделей тела человека, которые в точности повторяют форму человеческого тела и обладают такими же свойствами. Учеными Университета имени Джона Хопкинса в США (Вашингтон) были созданы конечно-элементная и экспериментальная модели грудной клетки человека, были построены ребра, грудина, хрящи, позвоночник, сердце, легкие, печень, желудок, мышцы и кожа. Если значения ускорений, полученные экспериментально и с помощью расчета, близки, то отличие давлений существенно. Таким образом, разработки теоретических и экспериментальных моделей грудной клетки человека активно продолжаются, однако какие-либо достоверные данные пока получены не были. К тому же были созданы только модели деформирования тела человека без учета степени травмирования [3].

Для того чтобы использовать численную модель грудной клетки человека для проектирования бронежилетов необходимо знать механические свойства всех ее элементов. Идентификацию параметров грудной клетки можно провести, сопоставив экспериментальные и расчетные перемещения при статическом нагружении, и ускорений, спектра собственных частот колебаний при динамическом нагружении. При этом динамическое нагружение грудной клетки реального человека должно быть низкоскоростным, чтобы не нанести травм человеку.

Экспериментальные модели грудной клетки человека имеют высокую стоимость, поэтому численное решение данной задачи является актуальной проблемой. При численном исследовании задачи возможно оценить степень травмирования грудной клетки человека.

3.4 Деформационные изменения структуры трикотажных полотен на различных участках фигуры человека

Сегодня изделия из трикотажного полотна имеют широкое распространение, поэтому актуальным является вопрос быстрого и качественного проектирования новых моделей. Трикотажные изделия значительно растягиваются при эксплуатации, причем не одинаково на разных участках тела человека, к тому же в изделии присутствуют различные виды швов (стачные, окантовочные, в подгибку с открытым срезом), которые имеют другие механические свойства. Поэтому при разработке трикотажных изделий использование геометрического метода является некорректным [4].

В настоящее время используют параллельные алгоритмы для изучения поведения тканей. Проектирование с использованием суперкомпьютеров позволяет значительно сократить материальные затраты и время на разработку нового изделия. В виртуальной модели можно легко менять различные параметры: механические свойства ткани и швов, геометрию тела человека и изделия.

3.5 Моделирование процессов газофазной конденсации металлических наночастиц

В производстве микро и наночастиц различных веществ часто используется метод «самосборки» частиц при их конденсации в пересыщенном паре в атмосфере инертного газа. При этом для дальнейшего использования полученного наноразмерного порошка необходимо соблюдение определенных требований к размеру частиц. Для этого необходимо задать определенный температурный режим в рабочей камере реактора, давление и вид инертного газа, геометрию установки, а также длительность производственного цикла. В настоящее время все эти параметры подбираются экспериментально, методом «проб и ошибок». В таких условиях очень сложно осуществлять управление технологическим процессом и прогнозировать выходное распределение частиц по размерам. Разрабатываемые математическая модель и программный комплекс направлены на решения данных задач [5].

Обычная схема формирования металлических наночастиц конденсацией из газовой фазы выглядит следующим образом: в камеру с охлаждаемыми стенками накачивается инертный газ, вблизи дна камеры помещается нагреваемый сосуд с кипящим жидким металлом, который служит источником атомов металла – мономеров. Испаряясь с поверхности жидкости, металлический пар распространяется в объеме камеры, где, по мере его охлаждения, могут возникнуть высокие степени пересыщения, являющиеся необходимым условием нуклеации.

Задача математического моделирования заключается в численном анализе процессов образования металлических наночастиц с целью прогнозирования размеров этих частиц, полученных при различных условиях: перепадах температуры в камере, давлении инертного газа, вида инертного газа, геометрических размеров рабочей камеры и испарителя и, возможно, расположения в камере охлаждаемых поверхностей, на которых будет происходить инерционное осаждение наночастиц. Решение задачи состоит из двух частей: первая заключается в моделировании конвективных течений в камере и расчете распределения температуры, плотности и концентрации мономеров в газовой смеси, вторая состоит в численной симуляции образования кластеров при их объемной конденсации в атмосфере инертного газа.

4. Анализ вычислительных экспериментов

Исследования эффективности выполнения приложений проводились на трех вычислительных системах: «СКИФ Урал», «СКИФ Аврора» и «Endeavor» (DuPont, США). Характеристики данных систем:

1. «СКИФ Урал»: Intel Xeon E5472 (Harpertown) 4 ядра на сокет, 2 процессора на узел, тактовая частота 3.00 ГГц.
2. «СКИФ Аврора»: Intel Xeon X5570 (Nehalem) 4 ядра на сокет, 2 процессора на узел, тактовая частота 2.93 ГГц
3. «Endeavor»: Intel Xeon X5670 (Westmere) 6 ядер на сокет, 2 процессора на узел, тактовая частота 2.93 ГГц.

Рассмотрим более подробно результаты запусков задач на данных вычислителях.

4.1 Задача вычислительной гидродинамики тонких турбулентных слоев в щелевых уплотнениях питательных насосов электрических станций

Для задачи вычислительной гидродинамики тонких турбулентных слоев в щелевых уплотнениях питательных насосов электрических станций приведены графики ускорений на рис. 3, 4. Исходный файл для решателя инженерного пакета ANSYS CFX содержит 29 тыс. элементов сетки.

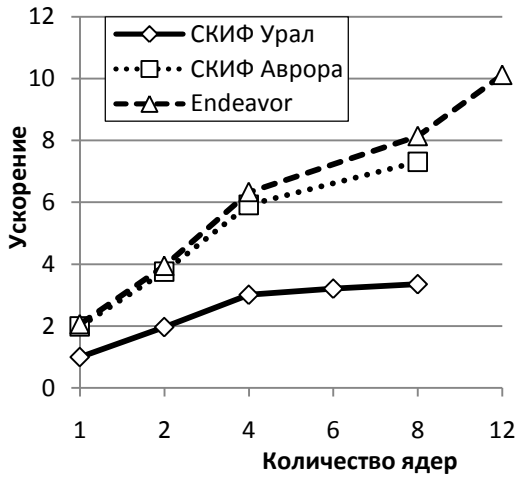


Рис. 3. График ускорений задачи гидродинамики тонких турбулентных слоев в пределах одного узла

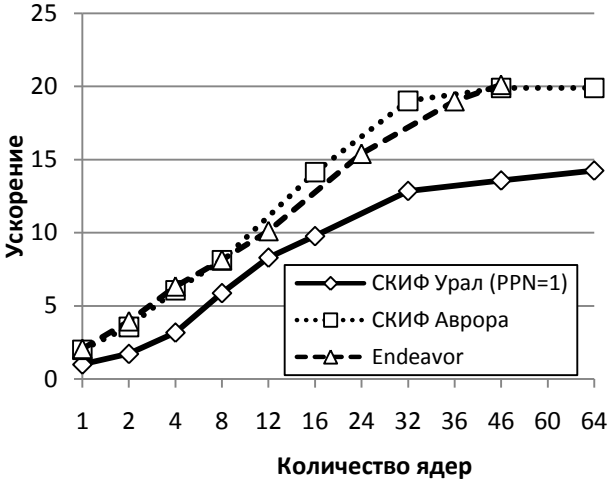


Рис. 4. График ускорений задачи гидродинамики тонких турбулентных слоев в пределах вычислительной системы

Отметим, что исследуемое приложение имеет небольшие размеры, и насыщение шкалируемости на вычислительных системах происходит достаточно быстро (рис. 4).

Основным достоинством процессоров поколения Nehalem и Westmere является многократно увеличенная пропускная способность подсистемы памяти. Если проводить сравнение с помощью теста Stream, то заметим, что пропускная способность увеличилась с типичных 10.5 Гбайт/с для систем Harpertown до 38 Гбайт/с для системы Nehalem (с памятью DDR3-1333). Соответственно, в предельных случаях на некоторых задачах вполне возможен прирост производительности в 3.5 раза. В свою очередь, увеличение пропускной способности в Nehalem вызвано несколькими причинами:

1) отказом от шинной архитектуры в пользу решения типа NUMA;
2) интегрированием контроллера памяти в сам процессор;
3) увеличением числа поддерживаемых каналов обмена с памятью с 4 до 6 (на систему из 2-х процессоров);
4) переходом на технологию памяти DDR3.

Первые два пункта увеличили реальную эффективность (40%-60%), а последние два – ее теоретическую пропускную способность с 25.6 Гбайт/с до 64 Гбайт/с. В процессоре Westmere эффективность была еще улучшена, это дало пропускную способность около 42.5 Гбит/с на тесте Stream (эффективность уже 66% от теоретического значения).

Принимая во внимание то обстоятельство, что средний поток данных в этой задаче ~20 Гбайт/с легко объяснить полученные результаты производительности. Для всех систем мы наблюдаем неидеальную шкалируемость внутри одного узла, которая объясняется тем, что некоторые ядра простаивают в ожидании необходимых данных из памяти. В то же время если для «СКИФ Урал» насыщение масштабируемости происходит довольно быстро (10.5 Гбайт/с < 20 Гбайт/с), то на «СКИФ Аврора» (38 Гбайт/с > 20 Гбайт/с) и «Endeavor» (42.5 Гбайт/с > 20 Гбайт/с) производительность продолжает расти с ростом числа задействованных ядер.

4.2 Деформирование и разрушение тканевых бронежилетов при локальных ударах

Задача деформирования и разрушения тканевых бронежилетов при локальных ударах рассчитывалась для пакета размером 5x5 см из 5 слоев баллистических тканей.

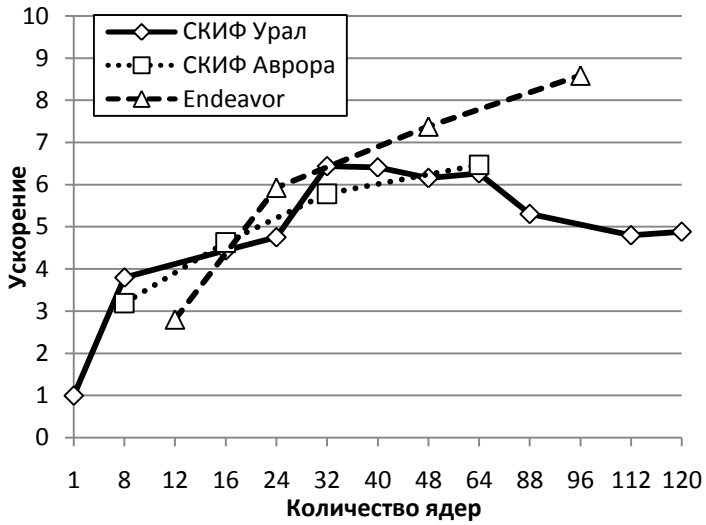


Рис. 5. График ускорений задачи деформирования и разрушения тканевых бронежилетов при локальных ударах

Данная задача относится к задачам другого типа – в ней превалируют вычисления, а взаимодействие с памятью не столь заметно. В силу этих обстоятельств, «СКИФ Урал» показывает результаты сравнимые со «СКИФ Аврора» за счет более высокой тактовой частоты. Однако «Endeavor» показывает несколько более высокие результаты за счет большего количества ядер на сокет (6 против 4). Стоит отметить, что в этом случае имеют место вычисления с одинарной точностью. При использовании двойной точности ситуация выглядела бы иначе, за счет возросшего в два раза объема взаимодействий с памятью. Обратим внимание на еще один момент, связанный с насыщением масштабируемости этой задачи. Дело в том, что исходный файл для решателя инженерного пакета LS-Dyna имеет небольшие размеры, и время, затрачиваемое на коммуникации между узлами, быстро становится сравнимым со временем вычислений.

4.3 Задачи моделирования механического поведения грудной клетки человека при локальных ударах и деформационных изменений структуры трикотажных полотен на различных участках фигуры человека

Графики ускорений для обеих задач приведены на рис. 6, 7.

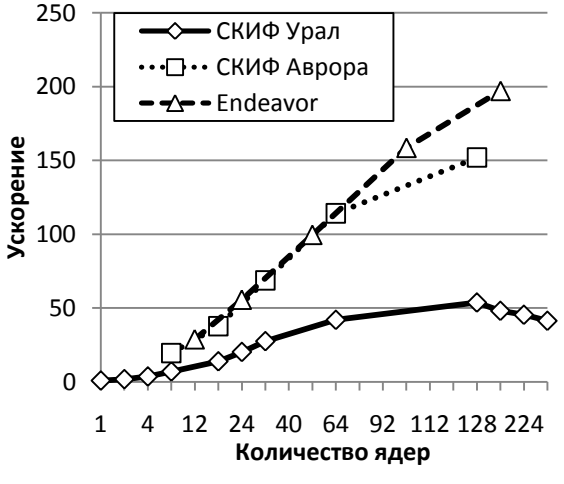


Рис. 6. График ускорений для задачи моделирования механического поведения грудной клетки человека

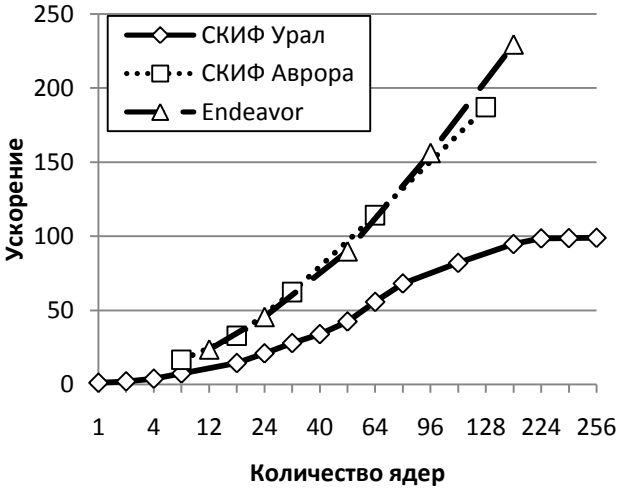


Рис. 7. График ускорений для задачи деформационных изменений структуры

Задача моделирования механического поведения грудной клетки человека при локальных ударах и задача деформационных изменений структуры демонстрируют сходное поведение, в целом типичное для HPC приложений. Производительность систем определяется правильным

балансом между их вычислительной способностью и скоростью взаимодействия с памятью. За счет улучшения этого баланса системы нового поколения показывают лучшую производительность, нежели «СКИФ Урал». Стоит также отметить, что при почти одинаковой производительности в расчете на одно ядро, «Endeavor» показывает более высокие результаты в расчете на один узел, так как имеет большее количество ядер – 6.

4.4 Моделирование процессов газофазной конденсации металлических наночастиц

График ускорений для задачи моделирования процессов газофазной конденсации металлических наночастиц приведен на рис. 8.

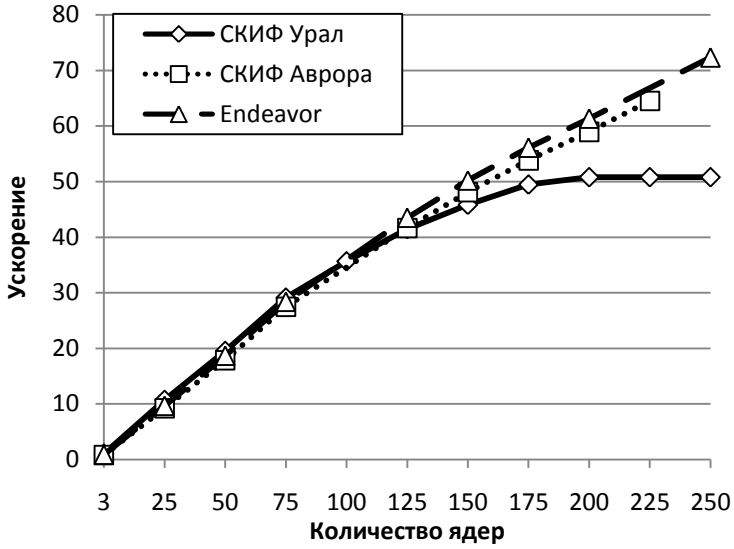


Рис. 8. График ускорений для задачи моделирования процессов газофазной конденсации металлических наночастиц

Данная задача в целом очень сходна с задачей деформирования и разрушения тканевых бронежилетов при локальных ударах, поскольку в ней зависимость от скорости общения с памятью не так велика. И все же она сказывается, хотя и достаточно неочевидным способом. На кластере «СКИФ Урал» производился запуск на меньшем количестве процессов на один узел с целью исключить конфликты обращения в память, исходящих от различных ядер внутри одного узла. Это приводит к росту числа задействованных узлов кластера, а соответственно и увеличению объема коммуникаций между ними. Увеличение объема коммуникаций в свою очередь приводит к более быстрому насыщению шкалируемости, которое наблюдается на «СКИФ Урал». Таким образом, производительность подсистемы процессор-память внутри одного узла ограничивает кластерную масштабируемость задачи.

Заключение

В данной работе рассмотрены несколько HPC приложений, обладающих различными свойствами: первая задача главным образом зависит от пропускной способности системы процессор-память, вторая задача – от вычислительной мощности системы, пятая задача чувствительна к объемам коммуникаций, а остальные сочетают в себе все вышеперечисленные свойства. Подводя общий итог, можно сделать следующий вывод. При работе с вышеперечисленными приложениями сталкиваемся с достаточно типичной ситуацией для HPC вычислений – производительность зависит не только от частоты процессоров и количества ядер на чипе, не менее важными факторами являются производительность системы процессор-память, коммуникации в распределенной системе и иногда скорость файлового ввода-вывода. Для обеспечения максимальной производительности требуется нахождение оптимального баланса этих факторов. Системы типа «СКИФ Аврора», построенная на процессорах с архитектурой Nehalem, делает значительный шаг вперед по сравнению с системой «СКИФ Урал» (архитектура процессора Harpertown), обеспечивая улучшение баланса между вычислительной мощностью процессора и

пропускной способностью подсистемы процессор-память. Система «Endeavor» (архитектура процессора Westmere) является следующим шагом на этом пути развития, улучшая как вычислительную способность (6 ядер на чипе вместо 4), так и скорость взаимодействия с памятью (42.5 Гбайт/с против 38 Гбайт/с). Все эти технологические новшества позволяют исследователям расширять «область поиска» и производить более глубокий анализ интересующих явлений за счет увеличения уровня детализации, принятия во внимание эффектов, которые прежде игнорировались.

Авторы выражают благодарность сотруднику корпорации Intel Николаю Местеру за организационную и методическую помощь при выполнении исследований, представленных в данной работе.

Литература

1. Абрамов С.М. СуперЭВМ Ряда 4 семейства СКИФ: штурм вершины суперкомпьютерных технологий // Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта - 3 апреля 2009 г.). Челябинск: Изд-во ЮУрГУ. 2009. С. 5-16.
2. Васильев В.А., Ницкий А.Ю. Сравнительный анализ области применения тестовых задач оценки вычислительной мощности HPC систем (мощных кластеров). См. настоящий сборник.
3. Долганина Н.Ю., Сапожников С.Б. Моделирование ударных процессов в тканевых бронежилетах и теле человека на вычислительном кластере СКИФ Урал. См. настоящий сборник.
4. Долганина Н.Ю., Персидская А.Ю., Усенко И.Н. Суперкомпьютерное моделирование деформационных изменений трикотажных полотен на фигуре человека. См. настоящий сборник.
5. Терзи Д.В. Моделирование процессов газофазной конденсации металлических наночастиц на вычислительном кластере «СКИФ Урал». См. настоящий сборник.

Средства программного-аппаратного мониторинга РВС

З.В. Каляев, М.К. Раскладкин

В статье рассматривается программно-аппаратный комплекс контроля критических параметров реконфигурируемых систем. К данным параметрам относятся: напряжение, ток, температура, состояние обдувающих элементов, потребляемая мощность и другие. Контролируемые параметры имеют несколько пороговых зон. При входе параметра в некоторую зону системой принимается решение о выполнении тех или иных действий по предотвращению выхода из строя аппаратуры РВС. Разработанный комплекс программно-аппаратных средств используется в составе РВС-5, установленной в НИВЦ МГУ им. Ломоносова, позволяет сократить время на поиск и устранение неисправностей, а также предупредить выход из строя аппаратных компонент РВС.

1. Введение

Решение ресурсоёмких вычислительных задач на реконфигурируемых вычислительных системах диктует необходимость контроля критических аппаратных параметров данных систем. К таким параметрам относятся температура, напряжение и ток, как отдельных аппаратных компонентов, так и РВС в целом.

Конструктивные решения для установки вычислительных узлов в стойки обладают средствами контроля стоек в целом, однако отдельные узлы или аппаратные компоненты не контролируются данными средствами, что зачастую приводит к сложности инспектирования сбоев и, как следствие, простаиванию вычислительного ресурса.

Для организации контроля аппаратной составляющей реконфигурируемых вычислительных систем разработана программно-аппаратная система контроля критических параметров.

К основным функциям данной системы относятся следующие:

- анализ работоспособности обдувающих элементов вычислительных блоков;
- анализ температуры, напряжения и тока вычислительных ПЛИС и блоков питания вычислительных модулей;
- анализ потребляемой мощности базовых модулей (БМ) и вычислительных блоков;
- действия по предотвращению выхода из строя аппаратуры вследствие сбоя, содержащие несколько уровней защиты: предупредительный, программный аварийный и аппаратный аварийный;
- программирование пороговых значений критических параметров аппаратуры;
- программирование действий по предотвращению выхода из строя аппаратуры;
- протоколирование и визуализация контролируемых параметров;
- оповещение с использованием существующих средств связи (локальные сети, Интернет, сотовые сети).

2. Теоретическая часть

С точки зрения формализованного описания программа, реализующая алгоритм мониторинга параметров РВС, представляет собой детерминированный конечный автомат.

Входными данными такого автомата будут являться сигнал отказа вентиляторов (Fan_alarm), температура (Temperature) и напряжения на выходе блоков питания: напряжение питания ядра (V_{CCINT}), дополнительное напряжение (V_{CCAUX}) и напряжение блоков ввода-вывода (V_{CCO}) каждой вычислительной ПЛИС; также к входным данным относятся напряжение (V_{48}) и ток ($I_{CURRENT}$) на входе блоков питания.

Выходные данные подразделяются на два типа: сигналы управления и сообщения о выходе входных параметров за допустимый предел.

К сигналам управления относятся: сигнал сброса конфигурации ПЛИС (Program), сигнал выключения тактовой частоты на вычислительных модулях (Clk_stop) и сигнал выключения питания вычислительных модулей (Power_off).

Сообщения о выходе входных параметров за допустимый предел состоят из двух групп: сообщения типа «предупреждение» (Warning) и сообщения типа «авария» (Alarm). Сообщения по выбору пользователя могут выводиться на экран, протоколироваться в файле или производить звуковое оповещение.

Задача автомата состоит в управлении выходными данными так, чтобы входные данные не превысили заранее заданных значений. Схема алгоритма приведена на рис. 1.

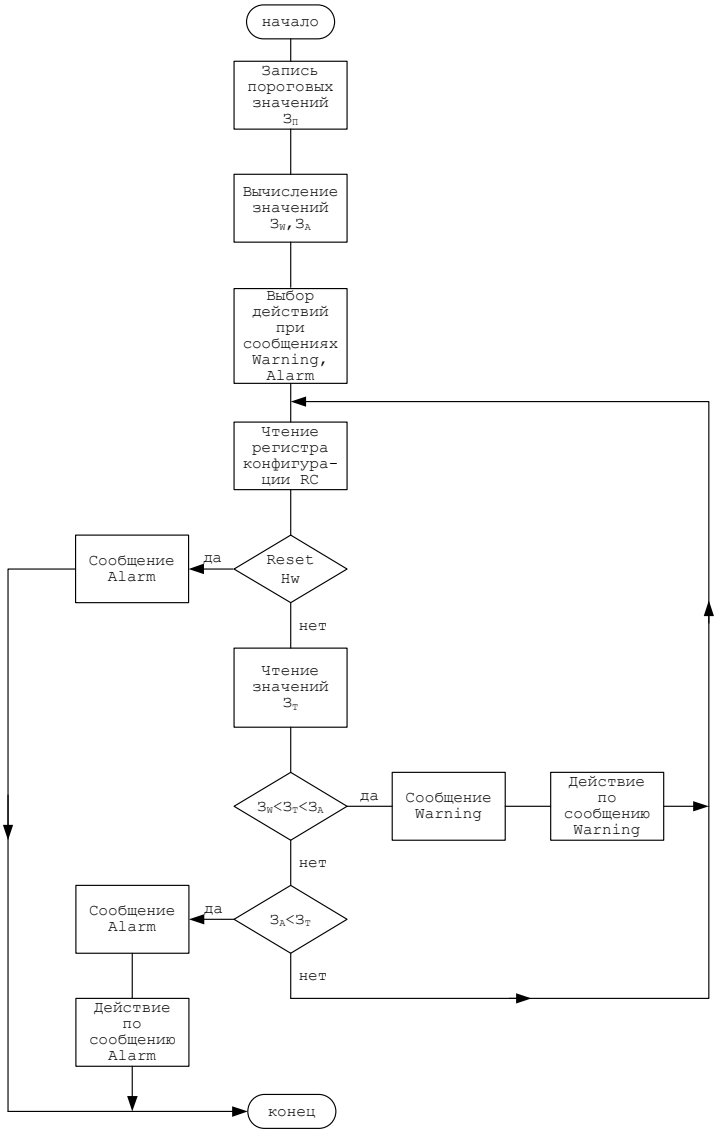


Рис. 1. Схема алгоритма программы

После запуска программы происходит запись пороговых значений ($З_П$) контролируемых параметров (Temperature, V_{CCINT}, V_{CCO} и др.) в вычислительные ПЛИС, при достижении значений которых должен произойти аппаратный сброс конфигурации вычислительных ПЛИС.

Производятся вычисления значений контролируемых параметров для диапазона значений типа Warning по следующей формуле

$$З_W = З_П - З_П \cdot \Delta,$$

где Δ=0,05 и может задаваться пользователем.

Значений типа Alarm вычисляются по следующей формуле:

$$З_А = З_П + З_П \cdot \Delta.$$

Также пользователь может выбрать действие, совершаемое программой при достижении предупреждающих и аварийных сообщений (управление сигналами Program, Clk_stop, Power_off и др.).

Производится считывание конфигурационного регистра (RC) базового модуля РВС, и в случае срабатывания аппаратного сброса (Reset Hw) конфигурации вычислительных ПЛИС программа выводит сообщение типа Alarm на экран и завершает работу, иначе производятся считывание текущих значений ($З_Т$) контролируемых параметров и вывод данных на экран.

Если считанные значений ($З_Т$) превышают $З_W$, но менее $З_А$, программа выдает сообщение типа Warning и производит действия, определенные ранее пользователем. Если считанные значения не превышают порог $З_А$, происходит продолжение мониторинга контролируемых параметров. В случае превышения порога $З_А$ выдается сообщение типа Alarm, производятся действия, определенные ранее пользователем, и программа завершает работу.

3. Реализационная часть

3.1 Программная реализация

На рис. 2 приведена экранная форма программы контроля критических параметров РВС. Программа является одним из компонентов многозадачной ОС РВС [1,2].

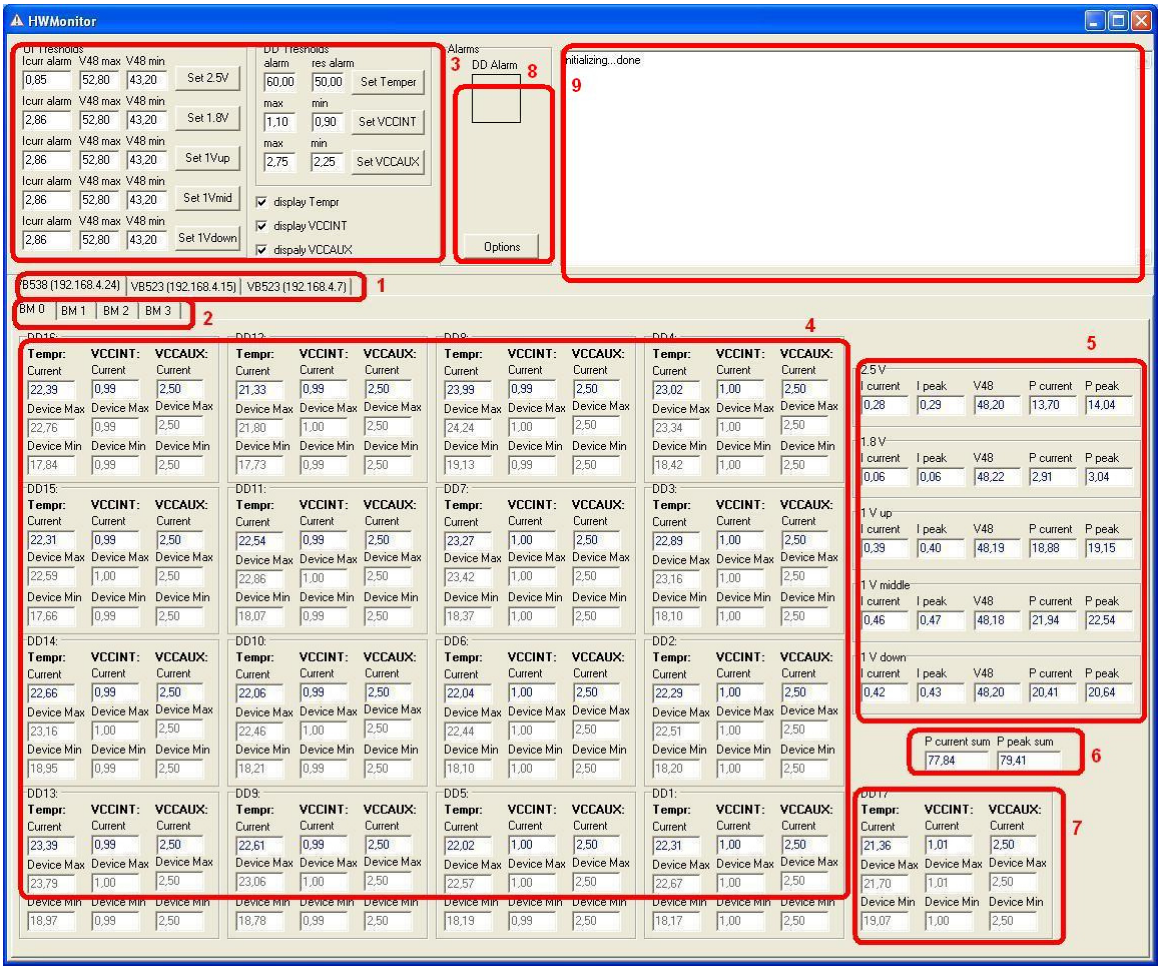


Рис. 2. Экранная форма программы контроля критических параметров РВС

Поле 1 отображает вкладки выбора контролируемых вычислительных блоков в стойке.

Поле 2 отображает вкладки выбора контролируемых базовых модулей в вычислительных блоках.

Поле 3 предназначено для программирования пороговых значений.

К программируемым пороговым значениям относятся:

- минимальное и максимальное значения температуры вычислительных и интерфейсной ПЛИС;
- минимальное и максимальное значения питания ядер вычислительных и интерфейсной ПЛИС;
- минимальное и максимальное значения дополнительного питания вычислительных и интерфейсной ПЛИС;
- минимальное и максимальное значения потребляемых токов блоков питания базовых модулей;
- минимальное и максимальное значения напряжения блоков питания базовых модулей.

Поле 4 отображает температуру, питание ядра и дополнительное питание вычислительных ПЛИС. В данном поле отображаются текущие значения, а также минимальные и максимальные значения, достигнутые во время мониторинга.

Поле 5 отображает значения тока, напряжения и мощности блоков питания базового модуля. Для тока и мощности отображаются максимально достигнутые значения.

Поле 6 отображает текущее и максимально достигнутое значения потребляемой мощности базового модуля.

Поле 7 отображает температуру, питание ядра и дополнительное питание интерфейсной ПЛИС. В поле отображаются текущие значения, а также минимальные и максимальные значения, достигнутые во время мониторинга.

Поле 8 отображает аварийный сигнал, а также имеет кнопку вызова окна программирования действий по предупреждению выхода из строя аппаратуры.

Поле 9 отображает протокол данных мониторинга со всех вычислительных блоков и базовых модулей. Данные, отображаемые дублируются в соответствующих вычислительным блокам текстовых файлах.

На рис. 3 приведена экранная форма окна программирования действий по предупреждению выхода из строя аппаратуры.

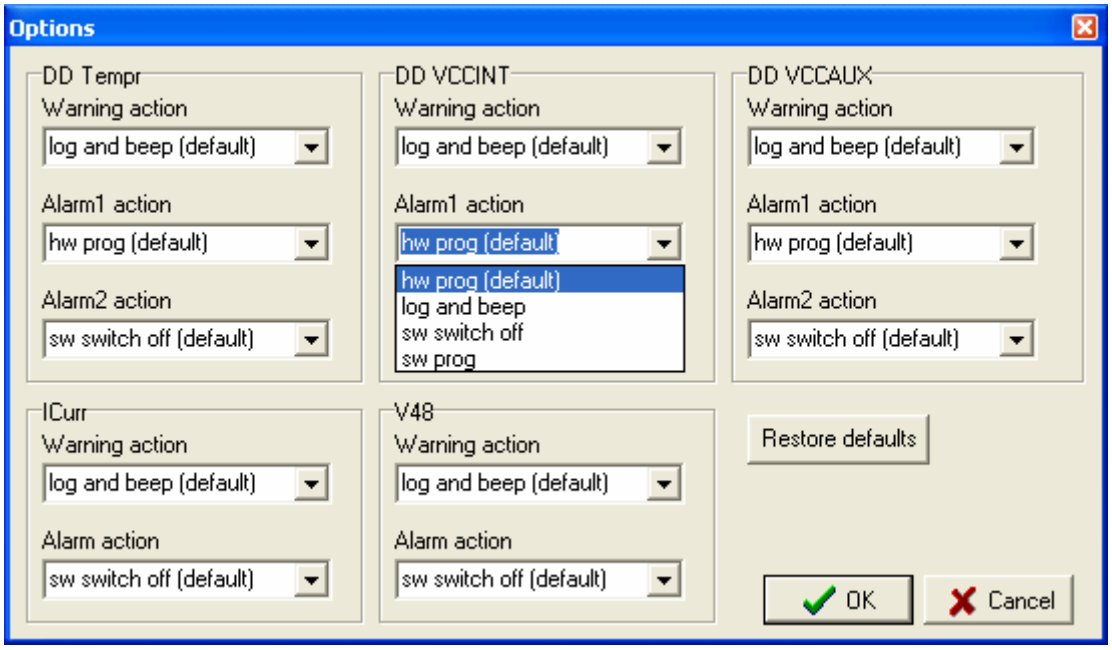


Рис. 3. Окно программирования действий по предупреждению выхода из строя аппаратуры

Как было указано выше, действия по предупреждению выхода из строя аппаратуры разделены на три зоны: зону предупреждения, аварийную программную зону и аварийную аппаратную зону.

Для программной аварийной зоны, например, можно выбрать следующие действия:

- аппаратную подачу сигнала Program на вычислительные ПЛИС;
- протоколирование и звуковое оповещение;

- программное выключение;
- программную подачу сигнала Program на вычислительные ПЛИС;
- аппаратное отключение синхронизирующей серии сигналов.

Возможно также определение других пользовательских действий по предупреждению выхода из строя аппаратуры.

На рис. 4 приведены значения температуры ПЛИС, которые вошли в зону предупреждения. Зона предупреждения задана в соответствии со значением порога в поле 1.

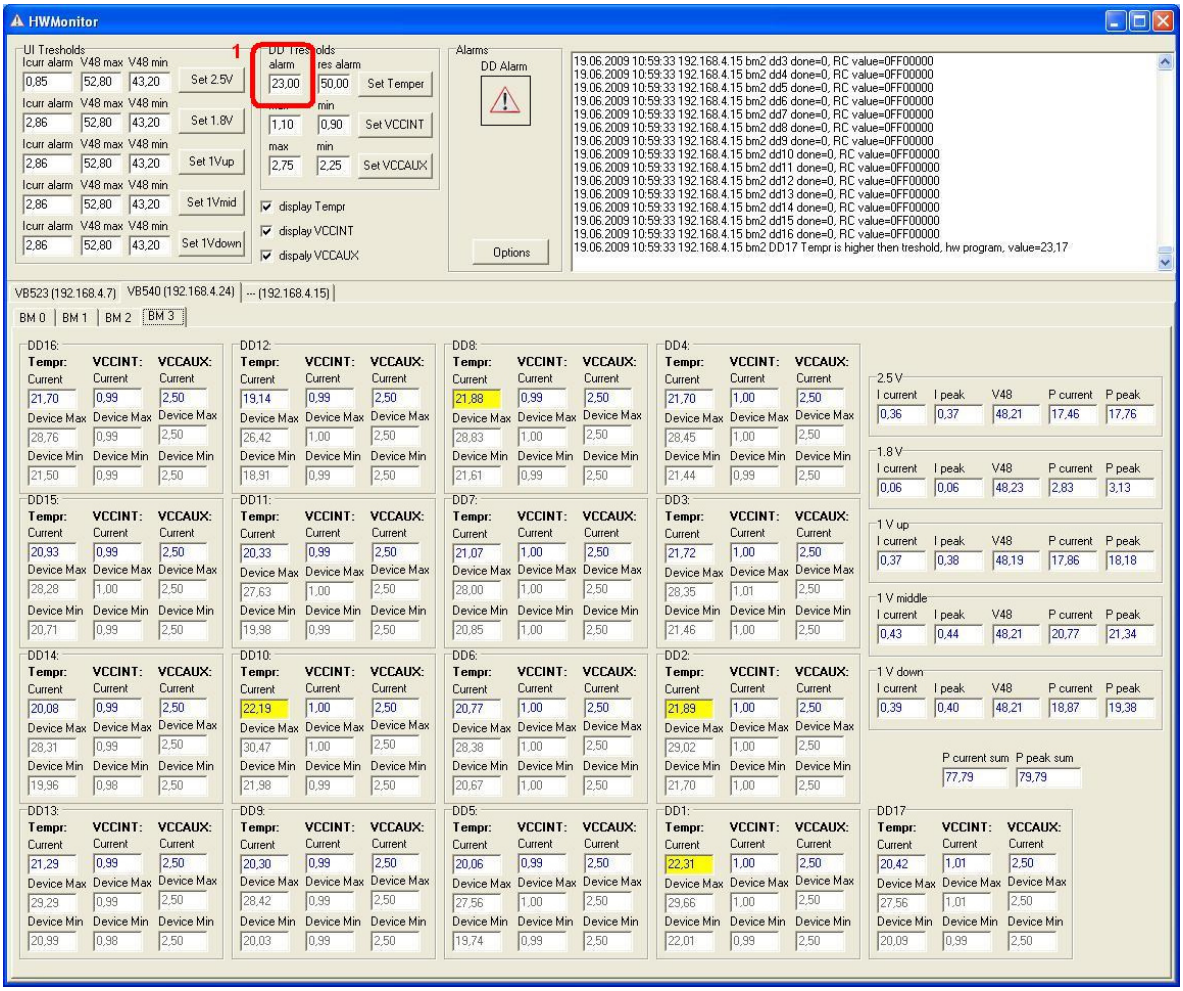


Рис. 4. Значения температуры ПЛИС, вошедшие в зону предупреждения

Как видно из рис. 4, значения, близкие к порогу, выделены желтым цветом, в окне протоколирования появились соответствующие записи. При входе значений в зону предупреждения подается звуковое оповещение.

На рис. 5 приведены значения тока блоков питания, которые вошли в зону предупреждения и в аварийную зону. Значение, вошедшее в аварийную зону и повлекшее выключение базового модуля, выделено красным цветом.

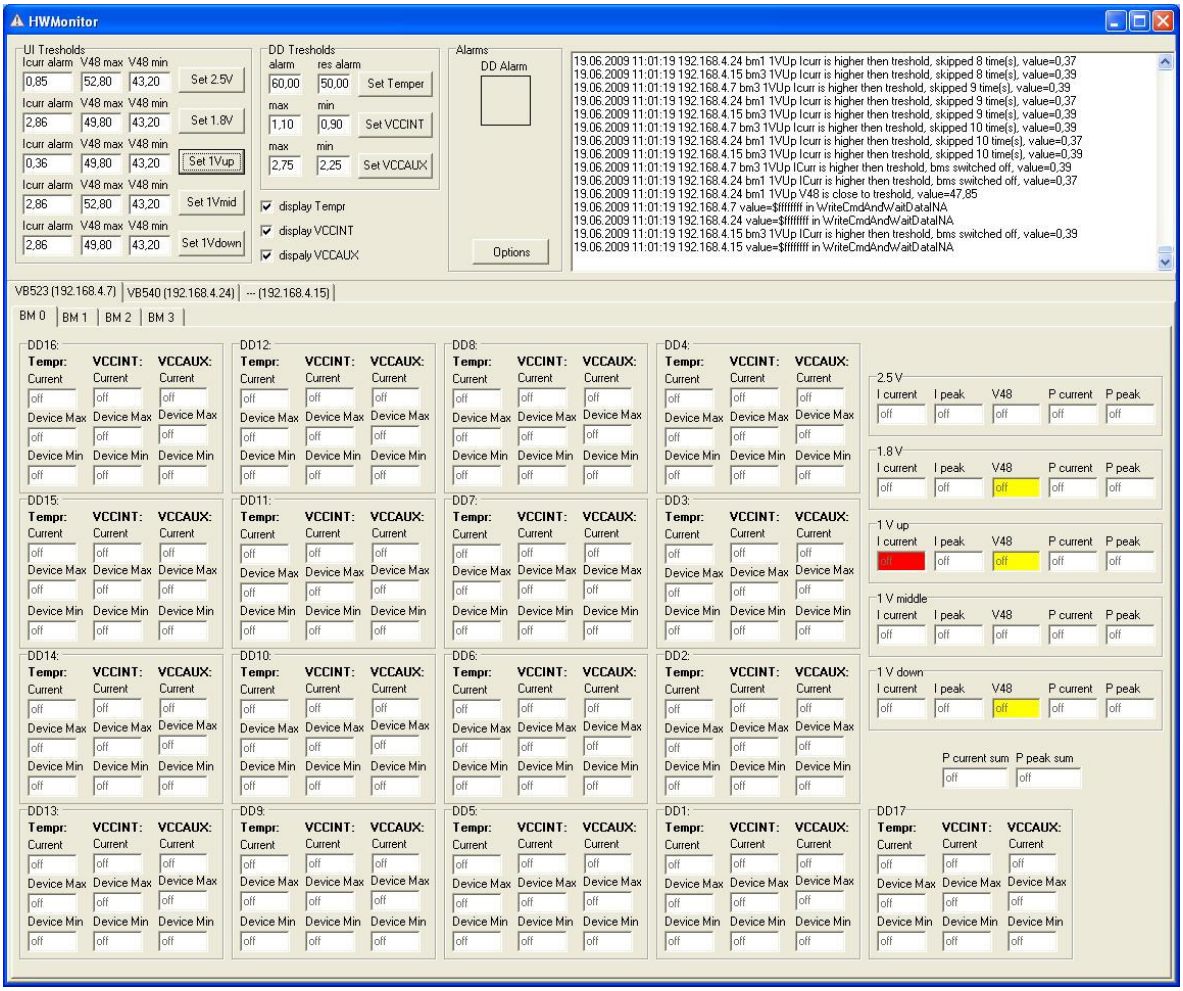


Рис. 5. Значения токов блоков питания, вошедшие в зону предупреждения и в аварийную зону

Программа осуществляет протоколирование контролируемых параметров в текстовом файле, также возможна визуализация данных параметров. В протоколе записываются следующие данные:

- время возникновения сбоя;
- описание аппаратных компонентов повлекших возникновение сбоя;
- описание принятых программой действий по предупреждению выхода из строя аппаратуры, связанных со сбоем.

3.2 Аппаратная реализация

Система питания базового модуля состоит из первичного источника напряжения, преобразующего входное переменное напряжение 220В в постоянное 48В, и вторичных источников питания, обеспечивающих питание ПЛИС и других микросхем, расположенных на базовом модуле. Такая схема обеспечивает двойную гальваническую развязку, что является существенным фактором, обеспечивающим безопасную эксплуатацию. Структурная схема системы питания базового модуля приведена на рис. 7.

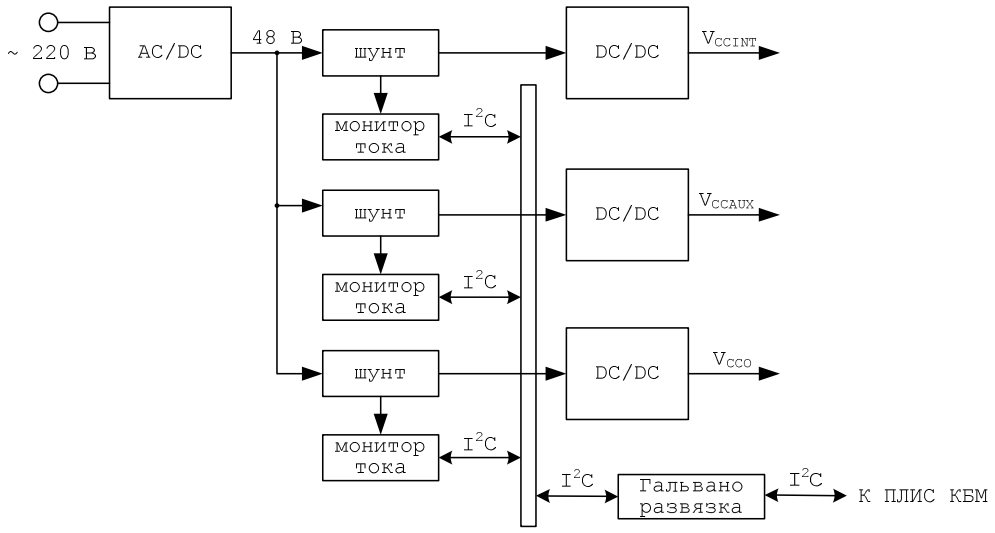


Рис. 7. Структурная схема системы питания базового модуля

Система питания базового модуля имеет в своем составе схему мониторинга потребляемого тока. Назначение этой схемы - измерять суммарный потребляемый ток каждого источника напряжения в процессе работы базового модуля. Цель мониторинга состоит в том, чтобы измерить значения потребляемых токов в процессе работы базового модуля и своевременно определить возможное состояние перегрузки по потребляемому току. Производитель ПЛИС не приводит в своих технических описаниях значения потребляемых токов, так как этот параметр существенно зависит от многих составляющих: количества задействованных в данном проекте логических ресурсов ПЛИС, тактовой частоты, примененного стандарта интерфейсов внешних связей, варианта построения электрической схемы внутри ПЛИС. Программа оценки потребляемой мощности, входящая в систему проектирования ПЛИС, не учитывает реального электрического и топологического окружения каждой микросхемы на конкретной печатной плате. Знание реальных предельных значений токов потребления позволит оптимизировать систему питания базового модуля.

Мониторинг тока ведется методом измерения падения напряжения на шунтирующем резисторе. Шунтирующие резисторы включены перед каждым источником напряжения со стороны входного питающего напряжения 48В. В качестве шунтов используются низкоомные и относительно маломощные резисторы. Дополнительное падение напряжения, которое вносят шунтирующие резисторы в цепь 48В, не отражается на значении выходных напряжений вторичных источников.

Напряжения на шунтах являются входными сигналами мониторов тока типа INA209 производства Texas Instruments. Результат измерения тока каждый монитор представляет в виде 12-битного кода, который может быть считан через стандартную I^2C шину с помощью ПЛИС контроллера базового модуля (КБМ).

Так как измерения и генерация результата проходят в цепях с напряжением 48В, а периферийные устройства КБМ питаются напряжением 2.5В, необходима гальваническая развязка информационной шины. Она достигается применением микросхемы ADUM1251, специально предназначенной для создания гальванической развязки при построении I^2C сети.

Для обеспечения возможности измерения температуры, а также напряжений питания ядра, вспомогательного напряжения и напряжения блоков ввода-вывода каждой вычислительной ПЛИС используется аппаратный системный монитор, встроенный в ПЛИС, фирмы XILINX [3] на основе десятиразрядного аналого-цифрового преобразователя. Системный монитор имеет набор регистров, в которых содержатся измеренные значения температуры и напряжений. Также имеются управляющие регистры, запись в которые позволяет установить пороговые значения для измеренных параметров [4]. Запись в эти регистры осуществляется программой контроля критических параметров РВС при ее запуске.

В случае если измеренные параметры превышают установленное значение, происходит активации аварийного сигнала. Этот сигнал от каждой вычислительной ПЛИС поступает в КБМ и в случае хотя бы одного активного сигнала происходит сброс конфигурации

вычислительных ПЛИС, т.е. происходит срабатывание аппаратного аварийного режима защиты.

Для обеспечения работы программы контроля критических параметров РВС аппаратная реализация КБМ содержит регистры, запись в которые позволяет осуществлять действия по предотвращению выхода из строя аппаратуры. Запись в эти регистры позволяет прекратить поступление сигналов тактовой частоты в вычислительные ПЛИС и производить сброс их конфигурации.

4. Экспериментальная часть

Для проверки работоспособности программно-аппаратных средств мониторинга РВС была разработана методика тестирования критических параметров РВС, состоящая из двух частей.

В первой части методики проверялось выполнение действий при достижении пороговых значений - как аппаратного, так и программного режима защиты.

Во второй части методики производилась оценка разница значений измерений считанных программой и значений, измеренных поверенными приборами.

Для выполнения первой части методики по параметру «температура» (Temperature) производился нагрев отдельных ПЛИС и фиксировался результат действий, выполняемых программой в случае сообщения типа Warning и Alarm (выделение цветом соответствующего окна, запись в окно протоколирования, звуковое оповещение, отключение питания базовых модулей и др.).

Для тестирования по параметрам напряжений и токов (V_{CCINT}, V_{CCAUX},V_{48}, $I_{CURRENT}$) в окне программы для задания пороговых параметров снижались соответствующие значения для каждого контролируемого напряжения или тока и, аналогично, контролировались действия в случае нахождения в зоне предупреждения, аварийной программной зоне и аварийной аппаратной зоне.

При выполнении методики по всем контролируемым параметрам программно-аппаратные средства успешно прошли тестирование.

Для реализации второй части методики с помощью поверенных приборов были произведены замеры значений напряжений V_{CCINT}, V_{CCAUX} и температуры каждой вычислительной ПЛИС, а также тока $I_{CURRENT}$ и напряжения V_{48} блоков питаний. Измерения проводились до загрузки конфигурационных файлов в вычислительные ПЛИС, после загрузки и после запуска тестовой задачи. Был произведен расчет разницы значений измеренных и зафиксированных программой. Среднее значение разницы для всех трех типов измерений не превысило погрешность измерений поверенных приборов и погрешность измерения АЦП системного монитора, встроенного в ПЛИС и микросхемы монитора тока INA209.

5. Выводы

Реализованная программно-аппаратная система контроля критических параметров РВС позволяет:

- предупредить повреждение путем своевременно предпринимаемых действий;
- сократить время на поиск и устранение неисправностей за счет контроля отдельных компонентов РВС;
- минимизировать выход из строя одних аппаратных компонентов РВС вследствие сбоя других.

Разработанная программно-аппаратная система контроля критических параметров внедрена в состав реконфигурируемой вычислительной системы РВС-1, установленной в НИИ МВС ЮФУ (г. Таганрог) и имеющей пиковую производительность $1 \cdot 10^{12}$ флопс. Также данная система входит в состав реконфигурируемой вычислительной системы РВС-5, обеспечивающей пиковую производительность $6 \cdot 10^{12}$ флопс. В состав РВС-5 входит двадцать вычислительных блоков, содержащих 80 базовых модулей. Таким образом, общее число контролируемых параметров составляет около 1600. Вычислительная система РВС-5

установлена в НИВЦ МГУ им М.В. Ломоносова (г. Москва) и эксплуатируется с ноября 2009 года.

Литература

1. Каляев З.В. Коваленко А.Г. Многозадачная распределенная операционная система многопроцессорной вычислительной системы с программируемой архитектурой. Известия ТРТУ. – Таганрог: Изд-во ТРТУ, 2006. – С. 179.
2. Каляев З.В. Компоненты многозадачной операционной системы для реконфигурируемой вычислительной системы. Материалы Третьей ежегодной научной конференции студентов и аспирантов базовых кафедр ЮНЦ РАН. – Ростов-на-Дону: Изд-во ЮНЦ РАН, 2007. – С. 140-141.
3. Virtex-5 FPGA System Monitor. User Guide UG192 (v1.4) April 25,2008. http://www.xilinx.com/support/documentation/user_guides/ug190.pdf.
4. Virtex-5 Family Overview. Product Specification DS100 (v5.0) February 6, 2009. http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf.

УДК 004.272.43

Семейство вычислительных систем с высокой реальной производительностью на основе ПЛИС

И.А. Каляев, И.И. Левин, Е.А. Семерников

В статье рассматриваются вопросы создания реконфигурируемых вычислительных систем (РВС) с высокой реальной производительностью, в которых в качестве основного вычислительного элемента используются программируемые логические интегральные схемы (ПЛИС), соединенные в большие вычислительные поля. Вычислительные структуры, которые могут быть созданы в больших вычислительных полях, составленных из множества ПЛИС, обладают значительно бо́льшим вычислительным потенциалом, чем при использовании отдельных кристаллов в качестве акселератора для универсального микропроцессора. На ряде примеров реализации базовых модулей и блоков показаны различные способы организации аппаратной платформы РВС и проводятся качественные оценки технических решений.

Ключевые слова

Реконфигурируемые вычислительные системы, программируемые логические интегральные схемы, вычислительные поля из ПЛИС, аппаратно-программные средства РВС

Известно, что высокую реальную производительность суперЭВМ с кластерной архитектурой демонстрируют, в основном, только при решении класса слабосвязанных задач, не требующих большого количества информационных обменов, в то время как при решении задач других классов их реальная производительность существенно снижается и не превышает 5-15% от декларируемой пиковой производительности системы [1-3]. Это является следствием неадекватности данной конкретной архитектуры суперкомпьютера информационной структуре решаемой задачи и невозможности адаптации его «жесткой» архитектуры под структуру задачи. Многие исследователи считают, что традиционные методы увеличения производительности кластерных суперЭВМ, такие как повышение тактовой частоты и механическое наращивание числа серийно выпускаемых вычислительных узлов на базе универсальных микропроцессоров, в настоящее время практически исчерпаны. Прорыв в направлении повышения реальной производительности суперЭВМ может быть достигнут только за счет поиска других концептуальных подходов при построении высокопроизводительных систем, обладающих практически линейным ростом производительности при увеличении аппаратного ресурса.

Недостатки суперЭВМ традиционной архитектуры могут быть устранены на пути создания высокопроизводительных реконфигурируемых вычислительных систем (РВС), которые развиваются в рамках созданной в НИИ многопроцессорных вычислительных систем имени академика А.В. Каляева Южного федерального университета (НИИ МВС ЮФУ) концепции многопроцессорных вычислительных систем с программируемой архитектурой [3, 4]. В отличие от многопроцессорных вычислительных систем с «жесткой» архитектурой, в частности, кластерных суперЭВМ, архитектура РВС может динамически изменяться в процессе функционирования. В результате у пользователя появляется возможность адаптации архитектуры вычислительной системы под структуру решаемой задачи. В качестве элементной базы для построения РВС используются ПЛИС высокой интеграции, соединенные в вычислительные поля. Вычислительные структуры, реализуемые в доступном пользователю ресурсе ПЛИС, обеспечивают высокую реальную производительность и пропорциональный рост производительности при увеличении задействованного оборудования.

Рассмотрим особенности построения высокопроизводительных РВС с большими вычислительными полями на примере старших представителей семейства РВС, созданных по Государственному контракту № 02.524.12.4002 «Создание семейства высокопроизводительных много-

процессорных вычислительных систем с динамически перестраиваемой архитектурой на основе реконфигурируемой элементной базы и их математического обеспечения для решения вычислительно трудоемких задач», выполняемого по заданию Федерального агентства по науке и инновациям в рамках Федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2012 годы».

Целью разработки являлось создание на единых архитектурных принципах семейства программно-совместимых реконфигурируемых высокопроизводительных вычислительных систем производительностью от 0,025 Тфлопс до 6 Тфлопс. В результате выполнения Государственного контракта создано семейство РВС, в состав которого входят: РВС-5 – высокопроизводительная система производительностью 6 Тфлопс; РВС-1Р и РВС-1К – системы производительностью более 1 Тфлопс; РВС-0.2-РС – рабочая станция производительностью 300 Гфлопс; РУПК-50 и РУПК-25 – ускорители персональных компьютеров производительностью 50 и 25 Гфлопс.

Старшие представители семейства РВС-5, РВС-1Р и РВС-0.2-РС создаются на принципах модульной наращиваемости и обладают почти линейным ростом реальной производительности в зависимости от увеличения аппаратного ресурса [3, 4]. Реальная производительность всех представителей семейства РВС на задачах различных классов составляет более 50% от указанной пиковой производительности.

В статье [5] подробно рассматривались конструктивные особенности созданного в рамках Государственного контракта № 02.524.12.4002 семейства РВС – компоновка и основные подсистемы, здесь же мы сосредоточим наше внимание на принципах организации основных вычислительных частей этих систем, представляющих собой вычислительные поля из ПЛИС, рассмотрим некоторые характеристики этих систем в сравнении с ранее созданными РВС.

Высокая реальная производительность старших представителей семейства РВС и почти линейный рост их производительности в зависимости от наращивания аппаратного ресурса обусловлен как архитектурными и конструктивно-технологическими особенностями построения вычислительных полей, так и организацией вычислительного процесса в них. В [3,4] показано, что с увеличением ресурсов вычислительного поля растет и эффективность РВС в целом, поэтому рассмотрим принципы построения вычислительных полей и пути наращивания их аппаратного ресурса. Все старшие представители семейства строятся на основе одного типа базового модуля – 16V5-75, имеющего следующие параметры:

Производительность (64 разряда), Гфлопс	75
Производительность (32 разряда), Гфлопс	140
Потребляемая мощность, ВА	200
Объем оперативной распределенной памяти, Мбайт	1,25
ПЛИС решающего поля XC5VLX110, шт.	16
Количество эквивалентных вентилей в ПЛИС, шт.	$11 \cdot 10^6$
Тактовая частота, МГц	250
Количество LVDS каналов, шт.	224
Скорость межмодульного обмена, Гбит/сек	>250

Структура базового модуля 16V5-75 показана на рис. 1. Вычислительное поле базового модуля содержит шестнадцать ПЛИС Virtex 5 XC5VLX110-2FF1153 фирмы Xilinx – ПЛИС, расположенных в узлах двумерной решетки 4 x 4 и соединенных между собой ортогональной системой связей по близкодействию. Связи по близкодействию позволяют существенно упростить печатную плату и улучшить ее частотные характеристики, поскольку соединения между соседними микросхемами не превышают четыре сантиметра. Данные между несмежными микросхемами передаются по транзитным каналам через промежуточные микросхемы, используя систему ортогональных связей.

Отличительной особенностью базового модуля 16V5-75 является реализация связей между ПЛИС вычислительного поля на основе стандарта LVDS.

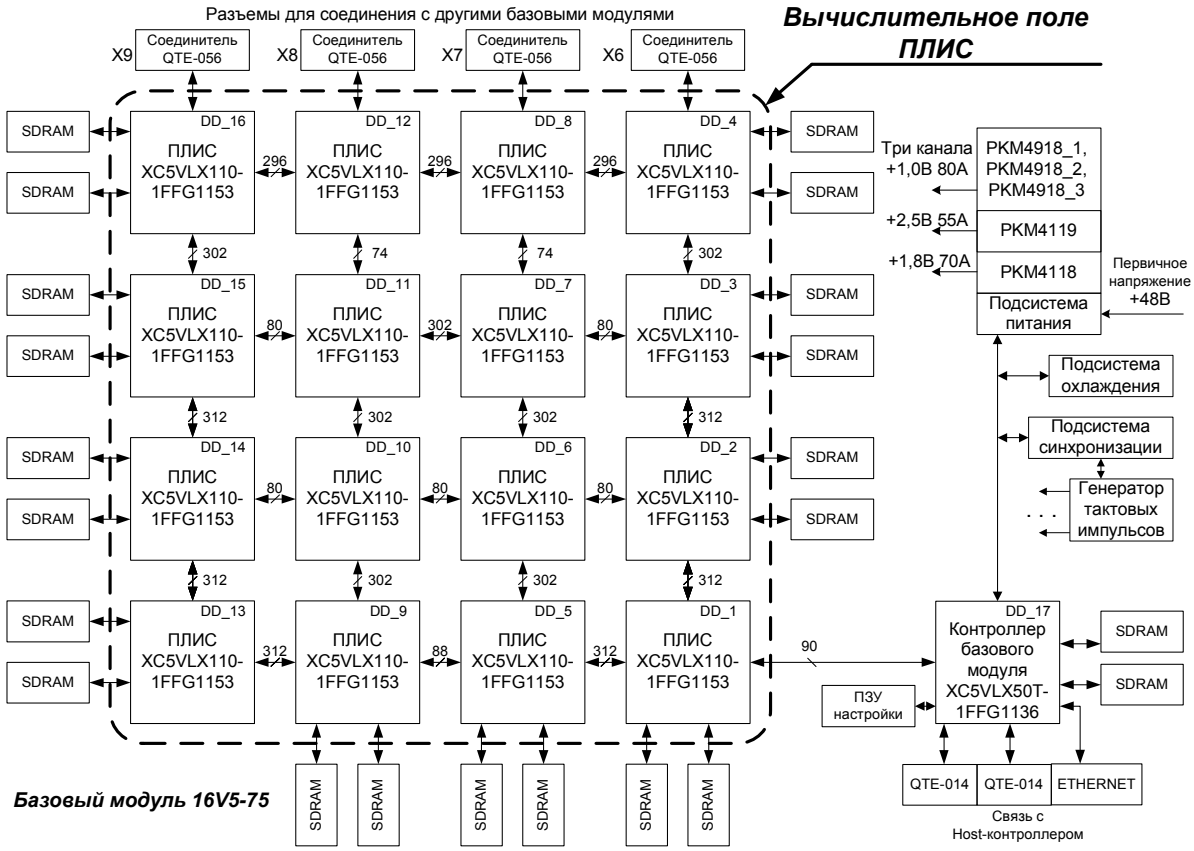


Рис. 1. Структура базового модуля 16V5-75

Стандарт LVDS позволяет снизить потребляемую мощность выходных каскадов, уменьшить уровень создаваемых электромагнитных излучений, обеспечивает невосприимчивость к синфазным электромагнитным помехам и имеет поддержку в микросхемах семейства Vertex 5 в виде аппаратно реализованных периферийных контроллеров. Физически шины связи представляют собой набор пар дифференциальных полосковых передающих линий, с обоих концов подключенных к определенным выводам микросхем. Для надежной передачи данных по LVDS в 16V5-75 задействованы специальные ресурсы семейства микросхем Vertex 5, поддерживающие алгоритм оптимальной битной и кадровой синхронизации и позволяющие учесть все нюансы линии передачи.

Для создания больших вычислительных полей базовый модуль 16V5-75 имеет специальные LVDS-разъемы QTE-056. Эти разъемы, подключенные к периферийным ПЛИС вычислительного поля базового модуля, предназначены для передачи промежуточных результатов вычислений непосредственно из микросхем данного модуля непосредственно в микросхемы вычислительных полей других базовых модулей. Передача осуществляется посредством специальных кабелей, подключаемых к соединителям типа QTE-056. Всего на каждом базовом модуле для наращивания вычислительного ресурса имеется 224 LVDS-канала, работающих на частоте 1,2 ГГц с общей пропускной способностью свыше 250 Гбит в секунду.

Базовый модуль содержит также ряд вспомогательных подсистем, которые предназначены для обеспечения его основных функций. Особое место среди них занимает контроллер базового модуля (КБМ), выполняющий функции управления всеми подсистемами базового модуля, а также функции передачи информации между базовым модулем и управляющим контроллером (ЭВМ типа IBM PC). Связь КБМ с управляющим контроллером осуществляется посредством LVDS-каналов через два разъема QTE-014, а также с помощью канала Ethernet.

На рис. 2 *а* показан внешний вид платы с установленными электронными элементами, а на рис. 2 *б* – базовый модуль 16V5-75 в сборе с подсистемой охлаждения, крепежной рамкой и кабелями LVDS.

а *б*

Рис. 2. Плата базового модуля 16V5-75 *а* и базовый модуль в сборе *б*.

Таким образом, базовый модуль 16V5-75 представляет собой мощный вычислительный узел производительностью свыше 75 (140) Гфлопс. На его основе могут строиться вычислительные блоки, содержащие от одного до восьми базовых модулей производительностью от 75 до 600 (от 140 до 1120) Гфлопс. В то же время базовый модуль обладает достаточной автономностью и может легко комплексироваться с персональным компьютером типа IBM PC в качестве ускорителя и использоваться при решении различных задач.

Первый этап наращивания ресурса вычислительного поля на основе вычислительных полей базовых модулей 16V5-75 воплощен при создании рабочей станции РВС-0.2-РС и блока РВС-0.2-ВБ производительностью свыше 300 Гфлопс. Основу этих изделий составляет объединенное вычислительное поле, включающее в себя вычислительные поля четырех базовых модулей, соединенные между собой в единый вычислительный ресурс быстрыми каналами LVDS.

Рабочая станция РВС-0.2-РС является представителем семейства РВС и предназначена для решения прикладных задач проектирования изделий микроэлектроники, управления в реальном времени сложными объектами, моделирования сложных технических и природных объектов и процессов, построения систем мониторинга, дистанционного зондирования, томографии и др.

Вычислительный блок РВС-0.2-ВБ практически полностью повторяет архитектуру рабочей станции РВС-0.2-РС, однако конструкции этих изделий значительно отличаются. Конструктивные отличия определяются назначением этих изделий: рабочая станция – это настольный вариант вычислительной системы, предназначенный для автономного использования, а вычислительный блок - это встраиваемый вариант, предназначенный для комплектования стоек СТ-1Р в составе представителей семейства РВС-1Р и РВС-5 и для создания суперЭВМ различных конфигураций. Вычислительный блок РВС-0.2-ВБ обладает теми же техническими параметрами, что и рабочая станция РВС-0.2-РС, и предназначен для решения перечисленных выше задач в составе РВС-1Р и РВС-5.

Аппаратно-программные средства РВС-0.2-РС и РВС-0.2-ВБ позволяют динамически перестраивать архитектуру в процессе решения задачи на двух уровнях: программном – на уровне элементарных процессоров и каналов распределенной памяти, обеспечивающем высокую скорость реконфигурации системы на задачи из данного класса, и схемотехническом – на уровне логических ячеек ПЛИС, обеспечивающем модернизацию системы команд элементарных процессоров и высокую удельную производительность системы при переходе на задачи различных классов.

Структура рабочей станции показана на рис. 3.

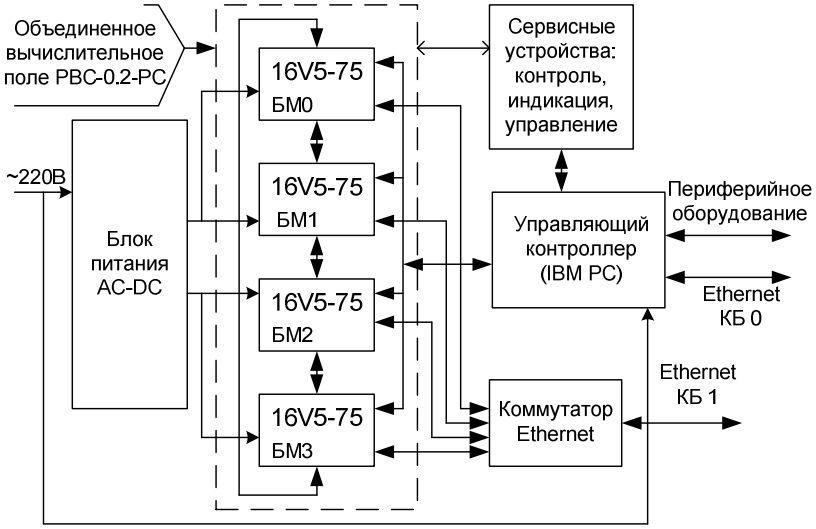


Рис. 3. Структура рабочей станции РВС-0.2-РС (РВС-0.2-ВБ)

Архитектурные отличия РВС-0.2-ВБ от РВС-0.2-РС заключаются в особенностях соединения ресурсов базовых модулей в объединенное вычислительное поле. На рис. 4 и рис. 5 показана структура связей в вычислительных полях рабочей станции РВС-0.2-РС и блоке РВС-0.2-ВБ. В вычислительном поле рабочей станции базовые модули соединяются в кольцо, а в вычислительном поле блока РВС-0.2-ВБ крайние базовые модули БМ0 и БМ3 имеют выходы за пределы блока с целью комплексирования вычислительных полей нескольких изделий РВС-0.2-ВБ в единую структуру с вычислительным полем до нескольких сотен ПЛИС в составе стойки СТ-1Р. Во всем остальном архитектура РВС-0.2-ВБ совпадает с архитектурой РВС-0.2-РС.

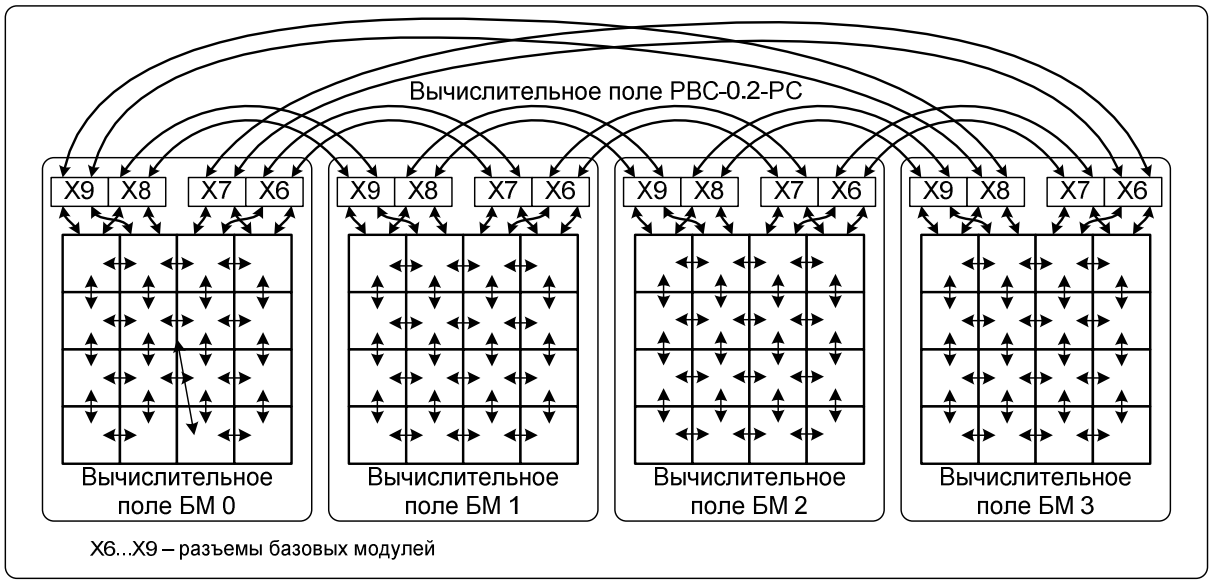


Рис. 4. Соединение базовых модулей в объединенное вычислительное поле РВС-0.2-РС

На рис. 4 и рис. 5 стрелками показаны LVDS-каналы между ПЛИС базовых модулей и LVDS-каналы, соединяющие вычислительные поля отдельных базовых модулей в общий вычислительных ресурс вычислительных полей.

Темп передачи данных между вычислительными полями любых двух базовых модулей достигает 134 Гбит в секунду. Суммарный темп передачи данных между всеми компонентами объединенного вычислительного поля РВС-0.2-РС или РВС-0.2-ВБ может достигать более 3 Тбит в секунду.

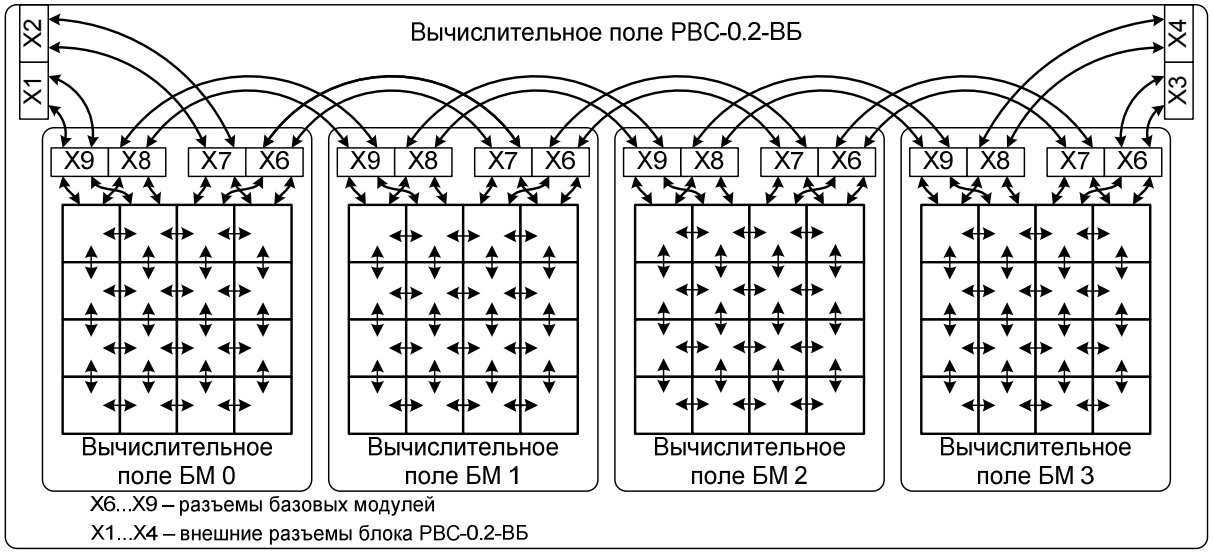


Рис. 5. Соединение базовых модулей в объединенное вычислительное поле РВС-0.2-ВБ

На рис. 6 показаны рабочая станция РВС-0.2-ВБ и вычислительный блок РВС-0.2-ВБ со снятыми верхними крышками. Кабели синего цвета на рис. 6*а* и на рис. 6*б* соединяют вычислительные поля отдельных базовых модулей в вычислительное поле РВС-0.2-РС или РВС-0.2-ВБ.

а *б*

Рис. 6. Рабочая станция РВС-0.2-РС а и вычислительный блок РВС-0.2-ВБ

Системы охлаждения РВС-0.2-РС и РВС-0.2-ВБ имеют некоторые отличия, связанные с особенностями их назначения и эксплуатации. Автономная работа рабочей станции позволяет использовать для охлаждения ПЛИС базовых модулей только медные штыревые радиаторы и проточно-вытяжную вентиляцию корпуса РВС-0.2-РС, что, в свою очередь, приводит к уменьшению габаритов рабочей станции по сравнению с РВС-0.2-ВБ, а также к уменьшению потребляемой мощности и шумности. Вычислительный блок РВС-0.2-ВБ работает в более жестких условиях из-за наличия фонового перегрева, создаваемого другими блоками и необходимостью прогонять воздушный поток не только через корпус блока, но и через корпус стойки. Поэтому для РВС-0.2-ВБ используется система с проточной вентиляцией корпуса блока, дополненная вентиляторами прямого обдува, установленными непосредственно на медных штыревых радиаторах ПЛИС базовых модулей (см. рис. 2).

Следующий этап наращивания ресурса вычислительного поля на основе вычислительных полей блоков РВС-0.2-ВБ воплощен при создании вычислительной стойки СТ-1Р, предназначенной для комплектования РВС-1Р производительностью свыше 1,2 Тфлопс и РВС-5 производительностью свыше 6 Тфлопс.

Реконфигурируемая вычислительная система РВС-1Р предназначена для: оснащения научных центров с целью проведения исследований в области физики, химии, биологии, космоса, построения информационно-управляющих систем для управления потенциально опасными производствами, решения задач аэрокосмической, автомобильной промышленности и энергетики. Пиковая производительность РВС-1Р составляет 1200 Гфлопс.

РВС-1Р включает с себя стойку СТ-1Р с подключенным к ней периферийным оборудованием. Вычислительная стойка СТ-1Р, помимо вспомогательных подсистем, содержит четыре блока РВС-0.2-ВБ, которые составляют ее основной вычислительный ресурс. Структурная схема СТ-1Р показана на рис. 7.

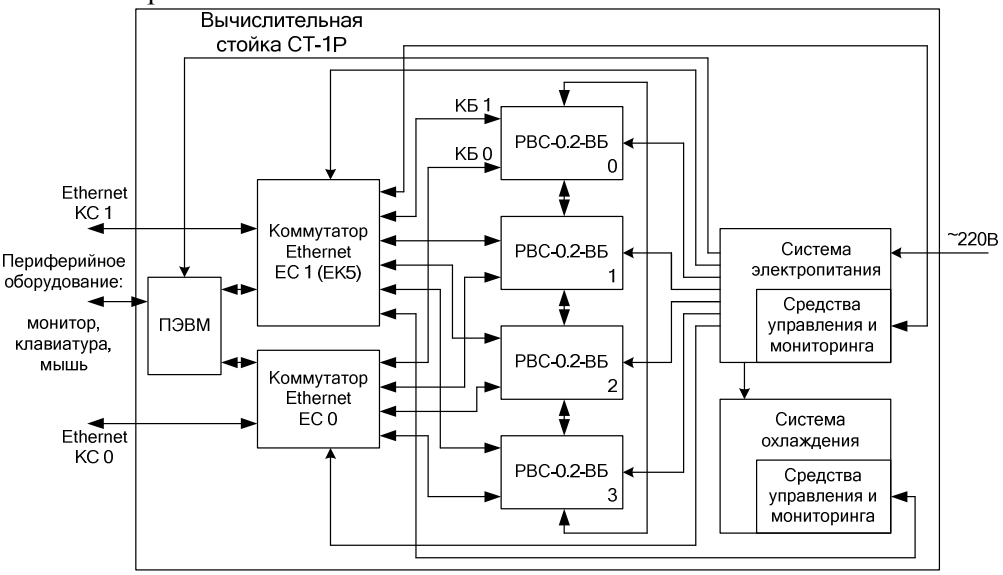


Рис. 7. Структурная схема РВС-1Р

Вычислительные поля четырех вычислительных блоков РВС-0.2-ВБ объединяются с помощью LVDS-каналов в единый вычислительный ресурс, содержащий до 16-ти базовых модулей 16V5-75 с общей пиковой производительностью 1,2 Тфлопс, как это показано на рис. 8.

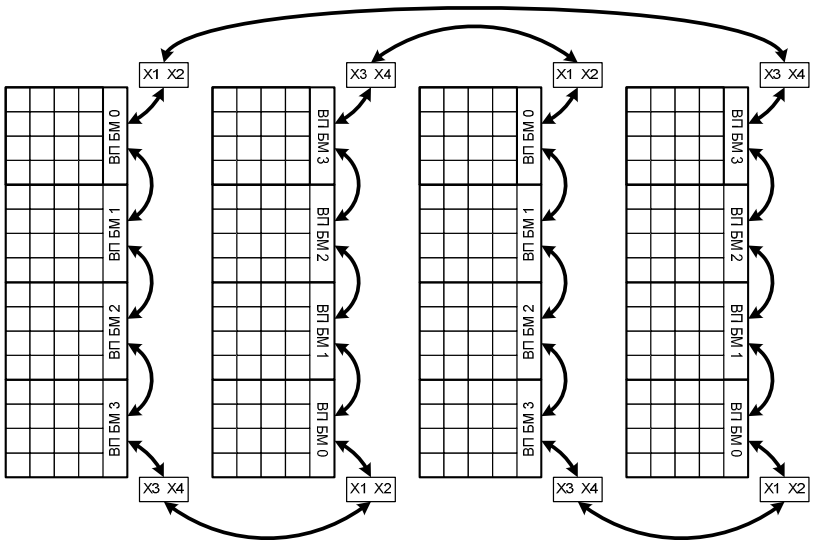


Рис. 8 Вычислительное поле стойки СТ-1Р

Межблочные связи являются продолжением межмодульных связей и, в свою очередь, продолжением связей между ПЛИС вычислительных полей базовых модулей. В целом подобная организация быстрых связей реализует в составе стойки СТ-1Р глобальный LVDS-канал передачи данных с единым темпом продвижения информации в объединенном вычислительном поле стойки, содержащем 256 ПЛИС или, с учетом их интеграции, свыше 2,8 миллиардов эквива-

лентных вентилей. Внешний вид стойки СТ-1Р вычислительной системы РВС-1Р, установленной и эксплуатируемой в вычислительном зале НИИ МВС ЮФУ, показан на рис 9.

Рис. 9. Внешний вид стойки СТ-1Р вычислительной системы РВС-1Р

Рассмотрим структуру Ethernet-связей в стойке СТ-1Р. Как было показано на рис. 3, каждый вычислительный блок РВС-0.2-ВБ имеет два канала Ethernet для связи с внешними сетями – КБ0 и КБ1. Для осуществления функций управления и мониторинга управляющие контроллеры вычислительных блоков по выходам КБ1 (см. рис. 7) соединяются между собой посредством коммутатора ЕС1 под общим управлением ПЭВМ. Сетевые выходы КБ0 блока РВС-0.2-ВБ объединяются сетевым коммутатором стойки ЕС0, посредством которого можно установить прямые связи с любым из базовых модулей в составе стойки, минуя управляющие контроллеры блоков. Такое соединение вычислительных ресурсов позволит максимально эффективно использовать возможности реконфигурируемой элементной базы вычислительной системы РВС-1Р. С одной стороны, быстрые каналы LVDS связывают вычислительные поля всех шестнадцати базовых модулей в единый вычислительный ресурс, позволяющий создавать многопроцессорную вычислительную систему со структурно-процедурной организацией вычислений в пределах четырех блоков РВС-0.2-ВБ, что дает возможность использовать все преимущества ресурсонезависимого программного обеспечения РВС. С другой стороны, система связей вычислительных блоков РВС-0.2-ВБ, благодаря сетевым технологиям, позволяет вычислительной системе РВС-1Р приобретать черты кластерной ЭВМ, где в качестве элементов кластерной системы могут выступать как блоки РВС-0.2-ВБ, так и базовые модули 16V5-75.

Старшим представителем семейства РВС является изделие РВС-5 с пиковой производительностью более 6 Тфлопс. Система РВС-5 предназначена для научно-исследовательских центров при решении прикладных задач различных предметных областей, требующих интенсивных информационных обменов, а также задач, допускающих «мелкозернистое» распараллеливание, таких как: моделирование сложных геофизических и гидродинамических процессов; цифровая обработка сигналов и изображений; молекулярное моделирование лекарств и материалов нового поколения; криптоанализ; мониторинг цифровых систем связи; томография; обработка информации и управление в реальном времени.

Вычислительная часть РВС-5 содержит пять стоек СТ-1Р, коммутатор Ethernet ЕК5 и управляющую ЭВМ (УЭВМ). К управляющей ЭВМ с целью взаимодействия с оператором и

для контроля состояния системы подключаются монитор, клавиатура, ручной манипулятор («мышь») и другие периферийные устройства.

Основным вычислительным ресурсом РВС-5 являются вычислительные поля пяти стоек СТ-1Р. Между ПЛИС вычислительных полей стоек нет непосредственных быстрых LVDS-каналов и в этом плане нет смысла говорить об объединенном вычислительном поле РВС-5. В РВС-5 одновременно функционируют пять вычислительных полей объемом 256 ПЛИС, расположенных в пяти стойках СТ-1Р. Однако фрагменты этих вычислительных полей, составленные из вычислительных полей базовых модулей 16V5-75, блоков РВС-0.2-ВБ и стоек СТ-1Р, могут соединяться множеством различных способов с помощью Ethernet-связей с использованием сетевых технологий под общим управлением УЭВМ. Один из возможных способов соединения вычислительных ресурсов РВС-5 показан на рис. 10. Приведенная схема соединения может реализовать, как один из вариантов, сеть типа 2D-тор для вычислительных полей блоков РВС-0.2-ВБ.

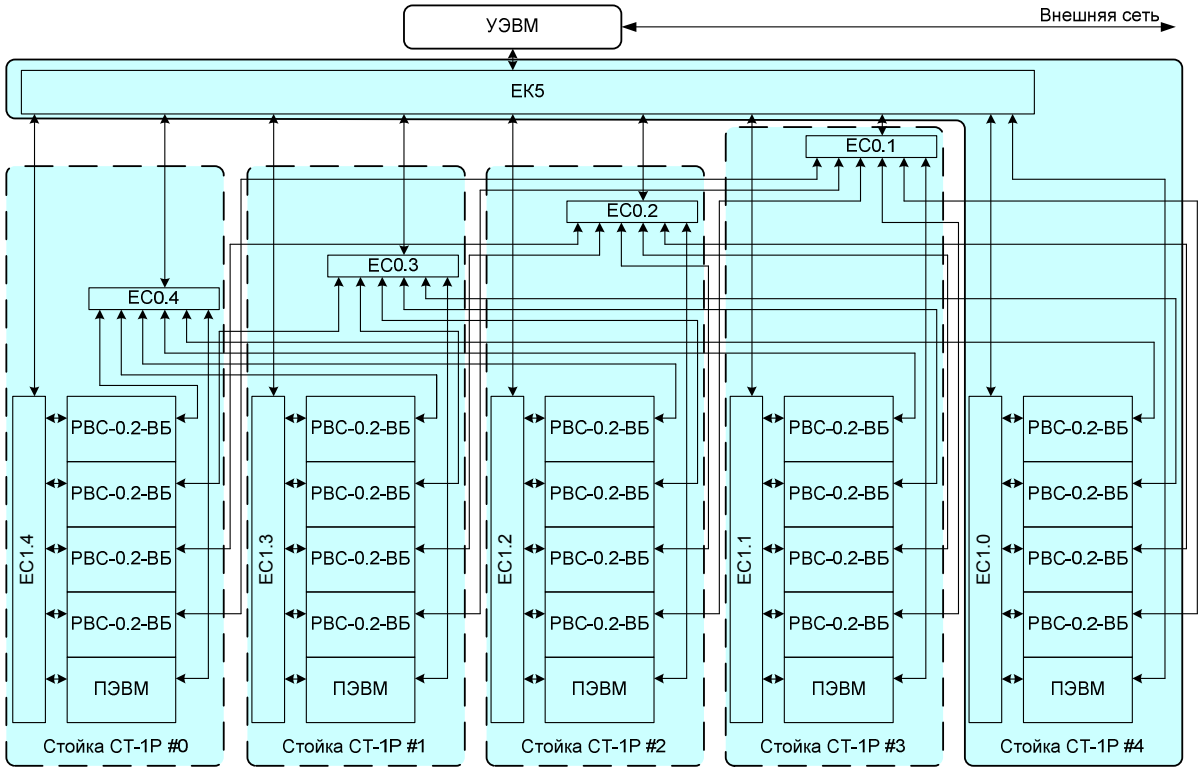


Рис. 10. Сетевое соединение фрагментов вычислительных полей РВС-5

Современная концепция построения высокопроизводительных РВС и их базовых модулей на основе ПЛИС сложилась к началу 2000-х годов, когда для этого появилась возможность использования логических матриц с интеграцией в несколько миллионов эквивалентных вентилей. В период с 2000 по 2009 годы в НИИ МВС ЮФУ были созданы десятки типов базовых модулей и более десяти РВС различной производительности и назначения на их основе. В этом плане представляет несомненный интерес процесс эволюции аппаратной платформы высокопроизводительных РВС, созданных в НИИ МВС ЮФУ, поскольку он отражает передовой научно-технический уровень в области создания систем с большими вычислительными полями на основе ПЛИС.

В качестве примера рассмотрим характеристики базовых модулей 16P25 и 16M50 [4, 6] в сравнении с характеристиками описанного выше базового модуля 16V5-75. В качестве объекта для сравнения с блоком РВС-0.2-ВБ целесообразно взять блок М200, созданный в 2006 году в НИИ МВС ЮФУ по Государственному контракту № 02.447.11.1007 в рамках федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития науки и техники на 2002-2006 гг.». Блок М200 включает четыре базовых модуля 16M50.

На рис. 11 показаны графики суммарной скорости передачи данных в каналах между распределенной памятью и вычислительным полем, в каналах межмодульного обмена и во каналах обмена между ПЛИС вычислительного поля.

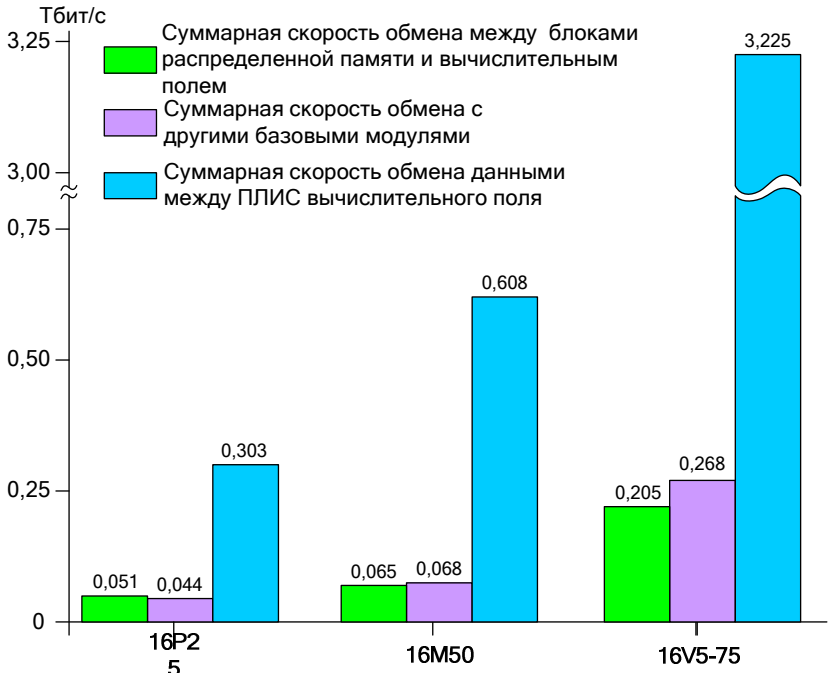


Рис. 11. Графики суммарной скорости передачи в каналах базовых модулей

В базовых модулях 16P25 и 16M50 связи между ПЛИС вычислительного поля выполнены в виде обычных соединений печатными проводниками и рассчитаны на темп передачи данных 100 и 200 МГц, межмодульные связи выполнены на основе стандарта LVDS с темпом передачи 400 и 640 МГц. На базовом модуле 16V5-75 связи между ПЛИС вычислительного поля и межмодульные связи реализованы на основе стандарта LVDS на частоте 1200 МГц, что позволило существенно (в 3 – 4 раза) увеличить суммарную пропускную способность как внутримодульных, так и межмодульных каналов передачи данных.

На рис. 12 показаны графики производительности базовых модулей для операций с плавающей запятой одинарной точности и байтных операций в секунду для задач символьной обработки.

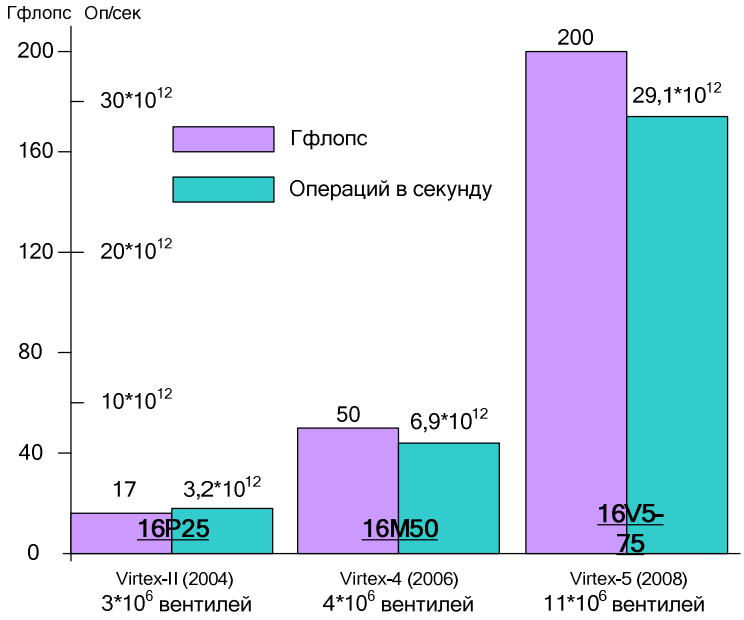


Рис. 12. Графики производительности базовых модулей

Сравним по ряду параметров вычислительные блоки М200 и блок РВС-0.2-ВБ, содержащие по четыре базовых модуля 16М50 и 16V5-75.

Н рис. 13 показаны значения реальной производительности вычислительных блоков М200 и РВС-0.2-ВБ.

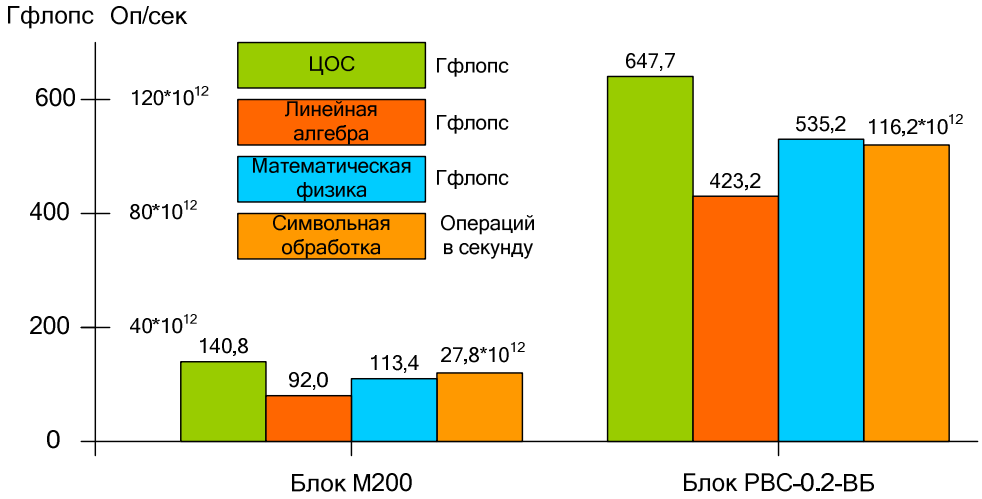


Рис. 13. Значения реальной производительности вычислительных блоков М200 и РВС-0.2-ВБ

Показатели «компактности» – отношение производительности вычислительных блоков к объему и «эффективности» – отношение стоимости блоков к производительности - приведены на рис. 14.

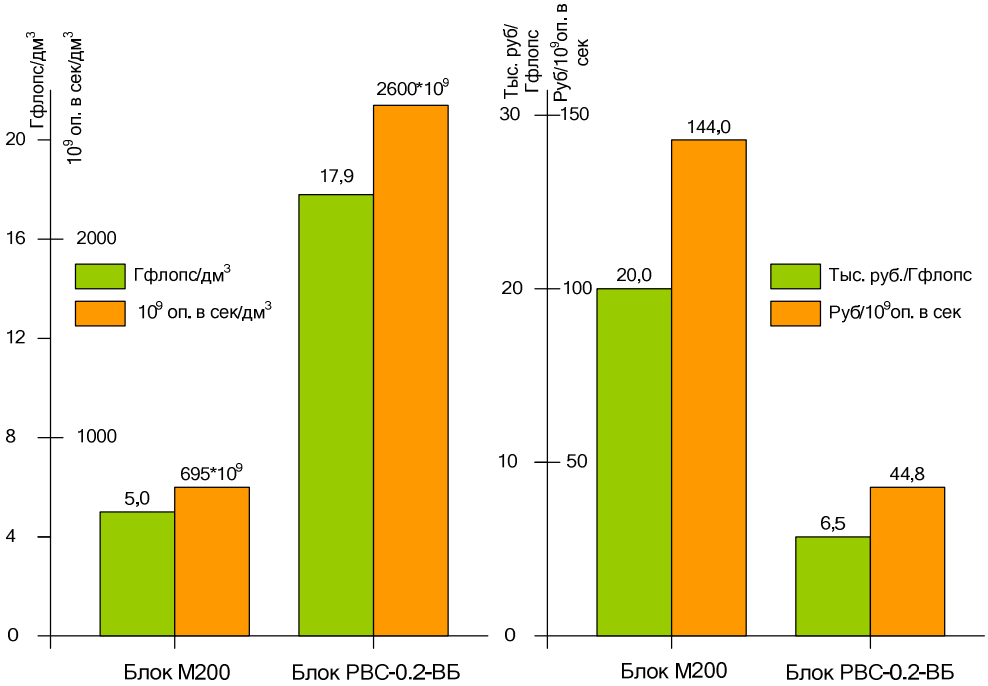


Рис. 14. Значения показателей «компактности» и «эффективности» блоков М200 и РВС-0.2-ВБ

Высокие показатели производительности, компактности и эффективности базового модуля 16V5-75 и блока РВС-0.2-ВБ на его основе достигнуты не только за счет прогресса в области ПЛИС, но и за счет целого комплекса прогрессивных технических решений, положенных в их основу. Технические параметры базового модуля 16V5-75 и блока РВС-0.2-ВБ позволили выполнить все требования, предъявляемые к представителям семейства РВС, заложенные в Государственном контракте № 02.524.12.4002. Базовый модуль 16V5-75 и блок РВС-0.2-ВБ могут

служить основой для создания РВС различных конфигураций с реальной производительностью от 200 Гфлопс до 20 Тфлопс.

Литература

1. Аладышев О.С., Дикарев Н.И., Овсянников А.П. и др. СуперЭВМ: области применения и требования к производительности - Известия ВУЗов. Электроника, 2004. - №1. – С. 13-17.

2. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. - С.-Петербург: «БХВ-Петербург», 2002. – 599 с.

3. Каляев А.В., Левин И.И. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений. - М.: Янус-К, 2003. – 380 с.

4. Каляев И.А., Левин И.И., Семерников Е.А., Шмойлов В.И. Реконфигурируемые мультиконвейерные вычислительные структуры /Изд. 2-е, перераб. и доп. / Под общ. Ред. И.А. Каляева. - Ростов-на-Дону: Изд-во ЮНЦ РАН, 2009. – 344 с.

5 Каляев И.А., Левин И.И. Семейство реконфигурируемых вычислительных системы с высокой реальной производительностью // Труды международной научной конференции «Параллельные вычислительные технологии» (ПАВТ’2009). – Нижний Новгород: электронное издание НГУ имени Н.И. Лобачевского, 2009. – С.186-196.

6. Беседин И.В., Дмитренко Н.Н., Каляев И.А., Левин И.И., Семерников Е.А. Семейство базовых модулей для построения реконфигурируемых многопроцессорных вычислительных систем со структурно-процедурной организацией вычислений // Материалы Всероссийской научной конференции «Научный сервис в сети Интернет: технологии распределенных вычислений», г. Новороссийск. – М.: Издательство Московского университета, 2006. – С. 47-49.

Сравнительный анализ производительности кластерных суперЭВМ на примере задачи о релаксации электронного пучка в высокотемпературной плазме *

А.В. Снытников

Проведено сравнение отдельных показателей производительности четырех отечественных кластеров (СКИФ Cyberia, МВС-100К, СКИФ-МГУ "Чебышёв"и кластера НГУ), достигнутых на задаче о моделировании взаимодействия электронного пучка с плазмой. Модель построена на основе метода частиц в ячейках. По результатам сравнения можно сделать вывод, что важнейшим показателем быстродействия кластера на реальных задачах является скорость доступа к оперативной памяти. Наибольшая скорость работы с помощью созданной программы достигнута на кластере СКИФ-МГУ "Чебышёв". По скорости межпроцессорных обменов лидируют СКИФ Cyberia и СКИФ-МГУ "Чебышёв".

1. АКТУАЛЬНОСТЬ РАБОТЫ

Актуальность настоящей работы связана с необходимостью повышения эффективности использования кластерных суперЭВМ. Для этого требуется определить показатели их быстродействия при решении реальных физических задач. Причина того, почему нельзя ограничиться измерениями только на универсальных тестах типа LinPack заключается в том, что существует очень большая разница между декларируемой производительностью той части кластера, которая доступна конкретному пользователю в конкретном расчете, и реально достигнутой. Например, если используется одна четвертая часть процессоров кластера с пиковой производительностью 5.4 TeraFlOp/S, то не будет большой ошибкой сказать, что пиковая производительность используемой части кластера должна быть на уровне 1 TeraFlOp/S. Фактически на рассматриваемой задаче получается 0.18 TeraFlOp/S, что вызвано не только недостатками программы пользователя, но и неприспособленностью кластера к такому типу задач.

Также возможно использование результатов данной работы при определении рейтинга кластеров. В настоящий момент списки Top50 и Top500 выстроены в порядке убывания пиковой производительности и производительности на тесте LinPack что, разумеется, дает определенную информацию о сравнительной скорости работы представленных там машин. Но очень многие факторы, такие как скорость работы и объем дисков, пропускная способность шины памяти и коммуникационной сети, неоднородность оборудования и т.д. - остаются за пределами рассмотрения. А это именно те проблемы, с которыми придется столкнуться при попытке посчитать на кластере большую задачу.

Физическая задача, на примере решения которой будет рассматриваться производительность кластеров, представляет собой следующее. На многопробочной магнитной ловушке ГОЛ-3 (ИЯФ СО РАН) наблюдается понижение электронной теплопроводности на 2-3 порядка по сравнению с классическим значением [1] в результате релаксации в плазме мощного электронного пучка. Этот эффект представляет большой интерес с точки зрения построения энергетического термоядерного реактора, и на данный момент теоретическое описание его отсутствует.

Необходимость применения суперкомпьютерных вычислений обусловлена тем, что требуется, во-первых, иметь достаточно подробную сетку для того, чтобы воспроизвести резонансное взаимодействие релятивистского электронного пучка с плазмой, и, во-вторых,

*Работа выполнена при поддержке Российского Фонда Фундаментальных Исследований, гранты 08-01-615 и 08-01-622, а также интеграционных проектов СО РАН № 103, № 113 и № 26 и гранта Президента Российской Федерации для государственной поддержки молодых российских ученых № МК-3562.2009.9.

большое количество модельных частиц для того, чтобы промоделировать возникающую в дальнейшем турбулентность.

2. ОПИСАНИЕ МОДЕЛИ

Численная модель, используемая для решения задачи о релаксации пучка, состоит из уравнений Власова для электронной и ионной компонент плазмы и системы уравнений Максвелла. В общепринятых обозначениях эта система имеет следующий вид:

$$\frac{\partial f_{i,e}}{\partial t} + \vec{v}\frac{\partial f_{i,e}}{\partial \vec{r}} + \vec{F}_{i,e}\frac{\partial f_{i,e}}{\partial \vec{p}} = 0, \qquad \vec{F}_{i,e} = \frac{q_{i,e}}{m_{i,e}}\left(\vec{E} + \frac{1}{c}[\vec{v}, \vec{B}]\right)$$

$$rot\vec{B} = \frac{4\pi}{c}\vec{j} + \frac{1}{c}\frac{\partial \vec{E}}{\partial t}$$

$$rot\vec{E} = -\frac{1}{c}\frac{\partial \vec{B}}{\partial t} \tag{1}$$

$$div\vec{E} = 4\pi\rho$$

$$div\vec{B} = 0$$

В данной работе используется алгоритм решения системы уравнений (1), описанный в работе [2]. Далее все уравнения будут приводится в безразмерном виде. Для обезразмеривания используются следующие базовые величины:

- характерная скорость $\tilde{v}$- скорость света $\tilde{v} = c = 3 \times 10^{10}$ см/с
- характерная плотность плазмы $\tilde{n} = 10^{14}$ см$^{-3}$
- характерное время $\tilde{t}$- плазменный период (величина, обратная к электронной плазменной частоте) $\tilde{t} = \omega_p^{-1} = \left(\frac{4\pi n_0 e^2}{m_e}\right)^{-0.5} = 5.3 \times 10^{-12}$ с

Уравнения Власова решаются методом частиц в ячейках (PIC). В рамках этого метода решаются уравнения движения модельных частиц, которые являются уравнениями характеристик для уравнения Власова. Величины с индексом i соответствуют ионам, с индексом e - электронам.

$$\frac{\partial \vec{p}_e}{\partial t} = -\left(\vec{E} + [\vec{v}_e, \vec{B}]\right)$$

$$\frac{\partial \vec{p}_i}{\partial t} = \kappa\left(\vec{E} + [\vec{v}_i, \vec{B}]\right)$$

$$\frac{\partial \vec{r}_{i,e}}{\partial t} = \vec{v}_{i,e}, \qquad \kappa = \frac{m_e}{m_i}, \qquad \vec{p}_{i,e} = \gamma\vec{v}_{i,e}, \gamma^{-1} = \sqrt{1 - v^2}$$

Для решения уравнений движения используется схема с перешагиванием:

$$\frac{\vec{p}_{i,e}^{\,m+1/2} - \vec{p}_{i,e}^{\,m-1/2}}{\tau} = q_i\left(\vec{E}^m + \left[\frac{\vec{v}_{i,e}^{\,m+1/2} - \vec{v}_{i,e}^{\,m-1/2}}{2}, \vec{B}^m\right]\right)$$

$$\frac{\vec{r}_{i,e}^{\,m+1} - \vec{r}_{i,e}^{\,m}}{\tau} = \vec{v}_{i,e}^{\,m+1/2},$$

здесь τ - временной шаг. Для нахождения электрических и магнитных полей используется схема, в которой поля определяются из разностных аналогов законов Фарадея и Ампера [2]. Эта схема имеет второй порядок аппроксимации по пространству и по времени.

3. ПОСТАНОВКА ЗАДАЧИ

Рассмотрим следующую постановку задачи. В начальный момент в трехмерной области решения (размер области L), которая имеет форму прямоугольного параллелепипеда:

$$0 \le x \le L_X, \qquad 0 \le y \le L_Y, \qquad 0 \le z \le L_Z$$

находится плазма, состоящая из электронов и ионов. Модельные частицы распределены по области равномерно. Задается плотность плазмы и температура электронов, температура ионов считается нулевой. Дополнительно в области присутствуют электроны пучка, которые также распределены по области равномерно (предполагается, что пучок уже вошел в расчетную область). Электроны пучка отличаются от электронов плазмы тем, что они имеют кинетическую энергию направленного движения $\varepsilon = 1$ МэВ, а их температура равна нулю. Модельные частицы, соответствующие электронам пучка, имеют меньшую массу, нежели модельные частицы, соответствующие электронам плазмы (отношение их масс равно отношению плотности плазмы и плотности пучка). Итак, исходными параметрами задачи являются: плотность и температура электронов плазмы, отношение плотности электронов плазмы к плотности электронов пучка, энергия электронов пучка:

- плотность электронов плазмы $n_0 = 10^{17}$ см$^{-3}$. Плотность плазмы намеренно завышена по сравнению с реальной плазмой на установке ГОЛ-3 для того, чтобы эффекты релаксации пучка проявлялись в течение более короткого времени;
- температура электронов плазмы $T_0 = 1$ КэВ;
- отношение плотности электронов пучка к плотности электронов плазмы $\alpha = 10^{-3}$;
- энергия электронов пучка $\varepsilon = 1$ МэВ.

4. ПАРАЛЛЕЛЬНАЯ РЕАЛИЗАЦИЯ

Распараллеливание выполнено методом декомпозиции расчетной области по направлению, перпендикулярному направлению движения электронного пучка. Используется смешанная эйлерово-лагранжева декомпозиция. Сетка, на которой решаются уравнения Максвелла, разделена на одинаковые подобласти по одной из координат. С каждой подобластью связана группа процессоров (в том случае, когда вычисления производятся на многоядерных процессорах, процессором для единообразия будет именоваться отдельное ядро). Далее, модельные частицы каждой из подобластей разделяются между процессорами связанной с этой подобластью группы равномерно, вне зависимости от координаты.

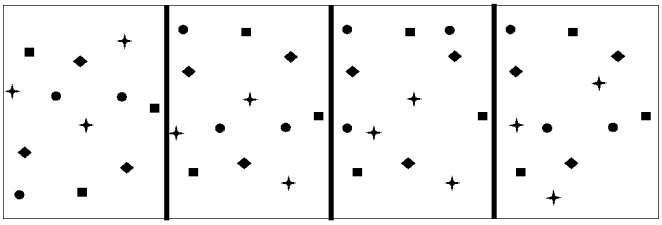


Рис. 1. Декомпозиция области. Область решения разбита вдоль координаты Y на 4 подобласти, частицы каждой подобласти равномерно распределены между четырьмя процессорами независимо от координаты. Различные символы, обозначающие частицы: круг, квадрат, ромб, звезда означают принадлежность частиц к разным процессорам.

Каждый из процессоров группы решает уравнения Максвелла во всей подобласти. Далее решаются уравения движения модельных частиц. После этого происходит суммирование

значений тока по всей подобласти. Один из процессоров группы производит обмен граничными значениями тока и полей с соседними подобластями, и затем рассылает полученные граничные значения всем процессорам своей группы. В случае, если и уравнения Максвелла для подобласти, и уравнения движения всех частиц подобласти частиц решаются на одном процессоре, вычисления с частицами занимают в 10-20 раз больше времени.

5. ЭФФЕКТИВНОСТЬ РАСПАРАЛЛЕЛИВАНИЯ

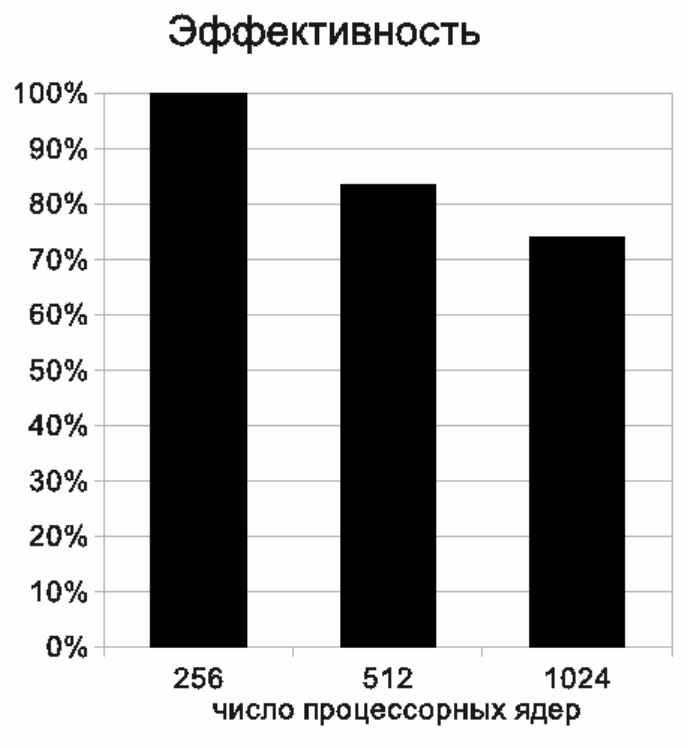


Рис. 2. Эффективность распараллеливания, вычисляемая по формуле 2 для небольшого числа процессорных ядер. Расчеты проведены на МВС-100К, МСЦ РАН.

Основная цель создания параллельной программы для моделирования аномальной теплопроводности в плазме - возможность счета на больших сетках и с большим количеством модельных частиц. Поэтому эффективность распараллеливания вычислялась следующим образом

$$k = \frac{T_2}{T_1} \times \frac{N_1}{N_2} \times \frac{S_2}{S_1} \times 100\% \qquad (2)$$

здесь T_1 - время счета с использованием N_1 процессоров, T_2 - время счета с использованием N_2 процессоров, S_1, S_2 - характерные размеры задач, в данном случае число узлов сетки по координате X. При этом размер задачи увеличивается пропорционально числу процессоров, т.е. нагрузка на отдельный процессор не возрастает. Цель такого определения эффективности – понять, насколько увеличивается время счета при увеличении числа процессоров и неизменной нагрузке на один процессор. В идеале время должно остаться тем же самым ($k = 100\%$ в идеале).

За основу для сравнения при этом берется модель, требующая большого времени вычислений (сетка $256 \times 128 \times 128$ узлов, 150 частиц в ячейке). В расчетах по определению эффективности увеличивается только размер сетки по X, все остальные параметры остаются неизменными. Вопрос о том, насколько быстрее можно посчитать на суперкомпьютере

задачу, которую можно посчитать и на настольной машине, в данной работе не рассматривается: такие задачи не представляют интереса с физической точки зрения. На рисунке 2 представлена эффективность для сравнительно небольшого числа процессорных ядер, на рисунке 3 приведена эффективность для большего числа ядер (> 1000). Из этих рисунков ожно сделать два вывода: во-первых, падение производительности при переходе с 1024 ядер на 2048 меньше, чем при переходе с 256 на 1024, во-вторых, созданная программа способна эффективно использовать более тысячи процессорных ядер.

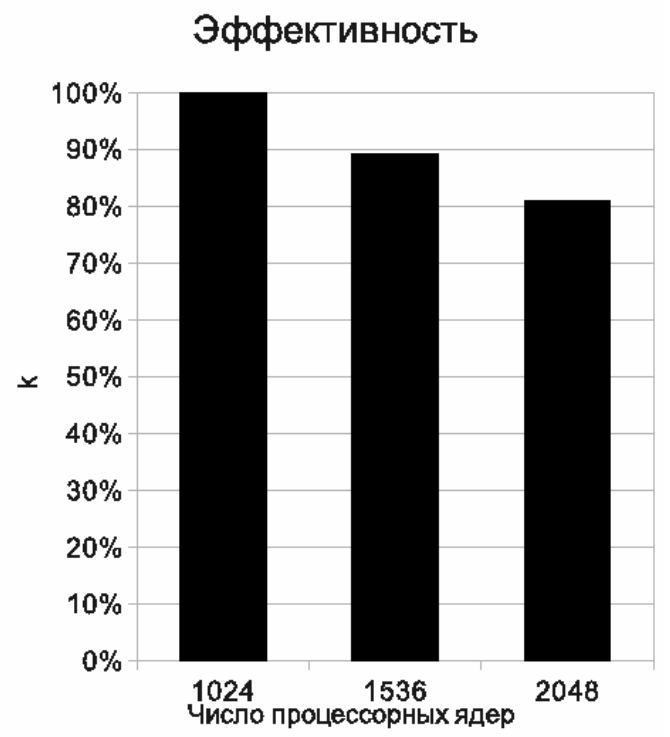


Рис. 3. Эффективность распараллеливания, вычисляемая по формуле 2 для большого числа процессорных ядер. Расчеты проведены на МВС-100К, МСЦ РАН.

6. СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ КЛАСТЕРОВ

№ по списку Top50	Название	Узлы	Сеть	Пиковая производительность, TeraFlOp/S
1	МВС-100К	4xXeon E5450, 3 GHz 8.192 GB RAM	Infiniband 4x DDR/ 2xGigabit Ethernet/ Gigabit Ethernet	95.04
2	СКИФ-МГУ	2xXeon E5472, 3 GHz 8.192 GB RAM	InfiniBand/ Gigabit Ethernet/ СКИФ-ServNet + IPMI	60
14	СКИФ-Cyberia	2xXeon 5150, 2.667 GHz, 4.096 GB RAM	QLogic InfiniPath/ Gigabit Ethernet/ СКИФ-ServNet	12.002
20	Кластер НГУ	2xXeon 5355, 2.66 GHz	Infiniband 4x DDR/ Gigabit Ethernet/ Gigabit Ethernet	5.4

Таблица 1. Основные характеристики кластеров, на которых производились расчеты, по материалам Российской суперкомпьютерной конференции «Научный сервис в сети Интернет: Масштабируемость, параллельность, эффективность», сентябрь 2009, www.supercomputers.ru

Каждый временной шаг работы программы состоит из следующих действий:

- Расчет электрического и магнитного поля;
- Расчет движения модельных частиц;
- Вычисление новых значений плотности тока и заряда.

Дополнительно, на отдельных временных шагах (как правило, каждый сотый шаг) проводится выдача диагностической информации, важнейшей частью которой являются фурье-образы основных величин (плотность заряда и тока, модуль электрического и магнитного поля).

Размер сетки во всех расчетах этого раздела $512 \times 64 \times 64$, 150 модельных частиц в ячейке, число использованных процессорных ядер – 160. Расчетная область поделена на 32 части вдоль координаты Y, далее частицы каждой подобласти дополнительно поделены между 8 ядрами.

Время работы всех этих процедур было измерено с помощью профилировщика gprof. В каждом случае приводится время работы одного вызова процедуры, то есть общее время,

затраченное на эту процедуру, поделенное на число вызовов.

6.1. РАСЧЕТ ДВИЖЕНИЯ МОДЕЛЬНЫХ ЧАСТИЦ И СКОРОСТЬ РАБОТЫ ОПЕРАТИВНОЙ ПАМЯТИ

Для вычисления новых значений координаты и импульса модельной частицы используются значения электрического и магнитного поля. Каждая компонента поля хранится в отдельном трехмерном массиве. Таким образом на каждом временном шаге для каждой модельной частицы происходит обращение к шести трехмерным массивам. Модельные частицы расположены внутри расчетной области случайным образом. Если даже модельные частицы расположены рядом в массиве, где хранятся их координаты, то сами значения координат будут близкими только вначале. В дальнейшем модельные частицы перемешиваются.

Это означает, что обращения к трехмерным массивам, содержащим электрическое и магнитное поля, являются неупорядоченными, и использование кэш-памяти в данном случае не позволяет сократить время счета. Таким образом, время расчета движения модельных частиц в основном определяется именно быстродействием оперативной памяти (пропускной способностью шины памяти). На рисунке 4 показано время, затраченное на вычисление движения частиц на одном временном шаге.

Вывод о том, что время обращения к оперативной памяти является определяющим при счете с частицами, подтверждается сравнением времен, полученных для МВС-100К и СКИФ-МГУ. Процессоры, установленные на этих кластерах, являются близкими по своим характеристикам, в то время как разница по времени счета почти в два раза.

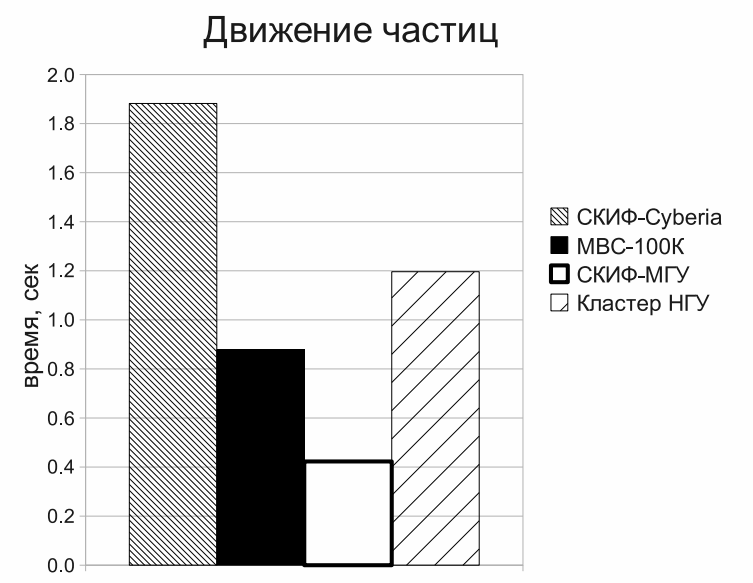


Рис. 4. Время работы процедуры интегрирования уравнений движения модельных частиц на различных кластерных системах. Расчеты проведены на СКИФ Cyberia (ТГУ, Томск), МВС-100К (МСЦ РАН, Москва), СКИФ-МГУ (МГУ, Москва) и на кластере НГУ (НГУ, Новосибирск).

Из рисунка 4 также видно, что существуют большие возможности для оптимизации, например, путем сортировки частиц по координате, что позволит упорядочить обращения к памяти и использовать кэш. Так как расчет движения модельных частиц занимает большую часть времени (от 92 % на СКИФ Cyberia до 64 % на СКИФ-МГУ) и это именно та часть программы, которая лучше всего распараллеливается, оптимизация этой процедуры

может привести к ухудшению ускорения и параллельной эффективности, однако уменьшение общего времени счета представляется более важной задачей.

6.2. ОДНОМЕРНОЕ ПРЕОБРАЗОВАНИЯ ФУРЬЕ И СКОРОСТЬ СЧЕТА

Для того, чтобы отделить время счета от времени обращения к оперативной памяти, было рассмотрено время работы процедуры, реализующей одномерное преобразование Фурье, (рисунок 5). Это процедура fftc из библиотеки NAG, она принимает на вход одномерный комплексный массив размером 512 или 64, и вычисляет быстрое преобразование Фурье. Все локальные переменные этой процедуры полностью помещаются в кэш, поэтому на примере этой процедуры можно судить о том, каким может быть время «быстрого» счета, то есть с использованием только кэша, без обращений к оперативной памяти.

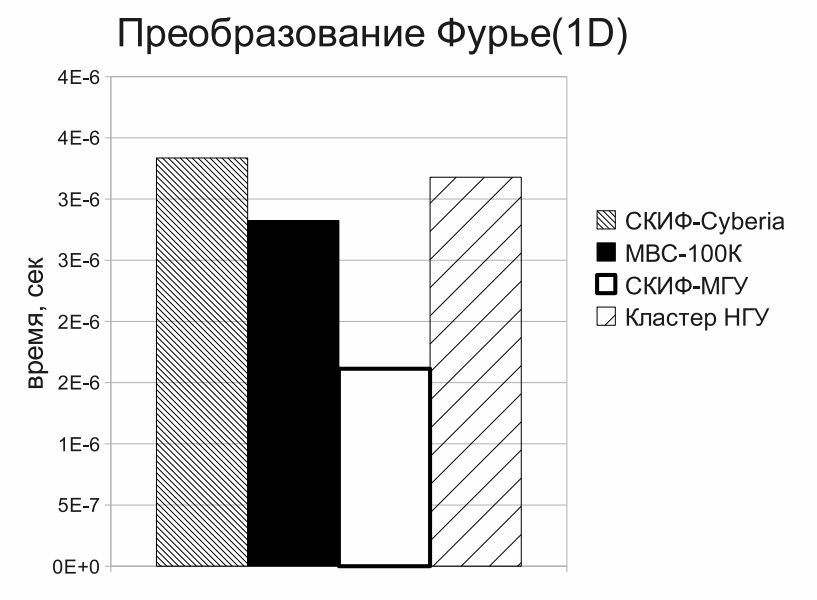


Рис. 5. Время работы процедуры, реализующей одномерное преобразование фурье на различных кластерных системах. Расчеты проведены на СКИФ Cyberia (ТГУ, Томск), МВС-100К (МСЦ РАН, Москва), СКИФ-МГУ (МГУ, Москва) и на кластере НГУ (НГУ, Новосибирск).

6.3. ПЕРЕСЫЛКА МОДЕЛЬНЫХ ЧАСТИЦ И СКОРОСТЬ РАБОТЫ МЕЖПРОЦЕССОРНЫХ КОММУНИКАЦИЙ

Скорость работы межпроцессорных коммуникаций была измерена с помощью процедуры пересылки частиц между процессорами. Эта процедура включает в себя определение модельных частиц, вылетевших за пределы подобласти, принадлежащей данному процессору, и находящихся в буферном пограничном слое, перемещение этих частиц в буфер для пересылки, и собственно пересылка, в зависимостии от номера процессора: если четный номер, то вначале «левому» процессору, то есть тому, чья подобласть распололожена левее по координатной оси, а затем «правому». Если номер процессора нечетный, то наоборот: вначале «правому», потом «левому». Вместе с отправкой частиц на другой процессор происходит прием частиц, перелетевших в подобласть, принадлежащую данному процессору. Дополнительно перед пересылкой собственно частиц соседние процессоры обмениваются количеством частиц, которые необходимо переслать.

Таким образом, время, показанное на рисунке 6 включает в себя просмотр списка частиц и четыре пересылки типа MPI_SendRecv. Количество перелетающих частиц не может быть большим из физических соображений (то есть, если возникают большие потоки частиц с процессора на процессор, это означает, что расчет физически некорректный), поэтому размер буфера для пересылки жестко задан - 5% массива частиц (на практике обычно еще меньше). Это означает, что объем пересылаемых данных порядка 10 Мб в каждой пересылке.

На рисунке 6 видно, что наименьшее время на пересылки тратится на СКИФ Cyberia и СКИФ-МГУ. Возможно, причиной этого является технология ServNet, используемая на машинах семейства СКИФ. Наибольшее время на пересылки израсходовано на МВС-100К, что можно объяснить большими масштабами и разнородностью этого вычислительного комплекса.

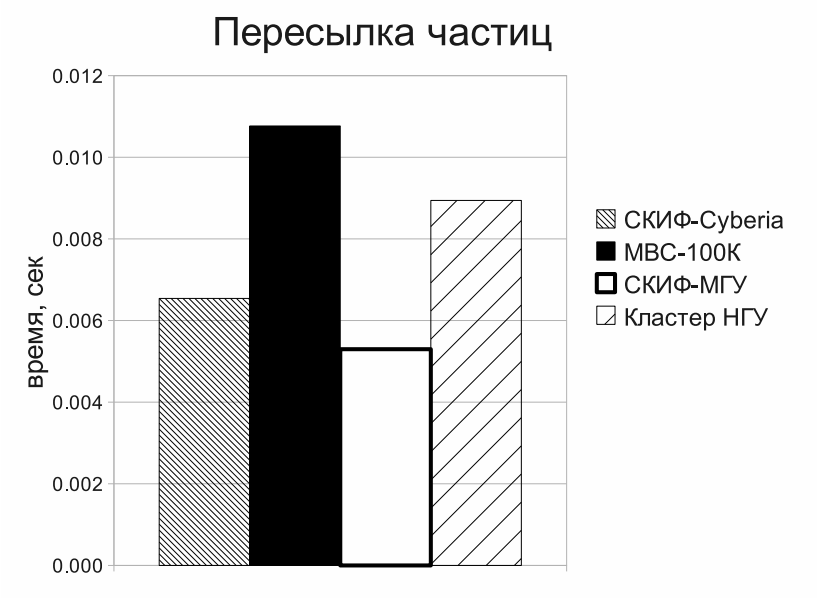


Рис. 6. Время работы процедуры пересылки модельных частиц на различных кластерных системах. Расчеты проведены на СКИФ Cyberia (ТГУ, Томск), МВС-100К (МСЦ РАН, Москва), СКИФ-МГУ (МГУ, Москва) и на кластере НГУ (НГУ, Новосибирск).

6.4. ВЫДАЧА ФИЗИЧЕСКОЙ ДИАГНОСТИКИ

Основной причиной падения эффективности распараллеливания в данной задаче является выдача физической диагностики: фурье-образы распределений поля, заряда, тока, скоростей и импульсов электронов и ионов, временные зависимости температуры, энергии частиц и полей, аномальный и классический коэффициент теплопроводности. Большое количество диагностики, а также однопроцессорный характер ее выдачи (сбор по всем процессорам, и затем запись на диск одним, выделенным процессором) являются причиной того, почему именно за счет диагностик происходит потеря производительности программы.

С другой стороны, их нельзя исключить из рассмотрения, и анализировать скорость работы только «вычислительной» части программы: формальный результат по ускорению и эффективности, безусловно, был бы лучше, но программа написана для получения физического результата, а, значит, для диагностики. Кроме того, целью настоящей работы является оценка фактической производительности кластеров именно с точки зрения конечного пользователя – формальные показатели их производительности хорошо известны.

Для примера была выбрана выдача температуры электронов и ионов. Эта диагностика выдается на каждом временном шаге. Для вычисления температуры необходимо вычислить дисперсию импульсов частиц отдельно на каждом процессоре, сложить частичные суммы, и затем 0-й процессор выдает вычисленные значения в файл. Таким образом показанное на рисунке 7 время выдачи температуры включает в себя расчет, пересылку и выдачу в файл. Так же , как и в случае с пересылкой частиц, лидируют представители семейства СКИФ. Отсюда можно сделать вывод, что основное время при выдаче диагностики расходуется именно на пересылку, и оптимизацию диагностики нужно будет начинать именно с этого.

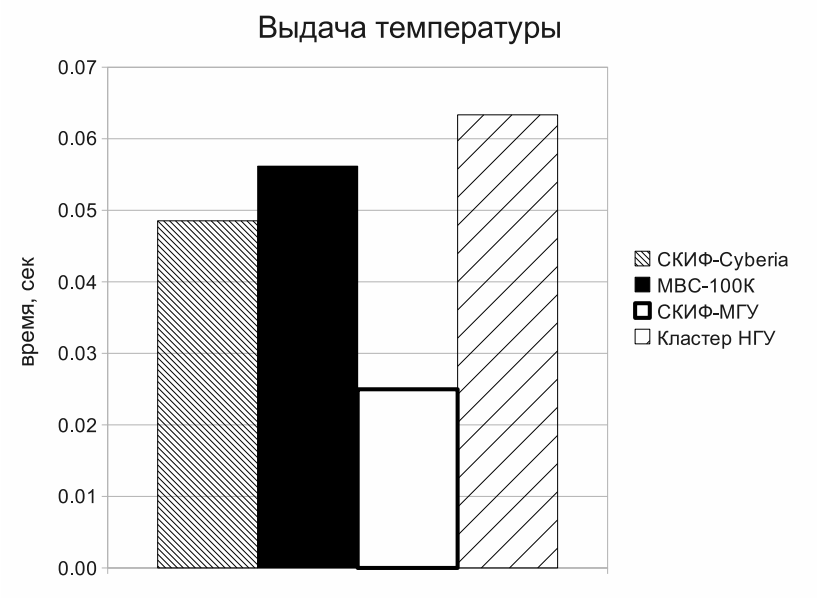


Рис. 7. Время работы процедуры, выполняющей расчет температуры электронов и ионов на различных кластерных системах. Расчеты проведены на СКИФ Cyberia (ТГУ, Томск), МВС-100К (МСЦ РАН, Москва), СКИФ-МГУ (МГУ, Москва) и на кластере НГУ (НГУ, Новосибирск).

Вопрос об оптимизации диагностических сообщений является очень актуальным, но его решение возможно только в том случае, когда будет точно известно, какую информацию и в каком объеме нужно выдавать – а такая ситуация равнозначна решению физической задачи. В данный момент диагностики различных видов убираются и добавляются после

каждого вычислительного эксперимента.

Для того, чтобы понять, насколько сильно выдача диагностики нуждается в оптимизации, приведем конкретные цифры: на СКИФ Cyberia расчет температуры занимает 2.39 % всего времени работы программы, на МВС-100К 4.17 %, на СКИФ-МГУ 3.94 % и на кластере НГУ 4.52 %. Это самая времяемкая процедура диагностики, все остальные потребляют на порядок меньше времени. Соотношение времени работы диагностики и времени счета следующее: расчет движения частиц занимает намного больше времени (от 64 % до 92 %), расчет поля – намного меньше (от 1.6 % на МВС-100К до менее чем 0.1% на СКИФ Cyberia).

7. АНОМАЛЬНАЯ ТЕПЛОПРОВОДНОСТЬ В ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТАХ

Актуальность настоящей работы связана с тем, что в экспериментах на многопробочной магнитной ловушке ГОЛ-3 (ИЯФ СО РАН) наблюдается понижение электронной теплопроводности на 2-3 порядка по сравнению с классическим значением. Известно множество работ по моделированию теплопроводности в термоядерных установках. Вычисление температуры в этих работах производится в основном с помощью гидродинамических уравнений. Таким образом функция распределения электронов по энергиям предполагается максвелловской, что может не соответствовать действительности. Для вычисления неравновесных распределений применяют, в частности, бессеточные модификации метода частиц. Однако для метода частиц в ячейках не известны критерии, позволяющие понять, насколько правильным является полученное в расчетах распределение температуры, и таким образом, корректность моделирования теплопроводности также оказывается под вопросом.

В результате проведения вычислительных экспериментов выяснилось, что движение электронов пучка, первоначально равномерное и однонаправленное, после релаксации пучка становится вихревым. Это приводит к тому, что поток энергии электронов

$$q = \left| \int dxdy \frac{mv_e^2}{2} \vec{v_e} \right|$$

также приобретает сложную структуру с модуляциями по осям X и Y, и более того, возникают подобласти, где поток энергии близок к нулю (рис. 8). На рисунке 8 линия уровня с наименьшим значением (0.01) соответствует 1 % начальной величины потока энергии электронов. Из рисунка видно, что в значительной части расчетной области поток энергии меньше начального. Более подробный анализ показывает, что в отдельных частях области поток энергии электронов на несколько порядков меньше начального.

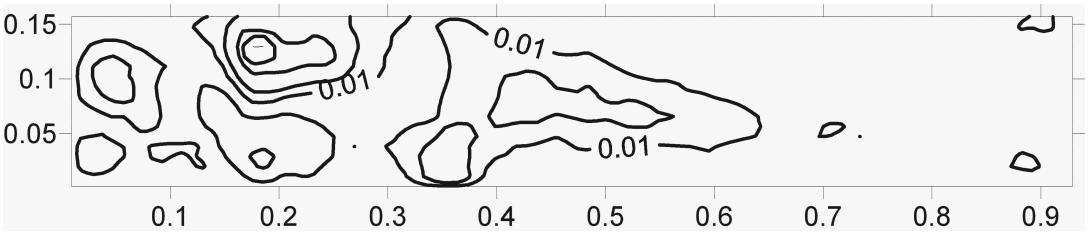


Рис. 8. Линии уровня потока энергии электронов в плоскости XY, $z = Z_M/2$, момент времени $t = 91.7$ (в единицах плазменного периода). Величина потока нормирована на его значение в момент времени $t = 0$.

8. ЗАКЛЮЧЕНИЕ

Описана параллельная реализация модели взаимодействия электронного пучка с плазмой, приведены показатели ускорения и эффективности. Измерено и проанализировано время работы отдельных частей программы на различных кластерах. В результате выяснилось, что важнейшим показателем быстродействия кластера с точки зрения программы, реализующей метод частиц в ячейках является скорость доступа к оперативной памяти, а основным сдерживающим фактором для эффективности распараллеливания является выдача физических диагностик. В отношении самой программы показано, что существуют большие возможности для ускорения счета.

Для того, чтобы не возникало впечатления, что мощные и дорогие кластера используются только для тестирования, автор считает своим долгом уточнить, где и как проводились физические расчеты. Программа была создана и отлажена как многопроцессорная ($>$ 100 ядер) на СКИФ Cyberia, там же проведены расчеты по влиянию энергии пучка на температуру плазмы. Основная масса расчетов по определению характеристик возникающей неустойчивости проведена на кластере НГУ и НКС-30Т (ИВМиМГ СО РАН). Расчеты по определению зависимости аномальной теплопроводности от физических параметров проведены на МВС-100К и СКИФ-МГУ.

Необходимо подчеркнуть, что при меньшем размере сетки и меньшем количестве модельных частиц в расчетах не происходит резонансного взаимодействия пучка с плазмой, и наблюдаемый в экспериментах эффект аномальной теплопроводности воспроизведен быть не может, поэтому использование суперЭВМ принципиально важно для решения данной задачи.

Список литературы

1. **Астрелин В.Т., Бурдаков А.В., Поступаев В.В.** Подавление теплопроводности и генерация ионно-звуковых волн при нагреве плазмы электронным пучком. //Физика плазмы, 1998, том 24, № 5, с. 45-462.

2. **Вшивков В.А., Вшивков К.В., Дудникова Г.И.** Алгоритмы решения задачи взаимодействия лазерного импульса с плазмой.//Вычислительные технологии, Том 6, № 2, 2001.

3. **Н.Кролл, А,Трайвелпис** Основы физики плазмы, Москва: «Мир», 1975.

4. **Григорьев Ю.Н., Вшивков В.А.** Численные методы «частицы-в-ячейках», Новосибирск: «Наука», 2000.

5. **Ч. Бедсел, Б.Лэнгдон** Физика плазмы и математическое моделирование, Москва: «Мир», 1989.

О восстановлении программ из контрольной точки*

А.Ю. Поляков

В работе описаны два подхода к проблеме восстановления распределенных программ из контрольной точки. Предложен алгоритм восстановления взаимосвязей типа "родитель-потомок" и алгоритм принадлежности к группам и сеансам для набора процессов в рамках элементарной машины распределенной вычислительной системы. Предложен алгоритм координированного восстановления набора связанных процессов, перезапускаемых раздельно (на различных элементарных машинах или терминалах). Описанные подходы реализованы в системе создания контрольных точек DMTCP (Distributed MultiThreaded CheckPointing).

1. Введение

Распределенные вычислительные системы (ВС) - это важнейший вычислительный инструмент, который используется для проведения научных, инженерных и экономических расчетов [1]. Такие ВС являются большемасштабными, они состоят из сотен тысяч процессорных ядер и имеют производительности порядка PetaFLOPS [2]. Однако даже на таких высокопроизводительных системах многие современные задачи требуют для своего решения дни, недели и месяцы. Несмотря на высокий уровень развития элементной базы и схемотехники, аппаратные ресурсы распределенных ВС не являются абсолютно надежными. В связи с их большемасштабностью вероятность выхода из строя одной или нескольких составляющих становится достаточно высокой. Отказы процессоров, жестких дисков, сетевых адаптеров, кабелей и шин передачи данных могут повлечь за собой потерю значительного количества промежуточных вычислений, что приведет к снижению технико-экономической эффективности ВС. Таким образом, актуальной задачей является обеспечение отказоустойчивого выполнения программ на распределенных ВС.

Наиболее распространенным подходом к решению данной проблемы является создание контрольных точек (КТ) [3]. В процессе выполнения программы происходит периодическое сохранение ее состояния на надежный носитель данных. В случае отказа производится "откат" к ближайшей доступной контрольной точке, и работа возобновляется. При этом теряется незначительное количество промежуточных вычислений.

В данной работе описаны два подхода к проблеме восстановления распределенных программ из контрольной точки. Предложен алгоритм координированного восстановления набора связанных процессов, перезапускаемых раздельно (на различных элементарных машинах или терминалах). Разработан алгоритм восстановления взаимосвязей типа "родитель-потомок" и алгоритм принадлежности к группам и сеансам для набора процессов в рамках элементарной машины (ЭМ) распределенной ВС. Данные алгоритмы реализованы в программном пакете создания КТ DMTCP (Distributed MultiThreaded Check-Pointing) [4], который позволяет создавать КТ для последовательных, параллельных и распределенных программ в ОС GNU\Linux.

2. Классификация средств создания контрольных точек

Существует достаточно много средств создания КТ (ССКТ) [4-7], каждое из них имеет свои преимущества и недостатки. Рассмотрим несколько подходов к классификации ССКТ.

Существует две основные схемы взаимодействия ССКТ с защищаемой программой: *явная* и *прозрачная* (неявная). ССКТ, построенные на основе *явной схемы*, позволяют задать ограниченный набор информации, которую необходимо сохранить в КТ. Это позволяет снизить объем дискового ввода/вывода, т.е. значительно уменьшает накладные расходы таких ССКТ. Недостатком явной схемы является необходимость модификации исходного кода, что не позволяет

* Работа выполнена при поддержке РФФИ (гранты 08-07-00018, 08-07-00022, 08-08-00300, 09-07-00185, 09-07-12016, 09-07-13534, 09-07-90403) и Совета по грантам Президента РФ (грант НШ-2121.2008.9)

применять ее к программам, доступным только в бинарном виде. Кроме того, КТ могут создаваться только в моменты времени, определяемые программой и связанные с завершенностью определенного периода вычислений.

ССКТ, построенные на основе *прозрачной схемы*, выполняют сохранение КТ незаметно для программы, что обеспечивает простоту и универсальность их использования. Недостатком этой схемы является больший объем дискового ввода/вывода, так как сохраняется все пространство памяти.

По классам поддерживаемых программ ССКТ можно разделить на *сосредоточенные* и *распределенные*. Сосредоточенные ССКТ обеспечивают отказоустойчивость выполнения одного или нескольких процессов в рамках вычислительной машины. Распределенные ССКТ обычно строятся на базе сосредоточенных и позволяют выполнять создание КТ для распределенных и параллельных программ, что делает их важным инструментом организации функционирования ВС. Для создания распределенной КТ (РКТ) необходимо:

1) создать сосредоточенные КТ для всех процессов, входящих в состав распределенной программы (РП);
2) сохранить граф связей между процессами РП;
3) сохранить сообщения, которые были отправлены, но не доставлены на момент создания РКТ (такие сообщения также называют in-transit).

Для распределенных ССКТ различают *координированный* и *некоординированный* подходы. При создании РКТ каждый процесс РП сохраняет свое состояние в КТ. Целостной РКТ [3] называется набор из N локальных КТ, формирующих допустимое состояние программы. Такая РКТ может быть использована для восстановления программы после сбоя. При координированном подходе создание КТ происходит синхронно, что гарантирует целостность РКТ. При некоординированном подходе каждый процесс создает КТ независимо от других. Следовательно, при восстановлении необходимо выполнять поиск целостного состояния программы на основе набора независимых КТ. Это вносит дополнительные накладные расходы. Для некоординированного подхода существует опасность возникновения "эффекта домино", когда в процессе поиска целостного состояния происходит откат к начальному состоянию программы.

Распределенные ССКТ также можно разделить на *универсальные* и *MPI-ориентированные*. Первые позволяют создавать РКТ для любых распределенных и параллельных программ, в том числе для различных реализаций модели передачи сообщений (PVM, MPI). Что касается вторых, то существует несколько ССКТ, построенных на базе конкретных реализаций MPI. Например, OpenMPI [8], MVAPICH2 [9], LAM-MPI [10]. Все они используют ССКТ BLCR [5] для создания сосредоточенных КТ и реализуют собственные механизмы сохранения графа связей и транзитных сообщений.

Для создания КТ сосредоточенного процесса необходимо сохранить информацию о его состоянии. Это может быть реализовано на различных программных уровнях:

1. **Уровень операционной системы (ОС)**. Предусматривает сохранение содержимого пространства ядра и пространства пользователя для всех процессов ОС. Такой подход может быть реализован с использованием систем виртуализации (например, VMWare);
2. **Уровень ядра ОС**. Предусматривает внедрение дополнительных компонентов, позволяющих сохранить необходимую информацию: содержимое памяти конкретного процесса и состояние ядра, относящиеся к нему.
3. **Уровень системных библиотек.** Предусматривает сохранение содержимого памяти и состояния ядра с использованием средств, предоставляемых ОС для управления процессами.
4. **Прикладной уровень.** Предусматривает сохранение минимального объема информации, необходимого для восстановления каждой конкретной программы.

ССКТ уровней ОС, ядра и системных библиотек реализуются в рамках прозрачной схемы. Кроме того, некоторые ССКТ уровней ядра и системных библиотек предоставляют программе возможность влиять на процесс обеспечения отказоустойчивости, например, выбирать наиболее удобные моменты для создания КТ. Прикладной уровень предусматривает только явную схему.

Преимуществом первого уровня является простота реализации, а недостатком - значительный объем дискового ввода/вывода и отсутствие гибкости. Второй уровень позволяет получать прямой доступ к внутренним структурам ядра и памяти процесса и выполнять сохранение необходимой для восстановления информации при меньшем объеме ввода/вывода. Недостатком данного подхода является зависимость от изменений в ядре ОС (новые версии ядра Linux выходят в среднем с частотой раз в 3-4 месяца). Также данный подход требует привилегий суперпользователя для установки и управления, а ошибки, допущенные в программном обеспечении уровня ядра, приводят к нарушению работы всей ОС.

Третий уровень позволяет обеспечить создание КТ, не требуя при этом привилегий суперпользователя и не подвергая угрозе функционирование всей ОС. Однако при данном подходе невозможно осуществить прямой доступ к внутренним структурам ядра, которые описывают защищаемый процесс. Для этого требуется перехват и обработка системных вызовов.

На четвертом уровне сохраняется лишь содержимое буферов, которые явно указываются в программе.

3. Distributed MultiThreaded Checkpointing - DMTCP

Программный пакет DMTCP реализован на уровне системных библиотек и является универсальной координированной распределенной ССКТ. DMTCP разработан в Северо-западном университете (Northeastern University) США под руководством профессора Дж. Купермана.

Наиболее распространенной сосредоточенной ССКТ на данный момент является пакет BLCR. Кроме того, как было отмечено ранее, он используется во многих распределенных MPI-ориентированных ССКТ. Таблица 1 отражает сравнение ССКТ DMTCP и BLCR по поддерживаемым функциям ОС. BLCR используется для создания сосредоточенных КТ в нескольких MPI-ориентированных распределенных ССКТ.

Таблица 1. Функции, поддерживаемые ССКТ

Поддерживаемые компоненты ОС	DMTCP		BLCR	
	Полностью	Частично	Полностью	Частично
Обработка сигналов	X		X	
Сокеты	X		-	-
Многопоточные приложения	X		X	
Идентификаторы ресурсов ОС (процессы, группы, сессии)		**X**	**X**	
Именованные и неименованные каналы	X		X	
Открытые файлы	X		X	
Отображенные (mapped) файлы	X		X	
/proc файлы		X		X
Статически скомпилированные программы	**-**	**-**		**X**
Отлаживаемые программы		X	-	-

Из таблицы 1 видно, что DMTCP уступает BLCR по двум параметрам. Во-первых, нет поддержки статически скомпилированных программ, т.к. для перехвата системных вызовов используется "предзагрузка" служебной динамической библиотеки dmtcphijack.so. Однако данный пункт не полностью поддерживается и в BLCR. Во-вторых, отсутствует восстановление идентификаторов ресурсов ОС, таких как идентификаторы групп и сессий. В пространстве ядра в связи с прямым доступом к его внутренним структурам данная задача является более простой. В DMTCP (на уровне системных библиотек) была реализована частичная виртуализация идентификаторов процессов. В данной работе предложен алгоритм, позволяющий более полно восстанавливать идентификационную информацию. Он был интегрирован и используется в DMTCP в настоящее время.

На рисунке 1 показан запуск программы с применением DMTCP. В процессе ее работы автоматически осуществляется контроль над созданием новых процессов с использованием сис-

темного вызова *fork()*. Как было сказано ранее, DMTCP реализует координированное создание КТ. На каждую вычислительную группу создается один координатор (*dmtcp_coordinator*). Он может быть запущен явно, как показано на рисунке 2. Если при запуске программы (рисунок 1) процесс координатор не обнаружен, то он будет запущен автоматически.

```
host1$ dmtcp_checkpoint ./program1
```

Рис. 1. Запуск программы *program1* под управлением DMTCP на узле *host1*

```
host1$ dmtcp_coordinator
```

Рис. 2. Запуск процесса-координатора на узле *host1*

```
host1$ dmtcp_checkpoint ./program2
```

Рис. 3. Запуск программы *program2* под управлением DMTCP на узле *host1*

```
host2$ DMTCP_HOST="host1" dmtcp_checkpoint ./program3
```

Рис. 4. Запуск программы *program3* под управлением DMTCP на узле *host2*

Возможно создание РКТ для нескольких взаимодействующих программ, запускаемых с разных терминалов. Например, как показано на рисунках 1 и 3. В этом случае вспомогательный модуль DMTCP, интегрированный в каждую из программ, выполнит соединение с координатором.

Также возможно создание РКТ для процессов, работающих на разных узлах сети. Для этого необходимо указать через переменную окружения *DMTCP_HOST* адрес узла, на котором выполняется координатор. Так, на рисунке 4 показан запуск программы *program3*, которая подключается к вычислительному процессу, уже содержащему программы *program1* и *program2*.

Создание РКТ происходит следующим образом: каждый процесс сохраняет свое состояние в отдельном файле, а координатор формирует shell-скрипт, содержащий последовательность действий, необходимых для запуска вычислений из данной РКТ.

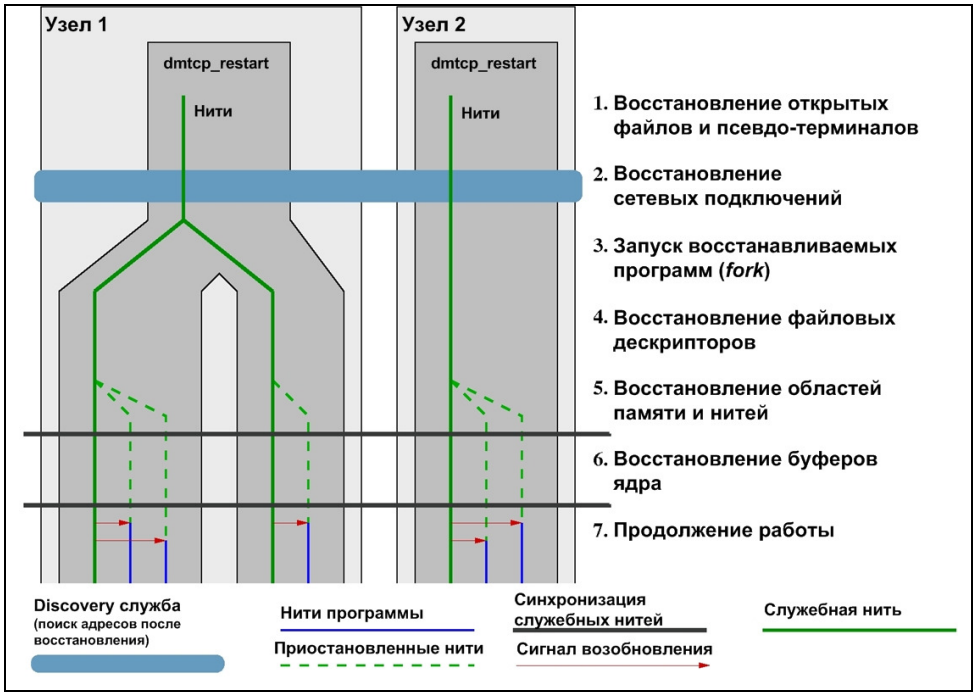


Рис. 5. Восстановление вычислений из РКТ DMTCP

Как показано на рисунке 5, на этапе восстановления на каждом узле запускается один служебный процесс, использующий РКТ для воссоздания компонентов программы (этап 3). Для синхронизации узлов используется этап восстановления сокетов (этап 2).

Недостаток данной схемы заключается в том, что процессы, выполнявшиеся на разных терминалах, будут перезапущены уже на одном. Например, DMTCP используется в качестве

основы для универсального реверсивного отладчика URDB [11]. Типичным сценарием применения URDB является подключение (attach) к уже выполняющейся программе и ее отладка. При восстановлении такой отладочной сессии необходимо сохранить принадлежность к разным терминалам, однако отсутствие средств синхронизации не позволяет этого сделать.

Рассмотрим другой пример: восстановление из контрольной точки группы процессов, распределенных по разным узлам сети. Процессы разбиты на подгруппы, не связанные между собой постоянными сетевыми соединениями. В этом случае барьер, образованный этапом 2 (Recreate and reconnect sockets), не является достаточным для синхронизации. Если одна подгруппа была запущена значительно раньше остальных, ее выполнение будет продолжено, а остальные подгруппы не будут иметь возможности возобновить работу.

Для устранения указанных недостатков был предложен дополнительный компонент схемы синхронизации, который представлен в данной работе.

4. Дополнительные компоненты схемы синхронизации

В DMTCP предусмотрен барьер, позволяющий синхронизировать восстановление из РКТ только для процессов, связанных постоянными сетевыми соединениями. Как было показано в разделе 3, существуют программы, для которых это условие не выполняется. Для устранения этого недостатка возникла необходимость расширения схемы синхронизации.

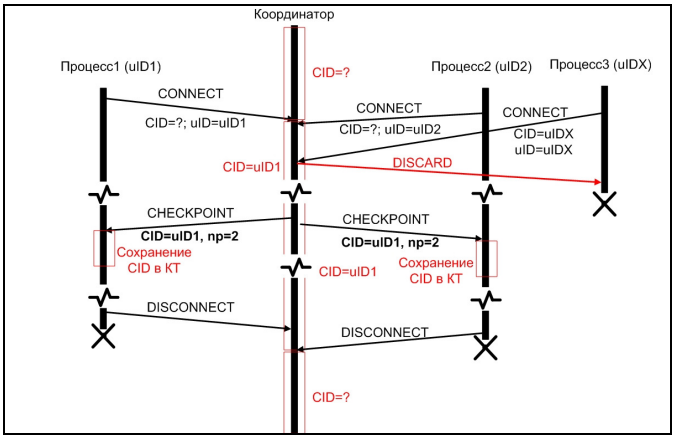


Рис. 6. Создание РКТ

Каждый процесс в DMTCP имеет уникальный идентификатор uID (*unique ID*), который формируется из трех компонент: <*хеш-код имени сетевого узла*>-<*PID*>-<*временная метка*>.

Координатор играет роль службы, предоставляющей сервис синхронизации. Его состояние подстраивается под выполняемые задачи и не сохраняется в РКТ. В качестве синхронизационного условия выбрано число процессов, принадлежащих вычислительной группе (ВГ) на момент создания контрольной точки. Как показано на рисунке 6, для идентификации ВГ (*CID – computational group ID*) используется uID процесса, который выполнил подключение первым. Если приходит запрос на подключение от другой ВГ (процесс 3 на рисунке 6), то оно отклоняется. На этапе создания РКТ координатор рассылает *CID* текущей ВГ и число ее участников (*np – number of process*). Эта информация сохраняется в каждой локальной КТ. При отключении последнего процесса из текущей ВГ координатор переходит в состояние *CID=?* и готов принимать новые запросы на услуги синхронизации от других ВГ.

На этапе восстановления (рисунок 7) процесс считывает *CID* и *np* из КТ и отправляет координатору при подключении. Если координатор не занят обслуживанием других заявок, он устанавливает параметр *CID* в значение, которое содержится в сообщений. Также запоминается количество клиентов, которое должно выполнить подключение до того, как можно будет продолжить вычислительных процесс.

Если подключение выполняет клиент, не имеющий *CID* или имеющий *CID*, который отличается от текущего, то такое соединение отклоняется.

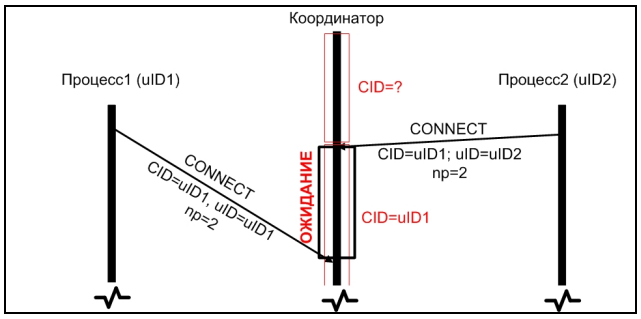


Рис. 7. Восстановление из РКТ

5. Алгоритм восстановления идентификационной информации

Как было сказано ранее, восстановление идентификационной информации на уровне системных библиотек затруднено отсутствием прямого доступа к внутренним структурам ядра ОС GNU\Linux. Необходима имитация процесса первоначального запуска программы. Рассмотрим подробнее восстанавливаемые идентификационные ресурсы.

5.1 Идентификационные ресурсы

В ОС GNU\Linux процесс описывается набором идентификаторов. Первый из них – идентификатор процесса PID (*process ID*). *PID* назначается при создании системными вызовами *fork()* или *vfork()* и используется для того, чтобы указать на процесс в ряде важных системных вызовов, таких как *kill()*, *ptrace()*, *setpriority()*, *waitpid()*.

Отношение родитель-потомок строится на основе *PID*. Процесс, выполнивший системный вызов *fork*, становится родителем созданного процесса. Для доступа к информации об идентификаторе родителя (*parent PID - PPID*) используется системный вызов *getppid*. Если процесс завершается, а потомки продолжают существование, их родителем становится системный процесс *init*, имеющий *PID=1*.

Каждый процесс принадлежит к одной и только одной сессии, для создания новой используется системный вызов *setsid*. Идентификатор сессии SID (*session ID*) равен идентификатору процесса-создателя (или лидера). Принадлежность к сессии наследуется потомком от родителя.

Каждый процесс принадлежит к одной и только одной группе. Если его идентификатор совпадает с идентификатором группы, то он называется ее лидером. Все процессы группы принадлежат одной и только одной сессии. Данные механизмы используются командными интерпретаторами при организации конвейеров, некоторыми отладчиками для управления отлаживаемыми программами и т.д.

5.2 Постановка задачи

Пусть имеется множество контрольных точек $C = \{c_i\}, i = 1...N$, каждая из которых однозначно соответствует восстанавливаемому процессу $p_i \in P$. КТ описывается четырьмя параметрами $c_i = (pid_i, ppid_i, sid_i, img_i)$, где: pid_i - уникальный идентификатор ($\forall i, j = 1...N, i \neq j, pid_i \neq pid_i$) в рамках ЭМ ВС; $ppid_i$ - идентификатор процесса, создавшего p_i через системный вызов *fork()*; sid_i - идентификатор сессии, если $pid_i = sid_i$, то КТ c_i содержит процесс-лидер сессии sid_i; img_i - сохраненное состояние процесса, необходимое для его перезапуска.

Обозначим через $s = \bigcup_{i=1}^{N} sid_i$ множество уникальных идентификаторов сеансов, к которым принадлежат процессы из P. Пусть $S = \{S_k\}, k = 1...|s|$ - множество сеансов, где

$S_k = \{c_i \mid c_i \in C, sid_i = s_k\}$ - подмножество КТ, содержащих процессы одного сеанса. Очевидно, что $C = \bigcup_{S_k \in S} S_k$ и $\forall k_1, k_2 = 1...|s|,\ k_1 \neq k_2,\ S_{k_1} \cap S_{k_2} = \varnothing$. Пусть также определено множество $R = \{c_i \mid \forall j = 1...N, j \neq i, ppid_i \neq pid_j\}$ независимых КТ.

Требуется, используя программный интерфейс ОС GNU/Linux, выполнить запуск процессов из контрольных точек так, чтобы восстановить их исходную иерархию и принадлежность к сеансам. При этом для изменения сеанса имеется только системный вызов *setsid()*, который позволяет процессу создать собственный сеанс, в котором он становится лидером.

5.3 Алгоритм восстановления иерархии процессов и сеансов

На вход алгоритма подается множество контрольных точек C. Алгоритм состоит из следующих шагов:

1. **Формирование отношений типа родитель-потомок**. Строится лес деревьев $T = \{T_l\}, l = 1...|R|$ (рисунок 8) , для которого определена функция однозначного соответствия f между КТ и узлами деревьев леса: $f = \{(t,c) \mid \exists l = 1...|R|, t \in T_l, c \in C, t \Leftrightarrow c\}$. Справедливы следующие утверждения:

1) $\forall T_l \in T$, если t - корень T_l, то $f(t) \in R$;

2) $\forall T_l \in T, \forall t_1, t_2 \in T_l$ то t_1 - непосредственный потомок $t_2 \Leftrightarrow \exists i, j : c_i = f(t_1), c_j = f(t_2)$ и $pid_i = ppid_j$.

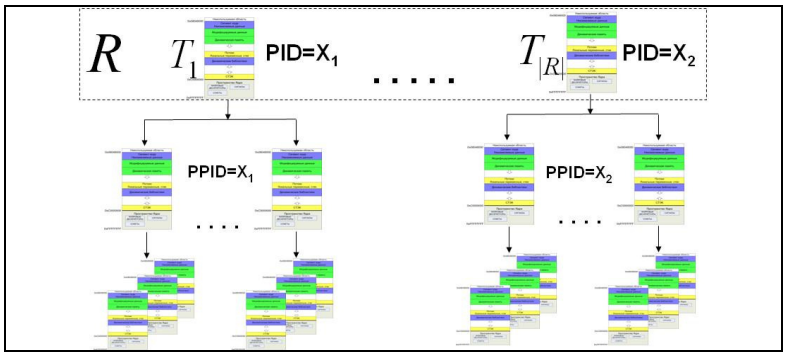


Рис. 8. Лес деревьев T, соответствующий восстанавливаемым КТ

2. **Построение метаинформации**. Для того чтобы восстанавливать принадлежность к одной сессии процессов, которым соответствуют узлы различных деревьев, выполняется построение метаинформации. Для каждого дерева $T_l \in T$ выполняется его обход в глубину. Для каждого обрабатываемого узла $t \in T_l$ определяется номер соответствующей ему КТ $i : c_i = f(t)$. Строится метаинформация, которая представляется в виде пары $(x^i{}_k, y^i{}_k)$, где $x^i{}_k \in x^i$, $x^i = \{x^i{}_k \mid x^i{}_k = sid_j,\ j = 1...N, f^{-1}(c_j) \in \{t\} \cup \{потомки\ t\}\}$,

$$y^i{}_k = \begin{cases} 1, & \exists c_j \in C : f^{-1}(c_j) \in \{t\} \cup \{потомки\ t\}\ и\ pid_j = x^i \\ 0, & иначе \end{cases}.$$

Второй компонент пары ($y^i{}_k$) указывает на наличие или отсутствие лидера сессии с идентификатором $x^i{}_k$.

На рисунке 9 показан пример сбора метаинформации. Рассмотрим узел X, которому соответствует метаинформация, состоящая из одной пары *(SID_3,0)*. X является листовым и не является лидером SID_3. Корень дерева (узел Y) содержит метаинформацию из четырех пар: *(SID_1,1)*, *(SID_2,1)*, *(SID_3,1)*, *(SID_4,1)*. Это означает, что на текущем и нижележащих уровнях дерева имеет-

ся четыре сессии с идентификаторами SID_1,SID_2,SID_3,SID_4, для каждой из них были найдены лидеры.

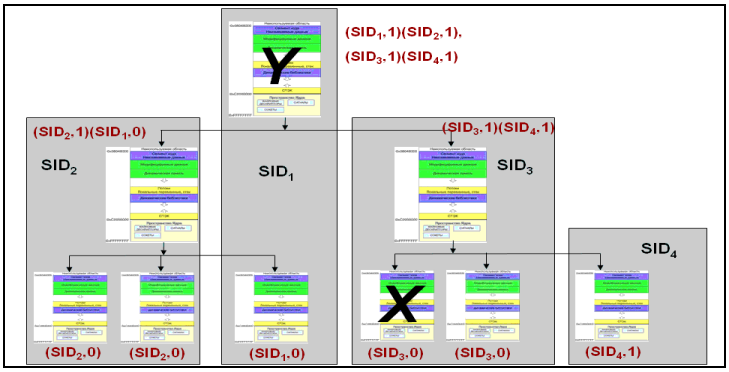


Рис. 9. Построение метаинформации

3. **Построение зависимостей между элементами** T. Каждому узлу $t \in T$ ставится в соответствие множество $Dep(t) = \varnothing$. Далее выполняется обработка метаинформации, соответствующей корням деревьев из леса T. Пусть $T_{M_1}, T_{M_2} \in T$, как показано на рисунке 10. Пусть t_1 - корень T_{M_1}, t_2 - корень T_{M_2}, i, j - соответствующие индексы КТ: $c_i = f(t_1)$, $c_j = f(t_2)$. Тогда T_{M_2} зависит от T_{M_1}, если $\exists k_1, k_2 : x^i{}_{k_1} = x^j{}_{k_2}$ & $y^j{}_{k_2} = 1$ & $y^i{}_{k_1} = 0$. В этом случае выполняется поиск лидера сессии $x^i{}_{k_1}$ - $\tilde{t} = f^{-1}(c_l) : pid_l = sid_l = x^i{}_{k_1}$ и модифицируется соответствующее множество $Dep(\tilde{t}) = Dep(\tilde{t}) \cup t_2$

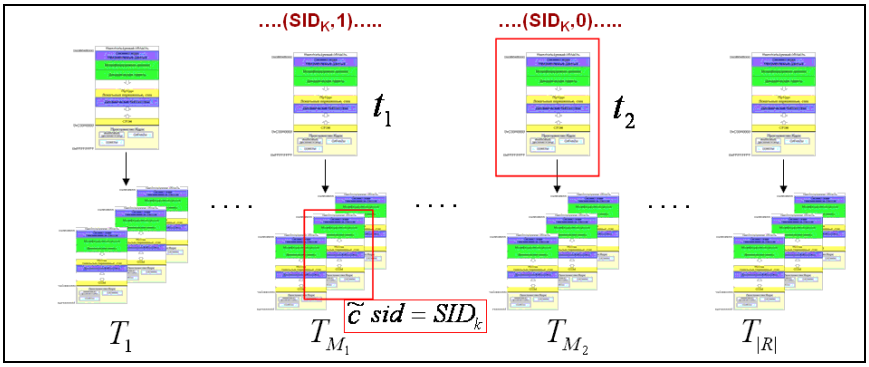


Рис. 10. Построение зависимостей между деревьями леса T

На рисунке 10 показано, что метаинформация корня дерева T_{M_2} указывает на отсутствие лидера сессии SID_k в T_{M_2}. Пара (SID_k,1), соответствующая корню дерева T_{M_1}, указывает на наличие узла $\tilde{t}$ и контрольной точки $\tilde{c} = f(\tilde{t})$, содержащей лидера SID_k. Тогда считаем, что дерево T_{M_2} зависит от узла $\tilde{t}$. То есть при запуске процессов из КТ сначала должна быть восстановлена КТ, соответствующая $\tilde{c}$, и создана новая сессия SID_k, а также все процессы дерева T_{M_2}.

4. Для всех независимых деревьев ($\forall T_l \in T : t_l = 0$) выполняется восстановление исходной структуры процессов с использованием *fork()* и *setsid()* по алгоритму, приведенному на рисунке 11.

```
procedure restore(t,psid)
    i = k : (c_k == f(t));
    if pid_i != sid_i then
        for t1 ← childs(t) do fork(); restore(f(t1), psid) done
        start(f(t));
    else
        for t1 ← childs(t) do
            j = k : (c_k == f(t1));
            if sid_j <> pid_i then
                if fork() = 0 then restore(f(t1),psid) end if
             end if
        done
        psid = setsid();
        // Восстановление всех деревьев, зависящих от узла t
        for t1 ← Dep(t) do
             if fork() = 0 then
                if fork() = 0 then restore(f(t1), psid)
                else exit(0) end if  // теперь p – наследник init
             end if
         done
         for t1 ← childs(t) do
             j = k : (c_k == f(t1));
             if sid_j = pid_i then
                if fork() = 0 then restore(f(t1),psid) end if
                end if
         done
         start(f(t))
    end if
procedure end
```

Рис. 11. Алгоритм запуска программ из КТ

5.4 Восстановление принадлежности к группам

Процессы внутри одного сеанса могут менять свои группы в произвольное время. Для этого используется системный вызов *setpgid()*, которому передается идентификатор новой группы и процесса, для которого выполняется смена группы. Если какой-то из аргументов нулевой, то считается, что подразумевается текущий. Восстановление принадлежности к группам выполняется после того, как все процессы запущены и им сопоставлены сеансы. Если текущий процесс является лидером группы, то он создает ее с помощью вызова *setpgid(0,0)*. Остальные члены группы выполняют попытки подключиться к ней до тех пор, пока не истечет таймаут (рисунок 12)

```
int ret = 1, i = 0;
struct timespec ts = {0,100000};
// Trial Timeout = 2 seconds
while( ret && ((float)(i*ts.tv_nsec) / 1E9) < 2.0 ){
    ret = setpgid(0,_gid);
    if( ret ){
        nanosleep(&ts,NULL);
    }
    i++;
}
```

Рис. 12. Алгоритм восстановления принадлежности к группе для процесса, не являющегося ее лидером

6. Экспериментальные данные

На рисунке 13 (а,б) показаны структуры двух программ, для которых были созданы КТ с применением ССКТ DMTCP.

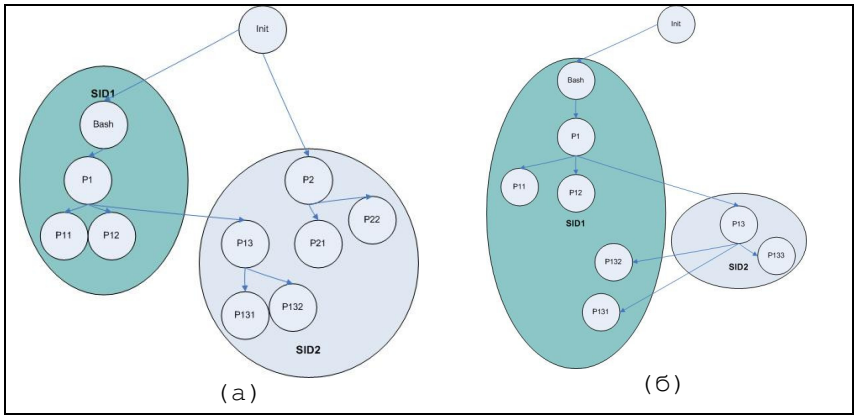


Рис. 13. Структуры тестовых программ

На рисунке 14 (а,б) показаны отношения между восстановленными процессами без применения разработанного алгоритма. Как видно из рисунка 14 (а), все процессы принадлежат одной сессии, а процесс *p2*, который должен быть потомком *init(PID=1),* является потомком *p1*. На рисунке 14 (б) мы также видим, что принадлежность к сессиям не восстанавливается.

На рисунке 15 (а,б) приведены результаты восстановления из КТ с применением предложенного алгоритма для соответствующих программ рисунка 13. Рассмотрим более подробно рисунок 15 (а). Очевидно, что процессы *p13*, *p131*, *p132*, *p2*, *p21*, *p22* принадлежат одной сессии, лидером которой является *p13*, как это и должно быть. Остальные процессы принадлежат сессии командного интерпретатора. Элемент *p2* восстановлен, как наследник *init* (*PID=1*).

На рисунке 15 (б) процессы *p13*, *p133* находятся в новой сессии, в которой *p13* является лидером. При этом потомки *p13 – p131* и *p132* остались в сессии командного интерпретатора, как это было в исходной программе.

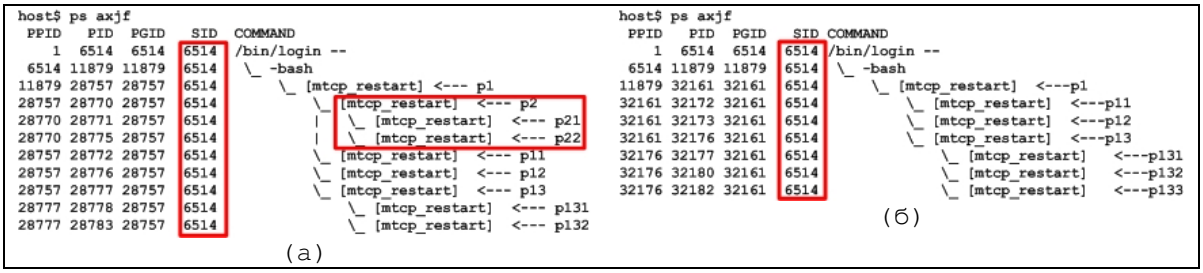


Рис. 14. Восстановление с неполным учетом идентификационной информации

(а) – для программы на рисунке 13(а); (б) – для программы на рисунке 13(б)

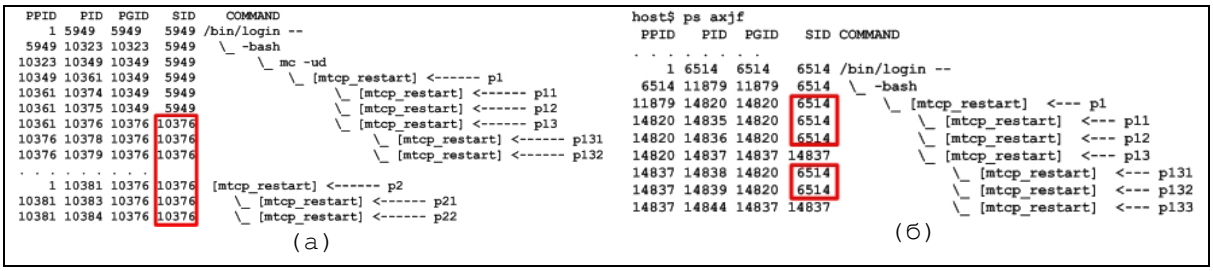


Рис. 15. Полное восстановление идентификационной информации

(а) – для программы на рисунке 13(а); (б) – для программы на рисунке 13(б)

На рисунке 16 (а) показана принадлежность к группам процессов исходной программы, для которой создавались КТ. Программа состоит из пяти процессов, которые принадлежат трем группам с идентификаторами 21226, 21231, 21239. При восстановлении из КТ исходными средствами DMTCP (рисунок 16 (б)) все компоненты остаются в группе, соответствующей некоторой внешней программе. Применение предложенного алгоритма, как показано на рисунке 16 (в), позволяет полностью восстановить исходную программу.

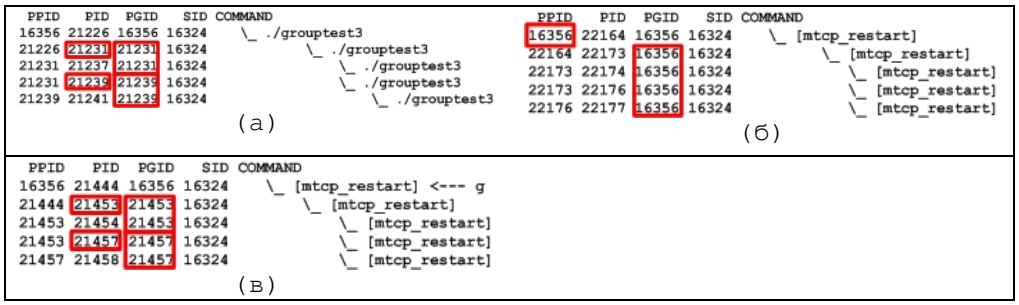


Рис. 16. Восстановление принадлежность к группам

(а) – исходная программа; (б) – восстановление алгоритмом DMTCP;

(в) - восстановление предложенным алгоритмом

На рисунке 17 показано восстановление тестовой программы. Она состоит из двух процессов, взаимодействующих через механизмы IPC. Запуск компонентов выполнен с двух различных терминалов: *pts/0* и *pts/2*. До реализации предложенного расширения схемы синхронизации подобное восстановление исходными средствами DMTCP было невозможно.

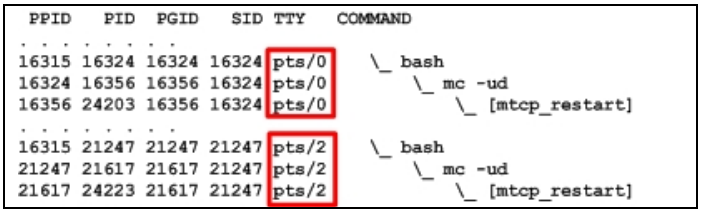


Рис. 17. Восстановление процессов на разных терминалах

7. Заключение

В работе были рассмотрены подходы к восстановлению программ из распределенных контрольных точек, реализованные в ССКТ DMTCP. Предложен алгоритм восстановления отношений родитель-потомок и алгоритм принадлежности к сессиям и группам с использованием стандартного механизма системных вызовов ОС GNU\Linux. Так как DMTCP реализован на уровне системных библиотек, он не имеет прямого доступа к внутренним структурам ядра. Следовательно, для восстановления указанных отношений между процессами требуется имитация основных шагов их запуска. Для этого выполняется построение древовидной структуры, отражающей родственные отношения между узлами. Далее происходит сбор метаинформации, содержащей описание принадлежности к сессиям. Разработанный алгоритм позволяет расширить диапазон программ, поддерживаемых DMTCP.

Расширена схема синхронизации, используемая в DMTCP: добавлен новый барьер, позволяющий выполнять восстановление процессов из РКТ на различных терминалах, расположенных на одном или разных узлах сети. Это позволяет использовать DMTCP для восстановления из РКТ программ, которые не связаны постоянными сетевыми соединениями. Предложенная доработка также необходима для организации реверсивной отладки, построенной на базе DMTCP.

Литература

1. Хорошевский В.Г. Архитектура вычислительных систем. – М.: МГТУ им. Н.Э. Баумана, 2008 . 520 с.
2. TOP500 supercomputer site [Электронный ресурс].- Режим доступа: http://www.top500.org/ . — Загл. с экрана. — яз. англ.
3. Elnozahy E. N., Alvisi L.,Wang Y.M.,Johnson D.B. A survey of rollback-recovery protocols in message-passing systems // ACM Computing Surveys . Vol. 34, No 3, 2002 . pp. 375-408.
4. J. Ansel, K. Arya, G. Cooperman, DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop // Proc. of IEEE International Parallel and Distributed Processing Symposium (IPDPS'09) . IEEE Press, 2009.
5. Paul H. Hargrove and Jason C. Duell Berkeley Lab Checkpoint/Restart (BLCR) for Linux Clusters In Proceedings of SciDAC 2006: June 2006.
6. Michael Litzkow, Todd Tannenbaum, Jim Basney, and Miron Livny. Checkpoint and migration of UNIX processes in the Condor distributed processing system. Technical report 1346, University of Wisconsin, Madison, Wisconsin, April 1997.
7. James S. Plank, Micah Beck, Gerry Kingsley, and Kai Li. Libckpt: Transparent checkpointing under Unix. In Proc. of the USENIX Winter 1995 Technical Conference, pages 213–323, 1995.
8. J. Hursey, J. M. Squyres, T. I. Mattox, and A. Lumsdaine. The design and implementation of checkpoint/restart process fault tolerance for Open MPI. In Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE Computer Society, March 2007.
9. Q. Gao, W. Yu, W. Huang, and D. K. Panda. Application-transparent checkpoint/restart for MPI programs over InfiniBand. Parallel Processing, Jan 2006.
10. Dewolfs, D., Broeckhove, J., Sunderam, V., Fagg, G. "FT-MPI, Fault-Tolerant Metacomputing and Generic Name Services: A Case Study," Lecture Notes in Computer Science, Springer Berlin / Heidelberg, ICL-UT-06-14, Vol. 4192, Number 2006, pp. 133-140, 2006.
11. Ana Maria Visan, Artem Polyakov, Praveen S. Solanki, Kapil Arya, Tyler Denniston, Gene Cooperman Temporal Debugging using URDB .- Режим доступа: http://arxiv.org/abs/0910.5046v1

Параллельные алгоритмы для решения обратных задач переноса примеси

А.В. Старченко, Е.А. Панасенко

При решении обратных задач переноса примеси используются сопряженные уравнения и двойственное представление функционала концентрации примеси. Параллельная реализация строится на основе следующих подходов: геометрическая декомпозиция, функциональная декомпозиция, комбинация функциональной и геометрической декомпозиций.

1. Введение

В настоящее время проблема охраны окружающей среды и ее восстановления становится одной из важных задач науки, развитие которой стимулируется всевозрастающими темпами технического прогресса во всех странах мира. Ухудшение качества атмосферного воздуха, накопление в нём газообразных компонентов происходит вследствие выбросов в атмосферу отходов промышленных предприятий и выхлопных газов автотранспорта, что приводит к ухудшению здоровья населения, а в глобальном масштабе - к изменению климата на планете. При этом наиболее значительный вклад в загрязнение воздуха вносит автотранспорт. Особенность автотранспорта заключается в том, что он является подвижным источником загрязнения, что проявляется в низком его расположении и непосредственной близости к зонам жилой застройки. Все это приводит к тому, что автотранспорт создает в городах обширные зоны, в пределах которых предельно-допустимая концентрация загрязняющих субстанций в атмосферном воздухе превышена в несколько раз.

Поступление загрязняющих веществ в атмосферу избежать невозможно, но разумное использование природных ресурсов и постоянный контроль качества атмосферного воздуха позволят обеспечить безопасный уровень воздействия на атмосферу и избежать глобально негативных последствий.

Одним из способов оценки и прогноза уровня загрязнения воздуха является контроль интенсивности выбросов вредных веществ, который производится с помощью постов наземных наблюдений. Но даже разветвленная сеть таких пунктов наблюдения не всегда может предоставить достоверную информацию для природоохранных служб. Большую помощь здесь может оказать применение методов математического моделирования и, особенно, технологии численного решения обратных задач по определению характеристик источников загрязнения атмосферного воздуха по данным измерений концентрации вредных веществ. Но сложность и взаимосвязанность процессов распространения и переноса загрязняющих субстанций, происходящих в атмосферном воздухе, делают модели оценки и прогнозирования качества воздуха громоздкими в математической записи и весьма требовательными к вычислительным ресурсам. Перспективным способом решения этих проблем является использование современных высокопроизводительных многопроцессорных вычислительных систем, которые обеспечивают существенное ускорение получения результатов расчетов и повышение качества численного прогноза.

Целью данной работы является создание новых вычислительных алгоритмов для решения обратных задач переноса примеси на суперкомпьютерах с параллельной архитектурой.

2. Физическая постановка основной задачи переноса примеси

Распространение примеси в приземном слое атмосферы над промышленным центром и его окрестностями исследуется при следующих условиях. Рассматривается ограниченный в прямоугольнике участок территории, на котором в течение времени исследования происходит выброс от различных источников. Мощность источников зависит от времени. Распределение по облас-

ти исследования поступающей примеси определяется метеорологическими условиями. Кроме того, сделано предположение, что примесь не вступает в химические реакции с другими веществами и скорость ее переноса совпадает со скоростью движения окружающего воздуха (рис. 1).

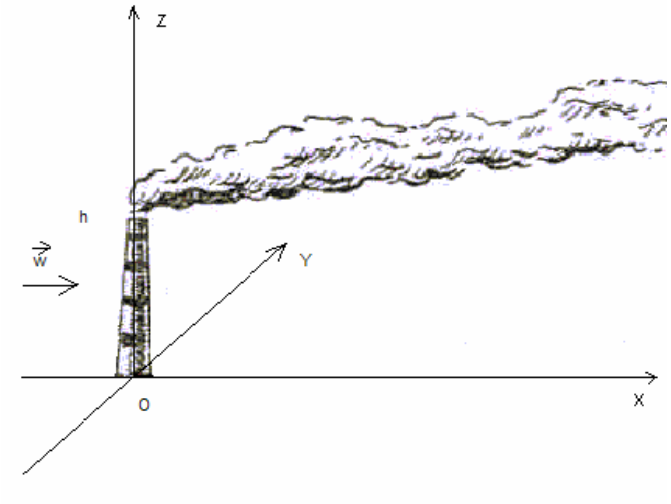


Рис. 1. Распределение поступающей примеси по области исследования

3. Математическая постановка основной задачи переноса примеси

С учетом принятой физической постановки задачи, адвективно-диффузионное уравнение, моделирующее перенос газообразной примеси в заданном потоке, представляется в следующем виде:

$$\frac{\partial C}{\partial t}+U\frac{\partial C}{\partial x}+V\frac{\partial C}{\partial y}+W\frac{\partial C}{\partial z}+\sigma C=\frac{\partial}{\partial x}\left[\Gamma\frac{\partial C}{\partial x}\right]+\frac{\partial}{\partial y}\left[\Gamma\frac{\partial C}{\partial y}\right]+\frac{\partial}{\partial z}\left[K_z\frac{\partial C}{\partial z}\right]+Q, \qquad (1)$$

где C - концентрация примеси; U,V,W - компоненты соленоидального вектора скорости атмосферного воздуха; Γ, K_z - коэффициенты турбулентной диффузии; Q - интенсивность поступления примеси от источников.

Начальные и граничные условия имеют следующий вид:

$$\begin{aligned}
&t=0:\ C(0,x,y,z)=C_0(x,y,z);\\
&x=0:\frac{\partial C}{\partial x}=0;\ x=L_x:\frac{\partial C}{\partial x}=0;\\
&y=0:\frac{\partial C}{\partial y}=0;\ y=L_y:\frac{\partial C}{\partial y}=0;\\
&z=0:K_z\frac{\partial C}{\partial z}=\alpha C-R;\ z=L_z:\frac{\partial C}{\partial z}=0.
\end{aligned} \qquad (2)$$

4. Физическая постановка обратной задачи переноса примеси

Требуется по известным метеорологическим параметрам атмосферы и результатам измерений концентрации примеси в N точках, проводимых в течение некоторого периода времени T, определить параметры (мощность, координаты и время срабатывания) источников примеси (рис. 2).

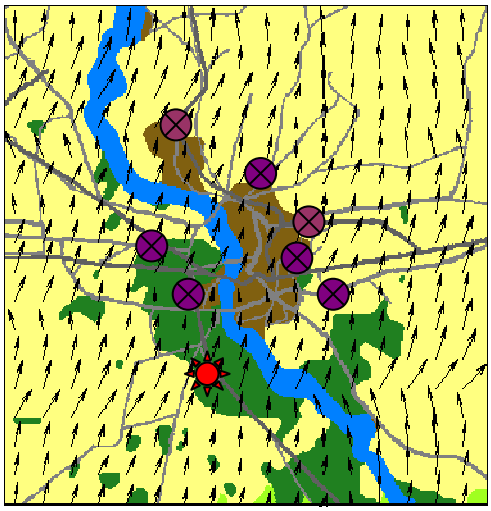


Рис. 2. Определение местонахождения источника выброса

5. Математическая постановка обратной задачи

В данной работе для решения обратных задач привлекается подход Г.И.Марчука [1], который основан на решении уравнения, сопряженного с полуэмпирическим уравнением турбулентной диффузии (1), и двойственным представлением функционала от концентрации примеси.

Математическая постановка обратной задачи переноса примеси включает несколько однотипных сопряженных уравнений (их количество совпадает с количеством измерений концентрации) и поиск минимума некоторого функционала в процессе решения сопряженных уравнений.

Сопряженная к (1) и (2) задача получается следующим образом: уравнение (1) умножается на функции $C_k^* = C_k^*(t,x,y,z)$ и интегрируется по времени и пространству. Тогда используя интегрирование по частям, приходим к сопряженным постановкам:

$$-\frac{\partial C_k^*}{\partial t}-\frac{\partial UC_k^*}{\partial x}-\frac{\partial VC_k^*}{\partial y}-\frac{\partial WC_k^*}{\partial z}+\sigma C^*-\frac{\partial}{\partial x}\left[\Gamma\frac{\partial C_k^*}{\partial x}\right]-\frac{\partial}{\partial y}\left[\Gamma\frac{\partial C_k^*}{\partial y}\right]-\frac{\partial}{\partial z}\left[K_z\frac{\partial C_k^*}{\partial z}\right]=P_k; \qquad (3)$$

$k=1,...,N;$

с соответствующими начальными и граничными условиями:

$$C_k^*(T,x,y,z)=0;$$

$$x=0: UC_k^*+\Gamma\frac{\partial C_k^*}{\partial x}=0;\ x=L_x: UC_k^*+\Gamma\frac{\partial C_k^*}{\partial x}=0;$$

$$y=0: VC_k^*+\Gamma\frac{\partial C_k^*}{\partial y}=0;\ y=L_y: VC_k^*+\Gamma\frac{\partial C_k^*}{\partial y}=0; \qquad (4)$$

$$z=0: K_z\frac{\partial C_k^*}{\partial z}=\alpha C_k^*;\ z=L_z: \frac{\partial C_k^*}{\partial z}=0,$$

где $P_k=\delta(x-x_k)\delta(y-y_k)\delta(z-z_k)\delta(t-t_k)$, N –количество точек наблюдения с координатами (x_k,y_k,z_k), t_k - момент времени измерения концентрации, и двойственному представлению функционала [2]:

$$J(C)=\int_0^T\int_0^{L_x}\int_0^{L_y}\int_0^{L_z} CP_k\,dzdydxdt=C_k=\int_0^T\int_0^{L_x}\int_0^{L_y}\int_0^{L_z} C_k^*Q\,dzdydxdt+\int_0^T\int_0^{L_x}\int_0^{L_y} C_k^*\Big|_{z=0} R\,dydxdt;\ k=1,...,N. \qquad (5)$$

При этом нахождение решения систем уравнений (3)-(4) и (5) позволит установить параметры (мощность, координаты и время срабатывания) источника загрязнения атмосферного воздуха.

6. Численное решение сопряженной задачи

Для численной реализации задачи (3) – (4) использовались метод конечного объёма и явные разностные схемы, для аппроксимации адвективных членов применялась схема MLU Ван Лира [**Ошибка! Источник ссылки не найден.**].

При построении разностного аналога дифференциальной задачи сначала строилась сетка. Для этого выбирались шаги $h_x, h_y, h_z > 0$ и на прямоугольной области $L_x \times L_y \times L_z$ задавалась сетка:

$$\varpi_{h_x,h_y,h_z} = \{x_i = ih_x, \mathrm{y_j} = jh_y, z_l = lh_z), \mathrm{i} = 0,1...,\mathrm{N_x}, \mathrm{j} = 0,1,...,\mathrm{N_y}, 1 = 0,1,...,\mathrm{N_z}\}.$$

Построенная сетка используется при определении векторов скорости и концентрации.

7. Параллельная реализация сопряженной задачи

При численном решении задач переноса примеси выделяют два основных способа параллельной реализации: распараллеливание по физическим процессам (функциональная декомпозиция) и геометрическая декомпозиция расчетной области. При первом способе реализации вычислительные процессы выполняются параллельно и при этом каждый из них занят выполнением принципиально различных задач.

При использовании же декомпозиции расчетной области область исследования делится на подобласти, число которых равно числу процессов, и каждый процесс ведет расчеты в своей подобласти независимо от других. При таком подходе объемы вычислений, которые осуществляет каждый процесс, очень близки между собой [**Ошибка! Источник ссылки не найден.**].

При решении обратной задачи определения характеристик источников на каждом шаге по времени решается не одна, а N независимых сопряженных задач, следовательно, они могут решаться параллельно.

Такие условия проведения численного моделирования позволяют привлекать высокопроизводительную вычислительную технику, в частности, вычислительный кластер ТГУ СКИФ Cyberia.

7.1 Функциональная декомпозиция

Распараллеливание по физическим процессам метода численного решения задачи по определению параметров точечных мгновенных источников производилось с использованием принципа «master-slave» (рис. 3). В этом случае управляющий master-процесс с частотой ω передает каждому slave-процессу значения метеорологических параметров, необходимые для решения сопряженных задач. Подчиненные slave-процессы, получив данные, в свою очередь, ведут расчеты независимо друг от друга, и найденные на каждом шаге по времени приближенные решения своей сопряженной задачи (3) – (4) возвращают управляющему процессу, который ищет глобальный минимум функционала

$$\Phi(C) = \sum_{k=1}^{N} \left(\int_0^T \int_0^{L_z} \int_0^{L_y} \int_0^{L_z} C_k^* Q dz dy dx dt + \int_0^T \int_0^{L_z} \int_0^{L_y} C_k^*\Big|_{z=0} R dy dx dt - C_k \right)^2, \qquad (6)$$

что в итоге позволит определить параметры источников загрязнения.

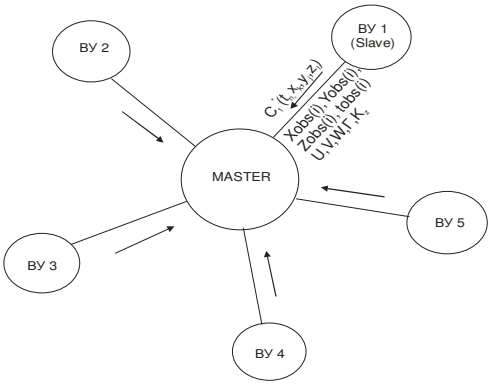


Рис. 3. Схема параллельной реализации

Заметим, что при таком способе организации параллельных вычислений при запуске параллельной программы на $p+1$ процессах большую долю вычислительной работы обычно выполняют p подчиненных slave-процессов. Один управляющий master-процесс координирует работу остальных (подготавливает и рассылает slave-процессам данные для расчета, собирает результаты их расчетов и осуществляет дополнительную обработку этих результатов), причем каждый slave-процесс в данный текущий момент целиком занят выполнением одной задачи.

В работе на основе теоретических оценок и в результате вычислительных экспериментов исследовано ускорение и эффективность данного параллельного алгоритма для задачи идентификации параметров мгновенного источника. Теоретические выкладки приводят к следующим результатам:

$$S_p^{\Phi 1} \approx \frac{p-1}{1+\frac{5\chi\cdot\omega}{m}},\ E_p^{\Phi 1} = \frac{S_p}{p} \approx \frac{1-1/p}{1+\frac{5\chi\cdot\omega}{m}}. \qquad (7)$$

Здесь p - число используемых процессов, m - число арифметических операций для вычисления одного значения сеточной функции, χ - отношение времени межпроцессорной передачи одного числа к времени арифметической операции, $\Phi 1$ - первый вариант функциональной декомпозиции.

Из (7) видно, что при таком способе организации параллельных вычислений не удается достичь идеального параллелизма, поскольку нет хорошей балансировки загрузки используемых в расчетах p процессоров. Это связано с тем, что master-процесс не участвует в проведении массовых вычислений (не решает отдельную сопряженную задачу). Поэтому с целью повышения эффективности алгоритма рассматривается другой подход, в котором master-процесс дополнительно к своей работе также решает одну из N сопряженных задач (3) – (4). В этом случае теоретическая оценка ускорения и эффективности может быть записана следующим образом:

$$S_p^{\Phi 2} \approx \frac{p}{1+\frac{5\chi\cdot\omega}{m}}, E_p^{\Phi 2} \approx \frac{1}{1+\frac{5\chi\cdot\omega}{m}} \qquad (8)$$

и разработанная параллельная программа будет иметь более высокие показатели по времени выполнения ($\Phi 2$ - второй вариант функциональной декомпозиции).

Для подтверждения полученных оценок ускорения параллельного алгоритма был проведен ряд расчетов на кластере ТГУ СКИФ Cyberia. Результаты представлены в таблице 1.

Таблица 1. Время счета (сек) параллельной программы для рассматриваемых вариантов функциональной декомпозиции

Количество измерений, N	Число используемых процессов/время счета для варианта $\Phi 1$	Число используемых процессов/время счета для варианта $\Phi 2$

5	6/312	5/323
10	11/316	10/337
20	21/322	20/354

Из таблицы видно, что оба рассматриваемых варианта имеют высокие показатели масштабируемости, что выражается в слабом изменении времени счета при добавлении процессорных элементов в связи с увеличением объема вычислений (N). Однако первый вариант функциональной декомпозиции алгоритма решения обратной задачи (3)-(5) имеет небольшое преимущество во времени выполнения параллельной программы, что обусловлено появлением дополнительных возможностей совмещения работы активных процессов, количество которых для одних и тех же условий на единицу больше, чем во втором варианте. В то же время соотношение эффективности рассматриваемых параллельных программ $e^{\Phi} = \frac{E_p^{\Phi 1}}{E_p^{\Phi 2}} < 1$ показывает лучшую масштабируемость второго варианта декомпозиции. Что хорошо согласуется с теоретическими оценками (7)-(8), согласно которым $\left(e^{\Phi}\right)_{теор} \approx \frac{p}{p+1}$.

7.2 Геометрическая декомпозиция

При решении каждой сопряженной задачи (3)-(4) в данной работе также использовалась одномерная (вдоль оси OY, рис. 4) геометрическая декомпозиция сеточной области. Этот способ распараллеливания численного алгоритма решения нестационарного адвективно-диффузионного уравнения показал высокую эффективность.

Рассмотрим параллельную реализацию алгоритма решения обратной задачи переноса примеси при использовании принципа геометрической декомпозиции сеточной области. В этом случае вычисления осуществляются следующим образом: N сопряженных задач (3) – (4) последовательно одна за другой решаются численно в подобласти вычислительной сетки, распределенной каждому процессу. Вычисления в течение промежутка времени $[0,T]$ проводятся каждым процессом одновременно, но для корректного расчета сеточных значений функции C_k^* вблизи границ подобласти (рис. 4) требуются некоторые сеточные значения с соседних по расположению подобластей. Это может быть обеспечено лишь путём межпроцессорной передачи данных с использованием, например, стандарта MPI[**Ошибка! Источник ссылки не найден.**,**Ошибка! Источник ссылки не найден.**].

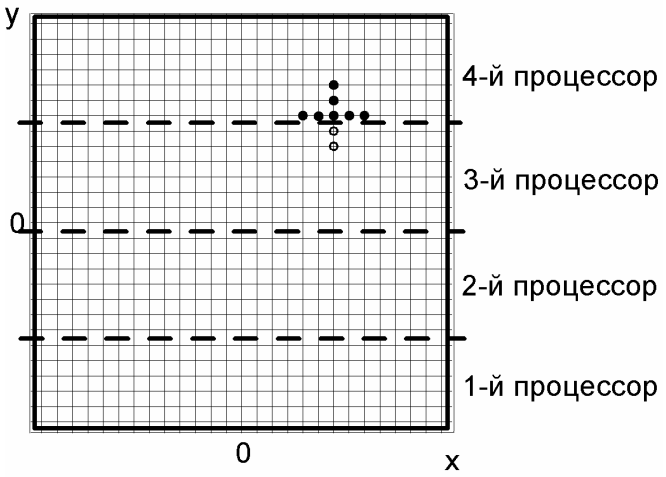


Рис. 4. Расчетная сетка в плоскости xOy с указанием распределения подобластей по четырем процессорам. На приграничной сеточной линии приведен разностный шаблон. Открытые кружки указывают на сеточные значения, которые необходимо получить с соседнего процессора

Теоретическая оценка ускорения и эффективности данного алгоритма имеет следующий вид:

$$S_p^{\Gamma} \approx \frac{p}{1+\frac{\chi}{m}\left(\frac{5\omega}{N}+\frac{4p}{N_x}\right)}, E_p^{\Gamma} = \frac{S_p^{\Gamma}}{p} \approx \frac{1}{1+\frac{\chi}{m}\left(\frac{5\omega}{N}+\frac{4p}{N_x}\right)}, \quad (9)$$

где N_x - размер вычислительной сетки по оси Ox, N - количество решаемых сопряженных задач.

Поскольку в полученной оценке эффективности рассматриваемого способа распараллеливания вклад межпроцессорной передачи данных во временные затраты более значительный, стоит ожидать, что применение геометрической декомпозиции при решении обратных задач переноса примеси, в которых на каждом шаге по времени требуется передача данных master – процессу, менее перспективно, чем использование функциональной декомпозиции.

Для подтверждения сделанных выводов были проведены вычислительные эксперименты, результаты которых представлены в таблице 2.

Таблица 2. Время счета параллельной программы для геометрической декомпозиции

Количество измерений, N	Число используемых процессов, p	Время, сек
5	5	627
10	10	643
20	20	743

Сопоставляя результаты, приведенные в таблицах 1 и 2 видно, что применение геометрической декомпозиции для рассматриваемой задачи идентификации параметров внезапного выброса в атмосферу по данным измерений существенно (более чем в два раза) уступает по быстродействию функциональной декомпозиции. Причиной этого являются коммуникационные затраты на межпроцессорную передачу данных, необходимую для корректного решения сопряженных задач (3) – (4) на всей сеточной области, разделенной между активными процессами [**Ошибка! Источник ссылки не найден.**].

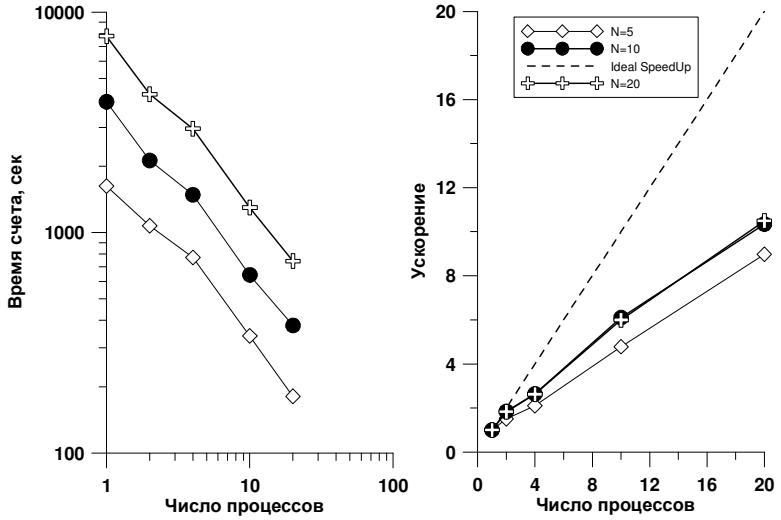


Рис. 5. Время счета (слева) и ускорение (справа) параллельной программы решения обратной задачи переноса примеси, базирующейся на принципе одномерной декомпозиции расчетной области

Данные, представленные на рисунке 5, показывают, что при увеличении числа активных процессов время счета уменьшается, причем пропорционально, для различного N. В целом, этот способ распараллеливания алгоритма решения обратной задачи для рассмотренных здесь

условий обеспечивает 50% эффективность параллельной программы для числа процессов $p = 2,4,10,20$, что свидетельствует о неплохой масштабируемости параллельной реализации для $p \le 20$.

7.3 Комбинированный способ распараллеливания

Рассмотренные выше подходы создания параллельных версий алгоритмов решения обратной задачи переноса примеси показали неплохие результаты по эффективности, однако количество используемых активных процессов в них ограничено: в случае функциональной декомпозиции числом проведенных измерений N, а при одномерной геометрической декомпозиции размером сетки N_x и выбранным сеточным шаблоном.

При использовании комбинированного подхода параллельной реализации предлагается для увеличения в расчетах количества используемых активных процессов совместить применение функциональной и геометрической декомпозиции. Все p активных процессов делятся на N групп (по числу измерений), каждая из которых решает численно одну сопряженную задачу (3),(4), используя одномерную геометрическую декомпозицию. Для получения приближенного решения процессы отдельной группы обмениваются между собой рассчитанными значениями сеточной функции вблизи границ подобластей. Поиск минимума функционала выполняет один из p процессов, которому остальные передают результаты промежуточных расчетов.

Теоретические оценки такого способа параллельной реализации записываются в следующем виде:

$$S_p^{комб} \approx \frac{p}{1+\frac{\chi}{m}\left(5\omega+\frac{4p}{N\cdot N_x}\right)},\ E_p^{комб}=\frac{S_p^{комб}}{p}\approx\frac{1}{1+\frac{\chi}{m}\left(5\omega+\frac{4p}{N\cdot N_x}\right)},\ \frac{p}{N}=1,2,3,... \qquad (10)$$

Сравнивая оценки параллельных реализаций (9) и (10) можно отметить, что эффективность комбинированного подхода не хуже, чем при геометрической декомпозиции, однако количество используемых активных процессов можно увеличить на порядок, что при имеющей место масштабируемости параллельной версии алгоритма должно привести к значительному сокращению времени счета.

Результаты вычислительных экспериментов для задачи идентификации характеристик мгновенного источника подтвердили предварительные теоретические оценки (таблица 3).

Таблица 3. Время счета параллельной программы для комбинации функциональной и геометрической декомпозиции

Количество измерений, N	Время счета последовательной программы, сек	Число используемых процессов/Время счета для комбинированного подхода, сек
5	1626	100/33
10	3922	200/42
20	7799	400/59

Стоит отметить, что результаты численных расчетов при использовании комбинированного подхода показывают снижение времени счета с уменьшением вычислительной трудоемкости задачи, замедление ускорения параллельной программы с ростом числа используемых процессов и при сокращении количества сопряженных задач N. Комбинированный подход действительно на порядок позволяет увеличить число используемых процессов, при этом эффективность параллельной программы остается на уровне не менее 50%, что является неплохим для задач такого класса показателем. Однако, в целом, функциональная декомпозиция имеет существенное преимущество, судя по (8), (9) и (10).

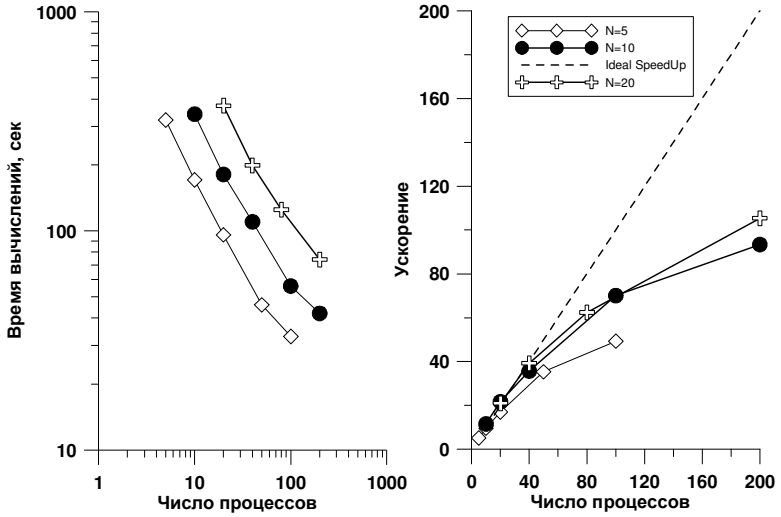


Рис. 6. Время счета (слева) и ускорение (справа) параллельной программы решения обратной задачи переноса примеси (комбинированный способ распараллеливания)

8. Заключение

В работе с единых позиций сформулированы математические постановки для решения обратных задач по определению параметров источника загрязнения атмосферного воздуха по измеренным значениям концентрации примеси. Математические постановки опираются на аппарат сопряженных уравнений и двойственное представление функционала от концентрации примеси. Для численного решения сопряженных задач используются конечно-разностные методы, явные разностные схемы, метод конечного объема. Для ускорения численного решения задачи обоснованы и применены следующие способы параллельной реализации задач переноса примеси: функциональная декомпозиция, геометрическая декомпозиция и комбинированный способ. Проведены оценки ускорения и эффективности всех трех подходов параллельной реализации. По произведенным оценкам ускорения и эффективности способов параллельной реализации получено, что, в целом, функциональная декомпозиция имеет существенное преимущество по сравнению с остальными подходами.

ЛИТЕРАТУРА

1. Марчук Г.И. Математическое моделирование в проблеме окружающей среды. М.: Наука, 1982.
2. Панасенко Е.А., Старченко А.В. Определение городских районов-загрязнителей атмосферного воздуха по данным наблюдений // Оптика атмосферы и океана. 2009. Т.22, № 03.С. 279-283.
3. Van Leer B. Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second order scheme // J. Comput. Phys. 1974. V.14. P. 361 – 370.
4. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПБ.: БХВ – Петербург, 2002.
5. Message Passing Interface Forum. MPI: A message – Passing Interface Standart // International Journal of Super Computer Applications. 1994. Vol. 8(314). P. 165 – 414.
6. Малышкин В.Э., Корнеев В.Д. Параллельное программирование мультикомпьютеров. Н-ск: изд-во НГТУ, 2006.
7. Старченко А.В. Параллельные вычисления в задачах охраны окружающей среды // Вторая Сибирская школа-семинар по параллельным вычислениям: Сб. науч. тр. / Томск: Изд-во Том. ун-та, 2004. С. 17-22.

Использование графических процессоров для решения разреженных СЛАУ итерационными методами подпространств Крылова с предобуславливанием на примере задач теории фильтрации

Д.А. Губайдуллин, Р.В. Садовников, А.И. Никифоров

В данной работе представлена реализация на современных графических процессорах NVIDIA библиотеки итерационных методов подпространств Крылова с предобуславливанием, на сегодняшний день наиболее общей группы методов, используемых в приложениях для решения больших разреженных систем линейных алгебраических уравнений (СЛАУ). Рассмотренные методы работают для матриц разреженных СЛАУ самого общего вида, с нерегулярной структурой, как симметричных, так и несимметричных. Библиотека итерационных методов предназначена для пользователей, которые хотят избежать трудностей и деталей параллельного программирования на графических процессорах, но которым приходиться иметь дело с большими разреженными СЛАУ и необходимо использовать эффективность параллельной архитектуры графических процессоров.

1. Введение

Наряду с темой высокопроизводительных вычислений и суперкомпьютеров сегодня все шире обсуждается новое направление в ускорении расчетов – использование графических процессорных устройств (ГПУ). Изначально графические процессоры не предназначались для высокопроизводительных вычислений, а разрабатывались для воспроизведения трехмерных изображений в реальном времени. Применение ГПУ для математических расчетов затруднялось не только сложностью архитектуры, но и отсутствием удобного программного интерфейса. Программистам приходилось использовать ГПУ через стандартный графический интерфейс — OpenGL или DirectX, когда данные к видеочипу передавались в виде текстур, а расчетные программы загружались в виде шейдеров.

Начиная с 2006 г., когда ведущие производители процессоров AMD и NVIDIA выпустили программные средства для взаимодействия с графическими процессорами в обход интерфейсов для работы с графикой, появилась возможность для написания программ, выполняемых на графических устройствах. Наибольшую популярность для расчётов на ГПУ приобрела, представленная компанией NVIDIA, унифицированная архитектура компьютерных вычислений CUDA (Compute Unified Device Architecture), которая хорошо подходит для решения широкого круга задач с высоким параллелизмом. Технология NVIDIA CUDA [1], основанная на расширении языка C, даёт возможность организации доступа к набору инструкций ГПУ и управления его памятью при организации параллельных вычислений.

При программировании с использованием ГПУ, последнее рассматривается как вычислительное устройство, способное выполнять большое число одинаковых вычислений параллельно. Это особенно удобно для применения в задачах линейной алгебры в операциях над матрицами и векторами, и к сегодняшнему дню уже разработаны алгоритмы для расчетов на ГПУ, позволяющие ускорить расчеты с плотными матрицами в десятки, а иногда и в сотни раз [2].

В настоящее время рядом авторов [3,4] разрабатываются алгоритмы для расчетов с разреженными матрицами на ГПУ, которые играют важную роль в итерационных методах решения разреженных СЛАУ. Такие системы уравнений возникают при численном решении широкого класса задач математической физики, описываемых дифференциальными уравнениями в частных производных [5]. Как правило, матрицы этих систем имеют большую размерность. Одной из самых критичных, относительно времени выполнения, операций в итерационных методах решения разреженных СЛАУ является операция умножения матрицы на вектор. Из-за непредсказуемого шаблона доступа к памяти и более сложной структуры данных

для представления разреженных матриц построение эффективных алгоритмов для этих операций затруднено. В работах [3,4] рассмотрены различные варианты форматов представления разреженных матриц и способы параллельной реализации операции умножения матрицы на вектор на ГПУ, а также вопросы их оптимизации.

Одна из первых работ, посвященных реализации алгоритмов решения СЛАУ с разреженной матрицей на ГПУ, является работа [6], в которой предложена реализация метода сопряженных градиентов. И хотя метод нельзя назвать универсальным, так как он ограничен использованием только симметричных матриц, тем не менее, в работе была предложена концепция использования итерационных методов на ГПУ. В работе [7], на основе [6], предложена реализация стабилизированного метода бисопряженных градиентов на ГПУ, который позволяет работать с несимметричными разреженными матрицами. Однако, очевидно, что не существует одного метода, подходящего для любого класса задач, поскольку, один и тот же метод может сходиться или не сходиться к решению в зависимости от типа разреженной матрицы и предобуславливателя. Поэтому, необходимо иметь коллекцию итерационных методов, работающих с различными типами матриц и предобуславливателей.

В данной работе представлена библиотека итерационных методов подпространств Крылова с предобуславливанием для решения разреженных СЛАУ с нерегулярной структурой на ГПУ. Представленные в библиотеке методы протестированы на примере численного решения задачи фильтрации методом контрольных объемов. Проведенные тесты показали, что использование ГПУ позволяет значительно ускорить расчеты.

2. Методы подпространств Крылова с предобуславливанием и их реализация на ГПУ

Основное преимущество итерационных методов перед прямыми методами решения разреженных СЛАУ заключается в минимальных требованиях к памяти для хранения матриц, а также в том, что итерационные методы вместо матрично-матричных операций умножения используют матрично-векторные и работают с результирующими векторами. Другое важное преимущество итерационных методов заключается в том, что эти методы хорошо распараллеливаются [8]. Наиболее эффективной группой итерационных методов, применяемой в настоящее время в линейной алгебре для решения разреженных СЛАУ, являются методы подпространств Крылова с предобуславливанием. К этой группе принадлежат следующие методы: CG – метод сопряженных градиентов, CGS – квадратичный метод сопряженных градиентов, BiCGSTAB – стабилизированный метод бисопряженных градиентов, GMRES – обобщенный метод минимальных невязок и др.

Итерационные методы решения разреженных СЛАУ, обычно включают следующие основные операции линейной алгебры: матрично-векторные умножения, умножение матрицы на вектор, линейные комбинации векторов (операции сложения, вычитания, умножения векторов на скаляр), скалярное произведение векторов, вычисление норм векторов и матриц, вычисление предобуславливателей, операции с предобуславливателями. Производительность этих операций может быть значительно повышена применением высокопроизводительных вычислений и массовый параллелизм, которым обладают ГПУ, предоставляет возможность их использования для решения задач линейной алгебры.

2.1. Реализация операции матрично-векторного произведения на ГПУ

Наиболее критичной, с точки зрения времени выполнения, операцией в итерационных методах решения разреженных СЛАУ является операция умножения разреженной матрицы на вектор. Способ вычисления этой операции на любой архитектуре вычислителя, в том числе на ГПУ, зависит от формата представления разреженной матрицы, который, в свою очередь, связан со спецификой рассматриваемой задачи, с типом используемой сетки (количество ненулевых элементов матрицы определяется связями узлов сетки) и способом аппроксимации. Известно довольно много схем хранения разреженных матриц, которые встречаются в

расчетах. Цель каждой из схем заключается в увеличении эффективности, как в использовании памяти, так и в уменьшении количества арифметических операций [5,8].

Для матриц с произвольной нерегулярной структурой ненулевых элементов, одним из самых распространенных форматов хранения разреженной матрицы является формат CSR (CSC) – формат сжатой разреженной строки (столбца). Другой формат MSR (MSC) – модифицированный формат разреженной строки (столбца) отличается от CSR (CSC) тем, что главная диагональ матрицы хранится отдельно от недиагональных членов в каждой строке (столбце). В нашей библиотеке реализованы эти, наиболее общие и часто используемые, форматы представления матриц. Выбор этих форматов для использования мотивирован несколькими факторами: их простотой, общностью представления, широким использованием и довольно легкой параллельной реализацией. В дальнейшем можно распространить этот подход и на другие форматы матриц.

Не останавливаясь на способе реализации операции умножения матрицы в формате CSR(CSC) на вектор на ЦПУ и ГПУ, который подробно изложен в работах [3,4], рассмотрим реализацию этой операции для матрицы в формате MSR (MSC). Структура данных для хранения матрицы в формате MSR (MSC) представляет собой два массива (размером $nnz+1$): вещественный массив, содержащий ненулевые значения матрицы, которые хранятся строка за строкой (столбец за столбцом), и один целочисленный массив, первые n позиций которого содержат значения главной диагонали, а последующие позиции содержат индексы столбцов (строк) элементов и указатели на начало каждой строки (столбца) в матрице.

```
for i=1, n
  start = bindx[i]
   stop = bindx[i+1]-1
    dot = A[i]*x[i]
  for j=start, stop
   dot += A[j]*x[bindx[j]]
  end
  y[i] = dot
end
```

Рис. 1. Псевдокод умножения матрицы в MSR формате на вектор на ЦПУ

```
index = get_thread_index ();
if ( index < n )
  start = bindx[index]
   stop = bindx[index+1]-1
    dot = A[index]*x[index]
  for j=start, stop
    dot += A[j]*x[bindx[j]]
  end
  y[index] = dot
end
```

Рис. 2. Псевдокод умножения матрицы в MSR формате на вектор на ГПУ

Версия последовательного алгоритма умножения разреженной матрицы $\mathbf{A}$ в формате MSR(MSC) на вектор $\mathbf{x}$ на ЦПУ, представлена на рис. 1. Для реализации матрично-векторного произведения $\mathbf{y}=\mathbf{Ax}$ параллельно на ГПУ, заметим, что каждая компонента результирующего вектора $\mathbf{y}$ может быть вычислена независимо, как скалярное произведение i-ой строки матрицы на вектор $\mathbf{x}$. Факт, что внешний цикл может быть выполнен параллельно, используется многими параллельными платформами. Для параллельного выполнения операции умножения разреженной матрицы в MSR (MSC) формате на вектор на ГПУ каждый из двух массивов просто копируется из памяти ЦПУ в память ГПУ, а далее каждая строка матрицы обрабатывается отдельным потоком. Псевдокод для выполнения умножения MSR (MSC) матрицы на ГПУ может быть записан в виде, представленном на рис. 2.

Конечно, представленные форматы матриц обладают тем недостатком, что количество ненулевых элементов и их положение в каждой строке (столбце) может значительно отличаться, что будет сказываться на производительности матрично-векторного произведения, так как потоки для строк (столбцов) с меньшим количеством ненулевых элементов будут выполняться быстрее, по сравнению с потоками для строк (столбцов) с большим количеством. Однако, этот недостаток компенсируется общим временем вычислений, которое уменьшается вследствие выполнения одних и тех же вычислений над разными данными одновременно большим количеством потоков.

```
for i=1, n
  start = bindx[i]
   stop = bindx[i+1]-1
   y[i] = A[i]*x[i]
  for j=start, stop
    y[bindx[j]] += A[j]*x[i]
  end
end
```

Рис. 3. Псевдокод умножения транспонированной матрицы в MSR формате на вектор на ЦПУ

Другая операция, которая часто встречается в итерационных методах решения разреженных СЛАУ это операция умножения транспонированной матрицы на вектор $\mathbf{y} = \mathbf{A}^{\mathrm{T}}\mathbf{x}$. Как видно на рис. 3, в алгоритме последовательной реализации этой операции вместо явного транспонирования матрицы, используется косвенная адресация. Сложность параллельной реализации этой операции на ГПУ, заключается в том, что разные потоки могут одновременно модифицировать один и тот же элемент вектора $\mathbf{y}$ и результаты могут быть непредсказуемыми. Способы реализации этой операции с помощью атомарных функций и разделения массива $\mathbf{y}$ на множество массивов $\mathbf{y}_{\mathrm{thld}}$ без наложения элементов оказались неэффективными. Непосредственное вычисление транспонированной разреженной матрицы также требует значительных затрат времени. Поэтому вопрос с умножением транспонированной матрицы на вектор остается пока открытым, по крайней мере, для тех форматов хранения разреженных матриц, которые рассмотрены в этой статье. Это, конечно, ограничивает возможности применения итерационных методов, которые используют такое произведение.

2.2. Предобуславливание

Скорость сходимости итерационных методов зависит от спектральных характеристик матрицы [5,8]. Для улучшения спектральных свойств исходная матрица умножается на матрицу предобуславливания, которая улучшает спектр, что увеличивает скорость сходимости. Поэтому, помимо самого итерационного метода важной частью вычислительной методики является предобуславливание матрицы. Использование предобуславливателя вносит дополнительные вычисления на этапе его построения и применения на каждой итерации. Однако, и выигрыш в скорости сходимости может быть значительным. Некоторые предобуславливатели практически не требуют никаких вычислительных затрат на этапе построения, а другие, наоборот, (например, неполная факторизация) требуют значительных затрат на вычисления и трудно распараллеливаются.

Сейчас в библиотеке реализован k – шаговый диагональный предобуславливатель Якоби (где $k>0$ - показатель степени), а также предобуславливание полиномами Неймана и полиномами наименьших квадратов [8].

2.3. Реализация операций с векторами на ГПУ

Для операций модификации векторов, вычисления скалярных произведений векторов, CUDA предоставляет реализацию процедур библиотеки BLAS (Basic Linear Algebra Subprograms) для векторов и плотных матриц, которая подробно описана в руководстве по программированию [1]. Поэтому для операций с векторами мы воспользовались CUDA BLAS.

2.4. Схема реализации методов на ГПУ

В реализации на ГПУ итерационных методов Крылова с предобуславливанием мы ограничились рассмотрением только тех методов, которые допускают использование самой матрицы без операции транспонирования, в силу указанных выше сложностей вычислений с транспонированной матрицей. Поэтому, среди методов, которые были реализованы в составе

библиотеки, следующие итерационные методы: IR – метод Ричардсона, Cheby – метод Чебышева, CG – метод сопряженных градиентов, CGS – квадратичный метод сопряженных градиентов, BiCGSTAB – стабилизированный метод бисопряженных градиентов, и GMRES – обобщенный метод минимальных невязок. Все эти методы спроектированы с использованием предобуславливания как полиномами Неймана и полиномами наименьших квадратов, так и k - шагового диагонального предобуславливателя Якоби. Все методы записаны в виде функций в формате шаблона, так что они могут использоваться с любой матрицей и вектором, обеспечивая необходимый уровень функциональности. Детали реализации структуры данных отделены от математического алгоритма с помощью объектно-ориентированного программирования на языке C++. Основные этапы реализации методов на ГПУ следующие: 1) загрузить вектора и матрицу (в одном из форматов) с ЦПУ на ГПУ; 2) произвести операции умножения, сложения, вычитания с векторами, матрицами и предобуславливателями; 3) загрузить результат с ГПУ на ЦПУ.

3. Численные результаты

В данном разделе представлены результаты тестирования описанной выше библиотеки итерационных методов подпространств Крылова с предобуславливанием на примере численного решения задачи фильтрации жидкости к скважинам в двухмерной области произвольной формы. Тестирование проводилось на трех видеокартах семейства NVIDIA GeForce: 9600M GT (32 ядра с частотой 500 МГц, 4 потоковых мультипроцессоров, 512 МБ DRAM DDR3), 9600 GT (64 ядра с частотой 650 МГц, 8 потоковых мультипроцессоров, 512 МБ DRAM DDR3 с полосой пропускания 57,4 ГБ/с), GTS 250 (128 ядер с частотой 745 МГц, 16 потоковых мультипроцессоров, 1 ГБ DRAM DDR3 с полосой пропускания 70,4 ГБ/с). А также на вычислительном модуле NVIDIA Tesla C1060 (240 ядер с частотой 1296 МГц, 4 ГБ DRAM DDR3 с полосой пропускания 102 ГБ/с). Время решения на ГПУ сравнивалось со временем решения задачи на ЦПУ настольного компьютера (Intel Core 2 Duo E6600 2,4 ГГц, 2 ГБ DRAM, 4 ГБ L2, Windows Vista Business 64 bit). Все вычисления на видеокартах проводились с одинарной точностью, поскольку, только вычислительный модуль Tesla поддерживает вычисления с двойной точностью.

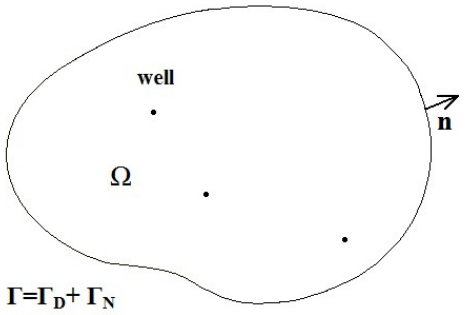


Рис. 4. Область фильтрации

3.1. Постановка задачи

Рассматривается однофазная установившаяся фильтрация несжимаемой жидкости к скважинам в двухмерной области произвольной формы (рис.4). Уравнение фильтрации жидкости в пористой среде имеет вид:

$$\nabla\left(\frac{\mathbf{k}}{\mu}\nabla p\right) = q(\delta(M_w)),\ (x,y)\in\Omega, \tag{1}$$

с граничными условиями

$$p(x,y) = p_R,\ (x,y)\in\Gamma_D, \tag{2}$$

$$\frac{\partial p}{\partial n} = q_R,\ (x,y)\in\Gamma_N, \tag{3}$$

где $p(x,y)$ - давление в жидкости, $\mathbf{k} = \begin{pmatrix} k_{xx} & k_{xy} \\ k_{yx} & k_{yy} \end{pmatrix}$ - тензор проницаемости, μ - вязкость, M_w - точка, в которой расположен центр скважины, Ω - область с границей $\Gamma = \partial\Omega = \Gamma_D \cup \Gamma_N$.

В представленном выше уравнении, на скважине может задаваться как условие забойного давления, так и дебита. Когда на скважине задано условие забойного давления, дебит скважины является неизвестным, и наоборот, когда на скважине задан дебит, неизвестным является забойное давление. Для определения этих значений используется модель точечного источника-стока, которая широко применяется в задачах такого типа [9].

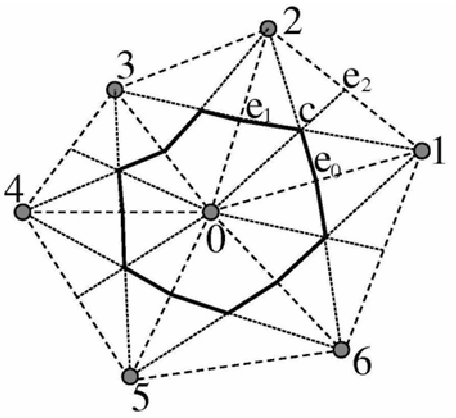


Рис. 5. Построение CVFE сетки

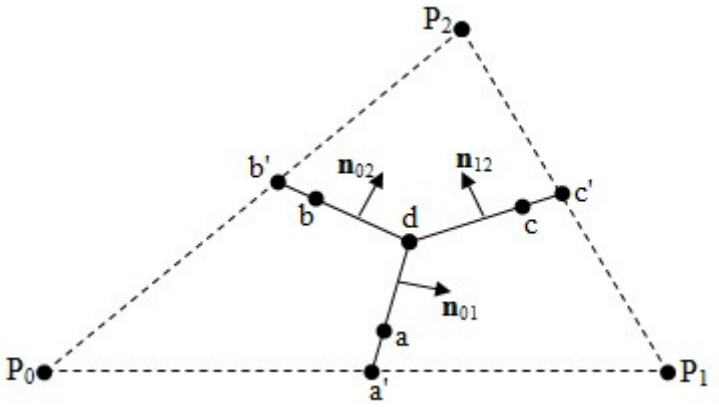


Рис. 6. Положение точек на границах контрольного объема

3.2. Аппроксимация задачи методом контрольных объемов

Для решения поставленной задачи методом контрольных объемов область сначала покрывается нерегулярной сеткой треугольников Делоне. Затем строятся контрольные объемы, в качестве вершин которых выбираются центры тяжести треугольников, лежащие на пересечении медиан. Полученный таким образом контрольный объем представляет, в общем случае, невыпуклый (звездчатый) полигон (рис. 5). Для выполнения условия непрерывности скорости и потенциалов давления на границах контрольного объема должны выполняться условия равенства скоростей и потенциалов в серединах ребер треугольников (рис. 6). Уравнение метода контрольных объемов получается из условия баланса потоков через грани контрольного объема сеточного блока и имеет вид:

$$\sum_{j=1}^{n_c}\left[\sum_{k=1}^{n_{sc}}\left(\sum_{l=1}^{n_{sc,n}}T_{kl}P_l\right)\right]=0, \tag{4}$$

где T_{kl} - среднегармоническое значение проводимости в узлах сетки, P_l - значение потенциала давления в узлах сетки.

Так как размеры сеточного блока значительно больше диаметра скважины, то давление в сеточном блоке (в котором закончена скважина) не может полагаться равным забойному давлению p_{wf}. На самом деле, это давление равно давлению на расстоянии r_0 от сеточного узла для радиального потока [9], т.е.

$$q=2\pi\frac{kh}{\mu}\frac{p_{wf}-p_0}{\ln\left(\frac{r_0}{r_w}\right)}, \tag{5}$$

где $k=\sqrt{k_{xx}k_{yy}}$, h - толщина пласта. Уравнение (5), которое дает соотношение между давлением сеточного блока p_0, забойным давлением p_{wf} и дебитом q называется моделью скважины. Для расчета забойного давления и фиктивного радиуса скважины r_0 с помощью уравнения (5) используются следующие допущения: ось скважины параллельна одной из координатных осей; скважина полностью вскрывает пласт. Тогда, рассматривая баланс

жидкости в окрестности скважины, можно получить следующее выражение для фиктивного радиуса скважины [9]:

$$\ln r_0 = \frac{\sum_u T_u \ln r_u - 2\pi kh}{\sum_u T_u}. \qquad (6)$$

Модель скважины, заданная уравнением (6), действительна только при соблюдении перечисленных выше допущений. Результирующая матрица системы уравнений (4) имеет разреженную структуру, в которой количество ненулевых позиций в каждой строке определяется количеством связей данного узла с другими узлами сетки.

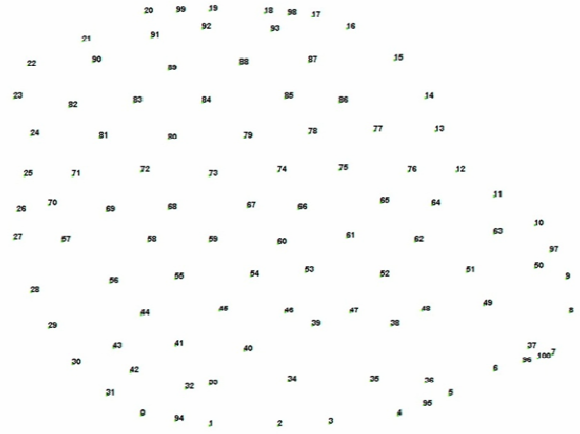


Рис. 7. Граница расчетной области и положение скважин

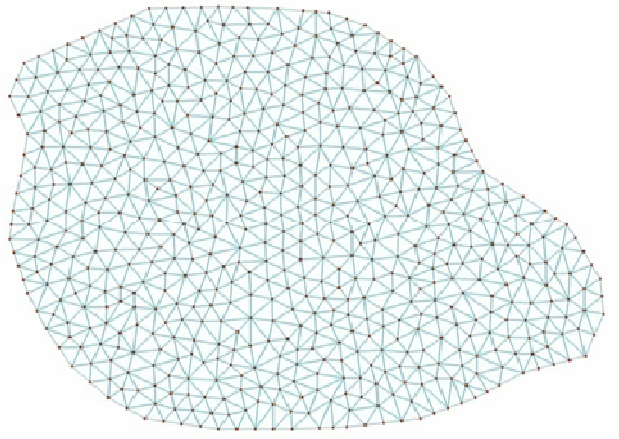


Рис. 8. Триангуляция области

3.3. Результаты расчетов

В примере рассматривалась двумерная область фильтрации, представленная на рис.7, с количеством скважин 64. Размеры области по простиранию 3500 м на 3500 м. . Значения компонент тензора коэффициентов фильтрации принимались постоянными во всей области фильтрации и равными $k_{xx} = 0{,}125$ мкм^2, $k_{xy} = 0{,}023$ мкм^2, $k_{xy} = 0{,}105$ мкм^2, $k_{yy} = 0$, 325 мкм^2. На скважинах задавались объемные дебиты, значения которых варьировались в пределах от 7,6 до 67,5 м^3/сут. Триангуляция области представлена на рис. 8. Расчеты проводились на сетках с различным количеством узлов, представленным в таблице 1.

Таблица 1. Данные для сеток с различным количеством узлов

Количество узлов сетки	Количество		
	ребер	треугольников	ненулевых элементов
178060	532562	354503	1243370
356341	1066784	710444	2490095
888616	2662294	1773679	6213390
1776694	5324793	3548162	12426404

На рис. 9-16 приведены результаты параллельного решения систем уравнений итерационными методами подпространств Крылова с диагональным предобуславливанием на ГПУ и ЦПУ. На всех диаграммах ось ординат – количество точек сетки. На рис.9-12 представлено время (в секундах), за которое невязка решения сходится к заданной точности 10^{-6}, или, в случае отсутствия сходимости, выполняется максимальное количество итераций, равное 2500. На рис.13-16 представлено ускорение вычислений на ГПУ по сравнению с ЦПУ.

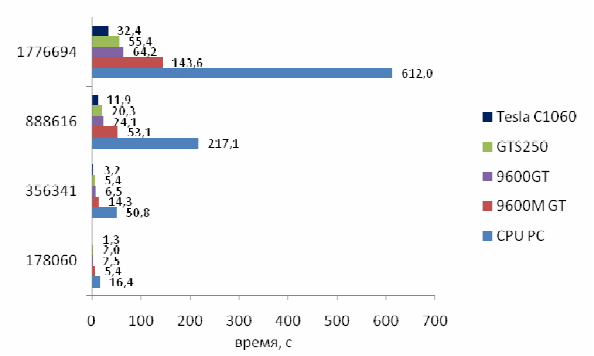


Рис. 9. Метод CG

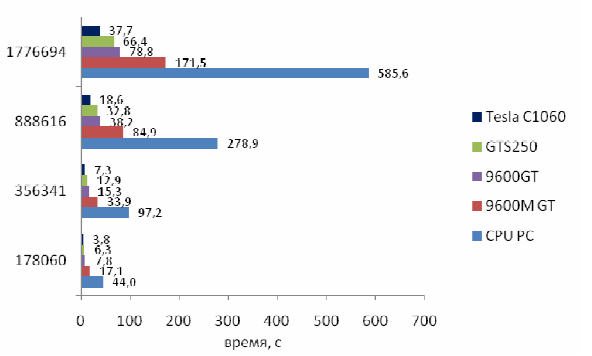


Рис. 10. Метод IR

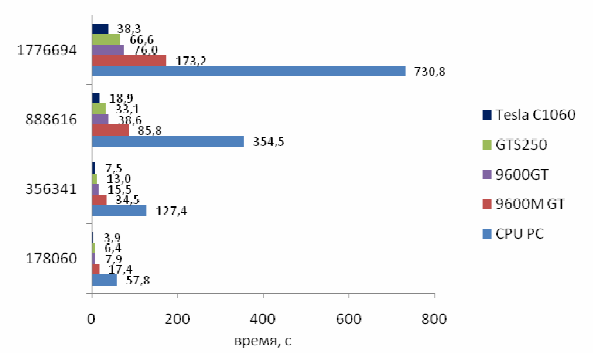


Рис. 11. Метод CHEBY

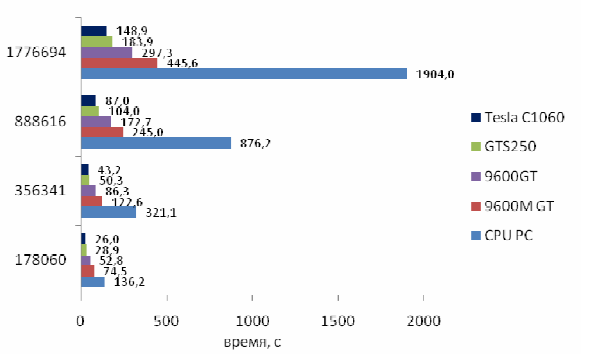


Рис. 12. Метод GMRES

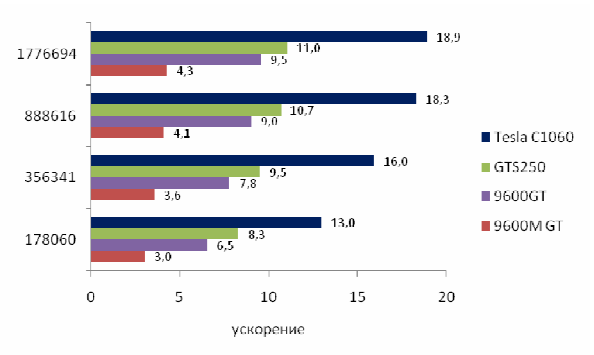


Рис. 13. Ускорение расчетов по методу CG

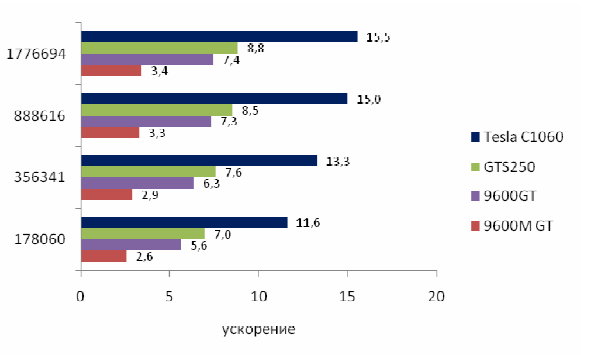


Рис. 14. Ускорение расчетов по методу IR

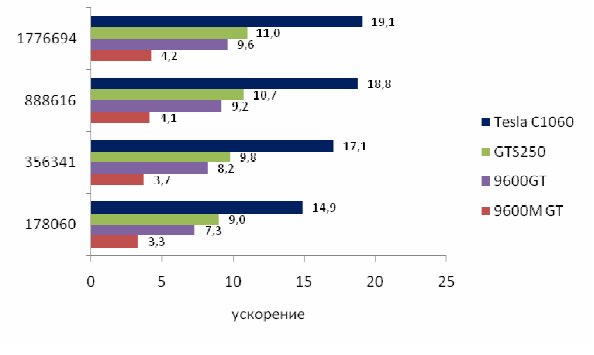


Рис. 15. Ускорение расчетов по методу CHEBY

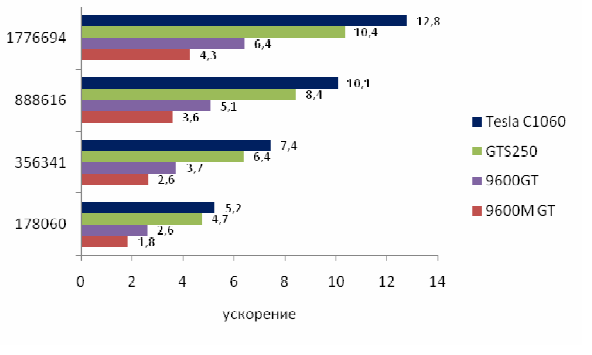


Рис. 16. Ускорение расчетов по методу GMRES

Ускорение подсчитывалось делением времени решения задачи на ЦПУ на время решения на ГПУ.

Из представленных результатов видно, что с увеличением количества неизвестных эффективность расчетов на ГПУ повышается. Ускорение расчетов на стандартных видеокартах NVIDIA GeForce варьируется в пределах 3-12 раз в зависимости от метода, количества точек и типа видеокарты. Так как возможности использования видеокарт NVIDIA GeForce в качестве отдельного вычислителя у нас не было, то часть их ресурсов была занята воспроизведением видеоизображения на экране монитора. Поэтому приведенные результаты не отражают полную производительность методов на видеокартах, но даже в этом случае, позволяют судить о значительном выигрыше в производительности. На вычислительном модуле NVIDIA Tesla C1060 ускорение достигает 19 раз по сравнению с расчетами на ЦПУ.

Таким образом, мы продемонстрировали некоторые эффективные применения итерационных методов подпространств Крылова с предобуславливанием на графических процессорах NVIDIA с помощью CUDA.

4. Заключение

В работе представлена библиотека итерационных методов подпространств Крылова с предобуславливанием для решения на современных графических процессорах NVIDIA разреженных СЛАУ общего вида с нерегулярной структурой, как симметричных, так и несимметричных. Библиотека итерационных методов предназначена для пользователей, которые хотят избежать трудностей и деталей параллельного программирования на графических процессорах, но которым приходиться иметь дело с большими разреженными СЛАУ и необходимо использовать эффективность параллельной архитектуры графических процессоров. Использование графических процессоров, а также построенных на их основе решений для высокопроизводительных вычислений, позволяют значительно повысить производительность расчетов при низкой себестоимости и низком энергопотреблении.

Литература

1. NVIDIA Corporation. NVIDIA CUDA Programming Guide, June 2008. Version 2.0.
2. Volkov V. and Demmel J. W. Benchmarking GPUs to tune dense linear algebra. In Proc. 2008 ACM/IEEE Conference on Supercomputing, November 2008.
3. Bell N., Garland M. Efficient Sparse Matrix Vector Multiplication on Cuda, NVIDIA Technical Report NVR-2008-004, December 11, 2008.
4. Baskaran M., Bordawekar R. Optimising sparse matrix-vector multiplication on GPUs. IBM Tech. Rep. 2009.
5. Barret R. et al. Temlates for the solution of linear systems: building blocks for iterative methods. Philadelphia: SIAM, 1994.
6. Buatois L., Cauman G., Levy B. “Concurrent Number Cruncher: An Efficient Sparse Linear Solver on GPU”. In High Performance Computation Conference (HPCC), Springer Lecture Notes in Computer Sciences, 2008
7. Чадов С.Н. Реализация алгоритма решения несимметричных систем линейных уравнений на графических процессорах. Вычислительные методы и программирование. 2009. Т.10. С. 321 -326.
8. Saad Y. Iterative methods for sparse linear systems. NY: PWS Publish. 1996
9. Verma S., Aziz Kh. A control volume scheme for flexible grids in reservoir simulation. Paper SPE 37999 presented at Reservoir Simulation Symposium, Dallas, Texas, 8-11 June, 1997.

Параллельный алгоритм для решения трехмерных уравнений Максвелла с разрывной диэлектрической проницаемостью

Т.З. Исмагилов

Предлагается конечно-объёмный метод для численного решения трехмерных уравнений Максвелла с разрывной диэлектрической проницаемостью на неструктурированных сетках. Метод имеет второй порядок аппроксимации по времени и пространству для разрыва диэлектрической проницаемости проходящего по произвольной гладкой поверхности. Численный алгоритм допускает параллельную реализацию с помощью геометрической декомпозиции для использования на многопроцессорных ЭВМ. Приведённые результаты тестовых расчётов подтверждают второй порядок предлагаемого метода и высокую эффективность параллельной реализации.

1. Введение

Уравнения Максвелла описывают эволюцию электромагнитного поля и его взаимодействие с зарядами и токами. Необходимость решения уравнений Максвелла обусловлена широким спектром практических приложений в таких актуальных областях как нанотехнологии. К сожалению, для большинства практических задач аналитических решений не существует. Этот факт привёл к появлению различных численных алгоритмов решения уравнений Максвелла. Наибольшее развитие из них к настощему времени получили конечно-разностные методы.

Начало интенсивному развитию конечно-разностных методов для решения уравнений Максвелла было положено в работе [1], где была предложена схема второго порядка аппроксимации по времени и пространству, основанная на введении смещённых сеток. В дальнейшем различные конечно-разностные алгоритмы успешно применялись для решения различных задач [2–4].

Однако для многих практических задач в областях со сложной геометрией более эффективными могут оказаться конечно-объёмные методы, которые позволяют использовать неструктурированные сетки. С их помощью можно более точно представить границы расчётной области, а также границы между подобластями с различными свойствами среды.

К настоящему времени было предложено несколько конечно-объёмных алгоритмов для решения уравнений Максвелла. В алгоритмах предложенных в работе [5] электрические и магнитные поля аппроксимируются на смещённых сетках как и в работе [1]. В алгоритмах рассмотренных в работах [6–8] все компоненты электромагнитного поля аппроксимируются в центрах ячеек.

С точки зрения приложений, важным является возможность распараллеливания численных алгоритмов, что позволяет применять их на многопроцессорных ЭВМ. Конечно-объёмные алгоритмы использующие неструктурированные сетки могут быть распараллелены с помощью геометрической декомпозиции. В работе [9] рассматривалась параллельная реализация конечно-объёмного алгоритма для решения двумерных уравнений Максвелла.

Одной из основных трудностей при построении схем второго порядка для уравнений Максвелла остаётся случай разрывных свойств среды. В работе [2] рассматривались различные способы сглаживания разрывной диэлектрической проницаемости. Но ни один из них не позволял сохранять порядок аппроксимации исходной схемы.

Для решения этой проблемы в работе [8] была предложена конечно-объёмная схема для решения двумерных уравнений Максвелла на треугольных сетках. Предложенная схема имела второй порядок аппроксимации и позволяла сохранить его даже для случая разрывной диэлектрической проницаемости. Однако, построенные алгоритмы позволяли прово-

дить расчёты только для случая, когда разрыв диэлектрической проницаемости проходил по координатной линии.

Очевидно, что необходимы более универсальные алгоритмы, позволяющие проводить расчёты для трёхмерных уравнений Максвелла в областях с разрывной диэлектрической проницаемостью где разрыв может проходить по произвольной гладкой поверхности. Такие алгоритмы должны допускать эффективную параллельную реализацию для использования на многопроцессорных ЭВМ.

В данной работе предлагается конечно-объёмная схема для численного решения трёхмерных уравнений Максвелла в областях с разрывной диэлектрической проницаемостью имеющая второй порядок аппроксимации по времени и пространству. Разрыв диэлектрической проницаемости может проходить по произвольной гладкой поверхности. Для получения второго порядка аппроксимации используется вычисление градиентов специально выбираемых компонент с помощью метода наименьших квадратов. Предлагаемая схема позволяет проводить расчёты на тетраэдральных сетках и допускает эффективную параллельную реализацию с помощью метода геометрической декомпозиции. В статье приводятся результаты расчётов на многопроцессорных ЭВМ которые подтверждают второй порядок точности предлагаемой схемы и высокую эффективность параллельной реализации.

2. Уравнения Максвелла

В отсутствие зарядов и токов система уравнений Максвелла в безразмерных переменных имеет следующий вид:

$$\frac{\partial \mathbf{D}}{\partial t} - \mathbf{rotH} = 0, \text{ где } \mathbf{D} = \varepsilon \mathbf{E}, \tag{1}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \mathbf{rotE} = 0, \text{ где } \mathbf{B} = \mu \mathbf{H}, \tag{2}$$

$$\mathbf{divD} = 0, \quad \mathbf{divB} = 0. \tag{3}$$

Здесь $\mathbf{E}$ — электрическое поле, $\mathbf{H}$ — магнитное поле, $\mathbf{D}$ — электрическая индукция, $\mathbf{B}$ — магнитная индукция, ε — диэлектрическая проницаемость, μ — магнитная проницаемость. Далее везде полагаем, что магнитная проницаемость $\mu = 1$. Данная система может быть представлена в векторной консервативной форме

$$\frac{\partial}{\partial t}\mathbf{U} + \frac{\partial}{\partial x_1}\mathbf{F}_1 + \frac{\partial}{\partial x_2}\mathbf{F}_2 + \frac{\partial}{\partial x_3}\mathbf{F}_3 = 0, \tag{4}$$

где $\mathbf{U}$ — вектор консервативных переменных, $\mathbf{F}_1$, $\mathbf{F}_2$ и $\mathbf{F}_3$ — векторы потоков

$$\mathbf{U} = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix}, \ \mathbf{F}_1 = \begin{pmatrix} 0 \\ H_3 \\ -H_2 \\ 0 \\ -E_3 \\ E_2 \end{pmatrix}, \ \mathbf{F}_2 = \begin{pmatrix} -H_3 \\ 0 \\ H_1 \\ E_3 \\ 0 \\ -E_1 \end{pmatrix}, \ \mathbf{F}_3 = \begin{pmatrix} H_2 \\ -H_1 \\ 0 \\ -E_2 \\ E_1 \\ 0 \end{pmatrix}. \tag{5}$$

Проинтегрировав уравнение (4) по объёму Ω с границей $\partial\Omega$ можно получить эквивалентную интегральную форму

$$\frac{\partial}{\partial t}\int\limits_{\Omega}\mathbf{U}d\Omega+\int\limits_{\partial\Omega}\left(n_1\mathbf{F}_1+n_2\mathbf{F}_2+n_3\mathbf{F}_3\right)dS=0, \tag{6}$$

где $\mathbf{n}=(n_1,n_2,n_3)$ — внешняя нормаль. Систему (4) также можно записать в недивергентной форме

$$\frac{\partial}{\partial t}\mathbf{V}+A_1\frac{\partial}{\partial x_1}\mathbf{V}+A_2\frac{\partial}{\partial x_2}\mathbf{V}+A_3\frac{\partial}{\partial x_3}\mathbf{V}=0, \tag{7}$$

где $\mathbf{V}$ вектор потоковых переменных

$$\mathbf{V}=\begin{pmatrix}E_1\\E_2\\E_3\\H_1\\H_2\\H_3\end{pmatrix}, \tag{8}$$

связаный с вектором консервативных переменных матрицей перехода Θ: $\mathbf{V}=\Theta\mathbf{U}$

$$\Theta=\begin{pmatrix}\frac{1}{\varepsilon}&0&0&0&0&0\\0&\frac{1}{\varepsilon}&0&0&0&0\\0&0&\frac{1}{\varepsilon}&0&0&0\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{pmatrix}, \tag{9}$$

а матрицы A_1, A_2, A_3 записываются как

$$A_1=\begin{pmatrix}0&0&0&0&0&0\\0&0&0&0&0&\frac{1}{\varepsilon}\\0&0&0&0&-\frac{1}{\varepsilon}&0\\0&0&0&0&0&0\\0&0&-1&0&0&0\\0&1&0&0&0&0\end{pmatrix}, \tag{10}$$

$$A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -\frac{1}{\varepsilon} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{\varepsilon} & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A_3 = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{\varepsilon} & 0 \\ 0 & 0 & 0 & -\frac{1}{\varepsilon} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{11}$$

3. Разностная схема

Рассмотрим расчётную область в трёхмерном пространстве. Будем считать что в ней построена сетка из тетраэдров Δ. Если две различные ячейки сетки соприкасаются, то они имеют общую грань, общее ребро или общую вершину. Для каждой ячейки Δ определим объём Ω_Δ и барицентр $\mathbf{X}^B_\Delta$ как

$$\Omega_\Delta = \int\limits_\Delta d\Omega, \quad \mathbf{X}^B_\Delta = \frac{1}{\Omega_\Delta}\int\limits_\Delta \mathbf{X} d\Omega. \tag{12}$$

Для каждой грани Γ определим площадь S_Γ и центр $\mathbf{X}^C_\Gamma$ как

$$S_\Gamma = \int\limits_\Gamma dS, \quad \mathbf{X}^C_\Gamma = \frac{1}{S_\Gamma}\int\limits_\Gamma \mathbf{X} dS. \tag{13}$$

Для приближенного решения уравнения (6) рассмотрим разностную схему

$$\Omega_{\Delta_i}\frac{\mathbf{U}^{n+1}_i - \mathbf{U}^n_i}{\tau} + \sum_{k=1}^{m}\int\limits_{\Gamma_k}\left(n_1\mathbf{F}_1 + n_2\mathbf{F}_2 + n_3\mathbf{F}_3\right)d\Gamma = 0. \tag{14}$$

где $\mathbf{U}^n$ аппроксимация значения $\mathbf{U}$ в барицентре i-ой ячейки $\mathbf{X}^B$ в момент времени $t_n = n\tau$, τ — шаг по времени, а Ω_{Δ_i} объём i-ой ячейки.

Для того чтобы в (14) найти значение $\mathbf{U}$ на новом временном слое — $\mathbf{U}^{n+1}_i$, надо вычислить интегралы по граням Γ_k ячейки Δ_i которые представляют собой потоки искомых величин через грани. Предполагаем, что в ячейке искомые функции изменяются линейно, по значениям этих функций в барицентре ячейки и по вычисленным градиентам функций находим значения функций в центре грани со стороны i-ой ячейки в момент времени $n\tau + \tau/2$. Аналогичным образом находим значения функций в центре грани со стороны ячейки, находящейся по другую сторону грани. По значениям функций по разные стороны грани, вообще говоря различным, находим потоки в центре грани в момент времени $n\tau + \tau/2$. Тогда интегралы в (14) приближенно вычисляем по формуле прямоугольников и получаем.

$$\mathbf{U}^{n+1}_i = \mathbf{U}^n_i - \frac{\tau}{\Omega_{\Delta_i}}\sum_{k=1}^{m} s^k_{\Delta_i}\mathbf{F}^k_i, \tag{15}$$

$s^k_{\Delta_i}$ — площадь k-ой грани $\mathbf{F}^k_i$— поток через k-ую грань.

3.1. Нахождение потоков через границу ячейки

Пусть $\mathbf{n} = (n_1, n_2, n_3)$ — вектор нормали к общей грани ячеек Δ_L и Δ_R в точке $\mathbf{X}^C$. Если предположить что производные компонент электромагнитных полей по касательной

к грани равны 0, то систему в недивергентной форме (7) можно переписать как

$$\frac{\partial}{\partial t}\mathbf{V} + A\frac{\partial}{\partial n}\mathbf{V} = 0, \tag{16}$$

где $A = A_1 n_1 + A_2 n_2 + A_3 n_3$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{\varepsilon}n_3 & -\frac{1}{\varepsilon}n_2 \\ 0 & 0 & 0 & -\frac{1}{\varepsilon}n_3 & 0 & \frac{1}{\varepsilon}n_1 \\ 0 & 0 & 0 & \frac{1}{\varepsilon}n_2 & -\frac{1}{\varepsilon}n_1 & 0 \\ 0 & n_3 & n_2 & 0 & 0 & 0 \\ n_3 & 0 & -n_1 & 0 & 0 & 0 \\ -n_2 & n_1 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{17}$$

Для этой системы можно рассмотреть одномерную задачу Римана где в качестве начальных значений принять аппроксимацию компонент электромагнитных полей в центре грани со стороны ячеек Δ_L и Δ_R — $\mathbf{V}_L\left(\mathbf{X}^C\right)$ и $\mathbf{V}_R\left(\mathbf{X}^C\right)$. Решение этой задачи [7, 8] будем использовать для вычисления потока через границу ячейки.

Обозначим $D = R^{-1}AR$, здесь D — диагональная матрица из собственных значений матрицы A, $D^{\pm}$ — диагональные матрицы, полученные из D заменой всех отрицательных (положительных) собственных чисел нулями, R — матрица, столбцы которой являются правыми собственными векторами матрицы A. Тогда $A = RDR^{-1} = RD^{+}R^{-1} + RD^{-}R^{-1} = A^{+} + A^{-}$. Учитывая $\mathbf{V} = \Theta\mathbf{U}$ для вычисления потоков через грань в (7) будем использовать

$$\mathbf{F} = \Theta^{-1}(A^{+}\mathbf{V}_L\left(\mathbf{X}^C\right) + A^{-}\mathbf{V}_R\left(\mathbf{X}^C\right)) \tag{18}$$

или

$$\mathbf{F} = C^{+}\mathbf{V}_L\left(\mathbf{X}^C\right) + C^{-}\mathbf{V}_R\left(\mathbf{X}^C\right) \tag{19}$$

где используя обозначения $a = \frac{1}{\sqrt{\varepsilon}}$, $b = \sqrt{\varepsilon}$, $\varepsilon = 2\varepsilon_L\varepsilon_R/\left(\varepsilon_L + \varepsilon_R\right)$

$$C^{+} = \frac{1}{2}\begin{pmatrix} a(1-n_1^2) & -an_1n_2 & -an_1n_3 & 0 & n_3 & -n_2 \\ -an_1n_2 & a(1-n_2^2) & -an_2n_3 & -n_3 & 0 & n_1 \\ -an_1n_3 & -an_2n_3 & a(1-n_3^2) & n_2 & -n_1 & 0 \\ 0 & -n_3 & n_2 & b(1-n_1^2) & -bn_1n_2 & -bn_1n_3 \\ n_3 & 0 & -n_1 & -bn_1n_2 & b(1-n_2^2) & -bn_2n_3 \\ -n_2 & n_1 & 0 & -bn_1n_3 & -bn_2n_3 & b(1-n_3^2) \end{pmatrix}, \tag{20}$$

$$C^{-} = \frac{1}{2}\begin{pmatrix} a(n_1^2-1) & an_1n_2 & an_1n_3 & 0 & n_3 & -n_2 \\ an_1n_2 & a(n_2^2-1) & an_2n_3 & -n_3 & 0 & n_1 \\ an_1n_3 & an_2n_3 & a(n_3^2-1) & n_2 & -n_1 & 0 \\ 0 & -n_3 & n_2 & b(n_1^2-1) & bn_1n_2 & bn_1n_3 \\ n_3 & 0 & -n_1 & bn_1n_2 & b(n_2^2-1) & bn_2n_3 \\ -n_2 & n_1 & 0 & bn_1n_3 & bn_2n_3 & b(n_3^2-1) \end{pmatrix}. \tag{21}$$

3.2. Нахождение значений компонент электромагнитных полей на границе ячейки

Рассмотрим исходную систему уравнений в недивергентной форме (7). Пусть $\mathbf{X}^B$ — барицентр ячейки, а $\mathbf{X}^C$ — центр грани. Тогда $\mathbf{V}(\mathbf{X}^C)$ может быть найдена по следующей формуле со вторым порядком по времени и пространству:

$$\begin{aligned}\mathbf{V}(\mathbf{X}^C) &= \mathbf{V}(\mathbf{X}^B) + \frac{\partial \mathbf{V}}{\partial \mathbf{x}}(\mathbf{X}^B)(\mathbf{X}^C - \mathbf{X}^B) - \\ &\frac{\tau}{2}\left(A_1 \frac{\partial \mathbf{V}}{\partial x_1}(\mathbf{X}^B) + A_2 \frac{\partial \mathbf{V}}{\partial x_2}(\mathbf{X}^B) + A_3 \frac{\partial \mathbf{V}}{\partial x_3}(\mathbf{X}^B))\right).\end{aligned} \tag{22}$$

Таким образом значения электромагнитных полей на грани ячейки сетки в (19) выражаются следующим образом

$$\begin{aligned}\mathbf{V}_L(\mathbf{X}^C) &= \mathbf{V}(\mathbf{X}_L^B) + \frac{\partial \mathbf{V}}{\partial \mathbf{x}}(\mathbf{X}_L^B)(\mathbf{X}^C - \mathbf{X}_L^B) - \\ &\frac{\tau}{2}\left(A_1 \frac{\partial \mathbf{V}}{\partial x_1}(\mathbf{X}_L^B) + A_2 \frac{\partial \mathbf{V}}{\partial x_2}(\mathbf{X}_L^B) + A_3 \frac{\partial \mathbf{V}}{\partial x_3}(\mathbf{X}_L^B)\right),\end{aligned}$$

$$\begin{aligned}\mathbf{V}_R(\mathbf{X}^C) &= \mathbf{V}(\mathbf{X}_R^B) + \frac{\partial \mathbf{V}}{\partial \mathbf{x}}(\mathbf{X}_R^B)(\mathbf{X}^C - \mathbf{X}_R^B) - \\ &\frac{\tau}{2}\left(A_1 \frac{\partial \mathbf{V}}{\partial x_1}(\mathbf{X}_R^B) + A_2 \frac{\partial \mathbf{V}}{\partial x_2}(\mathbf{X}_R^B) + A_3 \frac{\partial \mathbf{V}}{\partial x_3}(\mathbf{X}_R^B)\right).\end{aligned}$$

3.3. Нахождение градиентов компонент электромагнитных полей в ячейке

Вычисление градиентов проведём с использованием вектора непрерывных переменных. На границе разрыва диэлектрической проницаемости такими переменными будут нормальная компонента вектора электрической индукции, касательные компоненты вектора электрического поля и декартовы компоненты вектора магнитного поля. Будем считать что сетка построена таким образом что разрыв диэлектрической проницаемости проходит по граням ячеек. В каждой ячейке сетки введём свой вектор переменных $\mathbf{W}$. В ячейках имеющих общую грань с поверхностью разрыва диэлектрической проницаемости выберем

$$\mathbf{W} = \Xi(\theta, \phi, \varepsilon)\mathbf{V} = \begin{pmatrix} \varepsilon\sin(\theta)\cos(\phi) & \varepsilon\sin(\theta)\sin(\phi) & \varepsilon\cos(\theta) & 0 & 0 & 0 \\ \cos(\theta)\cos(\phi) & \cos(\theta)\sin(\phi) & -\sin(\theta) & 0 & 0 & 0 \\ -\sin(\phi) & \cos(\phi) & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{V}, \tag{23}$$

где θ и ϕ — углы вектора нормали к поверхности разрыва в сферической системе координат, а ε диэлектрическая проницаемость в ячейке. В остальных ячейках выберем $\mathbf{W} = \mathbf{V}$. В узлах сетки введём углы θ и ϕ. В узлах на поверхности разрыва диэлектрической проницаемости выберем их как углы вектора нормали к поверхности разрыва в сферической системе координат. В остальных узлах углы можно выбрать произвольно.

Сначала найдём градиенты $\mathbf{W}$ в ячейках с помощью метода наименьших квадратов. Для этого воспользуемся значениями $\mathbf{V}$ в барицентре самой ячейки и барицентрах соседних ячеек. Может получится так, что у ячейки отсутствуют несколько соседних ячеек. В

таком случае вместо значений в их барицентрах возьмём значения в барицентрах ячеек соседних для одной их присутствующих соседних ячеек. Таким образом в методе наименьших квадратов будем использовать

$$\left\{(\mathbf{X}^{B_j}, \Xi(\theta_i, \phi_i, \varepsilon_j)\mathbf{V}_j)\right\}, \tag{24}$$

где j пробегает индексы ячеек которые используются для вычисления градиентов в ячейке i. После нахождения градиентов в i-ой ячейке сетки по найденным градиентам вычислим значения $\mathbf{W}_i\left(\mathbf{X}^P\right) = \mathbf{W}_i^P$ в её узлах $\mathbf{X}^P = \left\{x_l^P\right\}$

$$\begin{aligned}\mathbf{W}_i\left(\mathbf{X}^P\right) &= \Xi(\theta^P, \phi^P, \varepsilon_i)\Xi^{-1}(\theta_i, \phi_i, \varepsilon_i)\left(\mathbf{W}_i + \frac{\partial}{\partial x_1}\mathbf{W}_i(\mathbf{X}^{B_j})\left(x_1^P - x_1^{B_i}\right) + \right. \\ &\left. \frac{\partial}{\partial x_2}\mathbf{W}_i(\mathbf{X}^{B_j})\left(x_2^P - x_2^{B_i}\right) + \frac{\partial}{\partial x_3}\mathbf{W}_i(\mathbf{X}^{B_j})\left(x_3^P - x_3^{B_i}\right)\right),\end{aligned} \tag{25}$$

где θ^P и ϕ^P значения углов в узле P. В одном узле $\mathbf{X}^P$ получаем несколько различных значений $\mathbf{W}_i\left(\mathbf{X}^P\right)$ по числу соседних ячеек, за итоговое значение $\mathbf{W}^P = \mathbf{W}\left(\mathbf{X}^P\right)$ принимаем их среднее арифметическое.

Теперь в каждой ячейке найдём градиенты $\mathbf{V}$ с помощью метода наименьших квадратов. Для этого будем использовать значения $\mathbf{V}$ в вершинах ячейки. Таким образом в методе наименьших квадратов для вычисления градиентов $\mathbf{V}$ в ячейке i возьмём

$$\left\{(\mathbf{X}^{P^k}, \Xi^{-1}(\theta^{P^k}, \phi^{P^k}, \varepsilon_i)\mathbf{W}^{P^k})\right\}, \tag{26}$$

где P^k, $k = 1, 2, ...$ номера вершин ячейки i.

4. Параллельная реализация

Комплекс программ для параллельной реализации предложенного алгоритма с помощью геометрической декомпозиции включает программу для декомпозиции сеток и вычислительную параллельную программу. Программа для декомпозиции сеток позволяет разбивать на фрагменты сетки построенные с помощью Gmsh [10], TETGEN [11] и NETGEN [12]. Каждый процесс вычислительной программы проводит расчёты на своём фрагменте. Для передачи данных между процессами используется интерфейс передачи сообщений MPI.

4.1. Декомпозиция сетки

Для проведения декомпозиции сетки достаточно задать количество фрагментов и алгоритм по которому определяется к какому фрагменту тетраэдр принадлежит. Если у двух фрагментов есть хотя бы одна общая вершина то они будут участвовать в обмене данными и для них программа создаст списки общих вершин и общих граней. Для каждого фрагмента программа выдаст файл в котором помимо исчерпывающей информации о сетке будет список соседних фрагментов и для каждого соседнего фрагмента список общих вершин и список общих граней.

4.2. Вычислительная программа

Основное отличие параллельной программы реализующей предложенный алгоритм от последовательной это необходимость обмена данными между процессами. Алгоритм требует обмена данных в вершинах сетки, в барицентрах тетраэдров и на сторонах граней. Обмен данных в вершинах сетки используется для получения суммы величин хранящихся в одной и той же вершине сетки в разных процессах. С его помощью вычисляются средние значения компонент электромагнитных полей в вершинах. Обмен данных в тетраэдрах

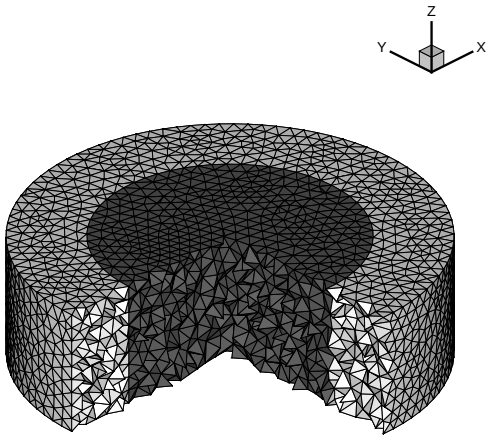


Рис. 1.

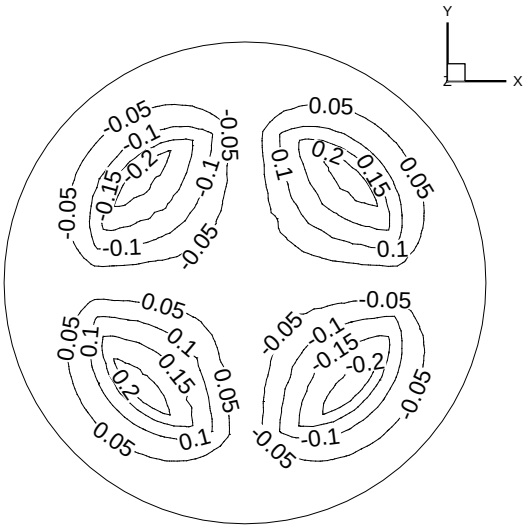


Рис. 2.

используется для передачи величин в фиктивные ячейки. Передаваемые величины включают компоненты электромагнитных полей и кооординаты барицентров тетраэдров. Обмен данных на сторонах граней используется для вычисления потоков. Особо отметим что алгоритм не требует передачи градиентов между процессами.

5. Результаты тестовых расчётов

Для проверки свойств предложенной схемы были проведены тестовые расчёты в которых использовались тетраэдральные сетки. Для построения тетраэдральных сеток использовался открытый программный продукт Gmsh [10]. Точность численного алгоритма оценивалась путём сравнения с аналитическими решениями. Ошибка численного решения в момент времени $t^n = n\tau$ вычислялась по формуле

$$\frac{\left\|\mathbf{V}^n\left(\mathbf{X}^B\right) - \mathbf{V}^{\mathbf{exact}}\left(\mathbf{X}^B, t^n\right)\right\|_{L_2}}{\left\|\mathbf{V}^{\mathbf{exact}}\left(\mathbf{X}^B, t^n\right)\right\|_{L_2}} = \sqrt{\frac{\sum\limits_{i=1}^{T}\left[\sum\limits_{k=1}^{6}\left(\mathbf{V}_k^n\left(\mathbf{X}^{B_i}\right) - \mathbf{V}_k^{\mathbf{exact}}\left(\mathbf{X}^{B_i}, t^n\right)\right)^2\right]\cdot S_{\Delta_i}}{\sum\limits_{i=1}^{T}\left[\sum\limits_{k=1}^{6}\left(\mathbf{V}_k^{\mathbf{exact}}\left(\mathbf{X}^{B_i}, t^n\right)\right)^2\right]\cdot S_{\Delta_i}}} \tag{27}$$

где T — общее количество тетраэдров в вычислительной области, $\mathbf{V}_k^n\left(\mathbf{X}^{B_i}\right)$ и $\mathbf{V}_k^{\mathbf{exact}}\left(\mathbf{X}^{B_i}, t^n\right)$ — вычисленные и точные значения электромагнитных полей в барицентре ячейки i, соответственно.

5.1. Тест 1

Рассмотрим распространение гибридной электромагнитной волны в световоде со ступенчатым профилем диэлектрической проницаемости. Разрыв диэлектрической проницаемости ε проходит по криволинейной поверхности $r \equiv \sqrt{x_1^2 + x_2^2} = a$

$$\varepsilon = \varepsilon(r) = \begin{cases} \varepsilon_1 = n_1^2, \text{ где } 0 \le r \le a, \\ \varepsilon_2 = n_2^2, \text{ где } r > a. \end{cases} \tag{28}$$

В этом случае система уравнений Максвелла имеет аналитическое решение [13] которое в цилиндрических координатах записывается в виде

$$
\begin{aligned}
0 \le r \le a: \quad & E_r = \beta \tfrac{a}{u} \left[\tfrac{1-s}{2} J_0\left(\tfrac{u}{a} r\right) - \tfrac{1+s}{2} J_2\left(\tfrac{u}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& E_\theta = -\beta \tfrac{a}{u} \left[\tfrac{1-s}{2} J_0\left(\tfrac{u}{a} r\right) + \tfrac{1+s}{2} J_2\left(\tfrac{u}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z), \\
& E_z = J_1\left(\tfrac{u}{a} r\right) \cdot \cos(\theta) \cos(kt - \beta z), \\
& H_r = k n_1^2 \tfrac{a}{u} \left[\tfrac{1-s_1}{2} J_0\left(\tfrac{u}{a} r\right) + \tfrac{1+s_1}{2} J_2\left(\tfrac{u}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z) \\
& H_\theta = k n_1^2 \tfrac{a}{u} \left[\tfrac{1-s_1}{2} J_0\left(\tfrac{u}{a} r\right) - \tfrac{1+s_1}{2} J_2\left(\tfrac{u}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& H_z = -\tfrac{\beta}{k} s J_1\left(\tfrac{u}{a} r\right) \cdot \sin(\theta) \cos(kt - \beta z),
\end{aligned}
\tag{29}
$$

$$
\begin{aligned}
r > a: \quad & E_r = \beta \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s}{2} K_0\left(\tfrac{w}{a} r\right) - \tfrac{1+s}{2} K_2\left(\tfrac{w}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& E_\theta = -\beta \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s}{2} K_0\left(\tfrac{w}{a} r\right) + \tfrac{1+s}{2} K_2\left(\tfrac{w}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z), \\
& E_z = \tfrac{J_1(u)}{K_1(w)} K_1\left(\tfrac{w}{a} r\right) \cdot \cos(\theta) \cos(kt - \beta z), \\
& H_r = k n_0^2 \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s_0}{2} K_0\left(\tfrac{w}{a} r\right) + \tfrac{1+s_0}{2} K_2\left(\tfrac{w}{a} r\right) \right] \sin(\theta) \sin(kt - \beta z) \\
& H_\theta = k n_0^2 \tfrac{a}{w} \tfrac{J_1(u)}{K_1(w)} \left[\tfrac{1-s_0}{2} K_0\left(\tfrac{u}{a} r\right) - \tfrac{1+s_0}{2} K_2\left(\tfrac{w}{a} r\right) \right] \cos(\theta) \sin(kt - \beta z), \\
& H_z = -\tfrac{\beta}{k} s \tfrac{J_1(u)}{K_1(w)} K_1\left(\tfrac{w}{a} r\right) \cdot \sin(\theta) \cos(kt - \beta z),
\end{aligned}
\tag{30}
$$

где J_0 и J_1 функции Бесселя первого рода, K_0 и K_1 функции Бесселя второго рода, $s_0 = s\beta^2/k^2 n_0^2$, $s_1 = s\beta^2/k^2 n_1^2$, $u = a\sqrt{k^2 n_1^2 - \beta^2}$, $w = a\sqrt{k^2 n_2^2 - \beta^2}$,

$$
s = 2 \left(\frac{1}{u^2} + \frac{1}{w^2} \right) \left[\frac{J_0(u) - J_2(u)}{u J_1(u)} - \frac{K_0(w) - K_2(w)}{w K_1(w)} \right]^{-1}, \tag{31}
$$

а β находится из дисперсионного соотношения

$$
\left[\frac{J_1'(u)}{u J_1(u)} + \frac{K_1'(w)}{w K_1(w)} \right] \left[\frac{J_1'(u)}{u J_1(u)} + \frac{n_0^2}{n_1^2} \frac{K_1'(w)}{w K_1(w)} \right] = \left(\frac{1}{u^2} + \frac{1}{w^2} \right) \left[\frac{1}{u^2} + \frac{n_0^2}{n_1^2} \frac{1}{w^2} \right]. \tag{32}
$$

Тестовые константы $\varepsilon_1 = 2.25$, $\varepsilon_2 = 1.0$, $k = 6.0$, $a = 0.64$, $\beta = 8.402440923258$, $u = 2.063837416842$, $w = 3.764648073438$. В качестве расчётной области был выбран цилиндр радиуса 1.0 и высотой 0.6:

$$
\sqrt{x_1^2 + x_2^2} \le 1.0, \quad 0 \le x_3 \le 0.6. \tag{33}
$$

Расчёты проводились на последовательности сеток состоящих из 18845, 51147, 144048, 405279 и 1250790 тетраэдров. Сетки строились таким образом, чтобы разрыв диэлектрической проницаемости проходил по граням тетраэдров. Шаг по времени выбирался пропорциональным линейным размерам тетраэдров. На **Рис. 1** показан пример расчётной сетки состоящей из 18845 тетраэдров. Не показаны тетраэдры у которых первые две координаты барицентра меньше 0.5. Подобласть с более высокой диэлектрической проницаемостью $\varepsilon_1 = 2.25$ обозначена тёмным цветом. На **Рис. 2** показаны изолинии распределения третьей компоненты магнитного поля H_3 в момент времени $T = 4.73$ в сечении $x_3 = 0.3$ полученные на сетке из 144048 тетраэдров. На **Рис. 3** показана эволюция ошибки δ_2 в норме L_2 на последовательности из пяти сеток. В таблице **1** приводятся максимальные значения ошибки в норме L_2. Как следует из результатов расчётов, алгоритм позволяет проводить расчёты со вторым порядком точности в областях с разрывной диэлектрической проницаемостью в случае, когда разрыв проходит по криволинейной поверхности.

Таблица 1.

Число тетраэдров	δ_2	Порядок аппроксимации
18845	0.055064	
51147	0.027107	2.12
144048	0.013602	2.06
405279	0.006120	2.14
1250790	0.002880	2.11

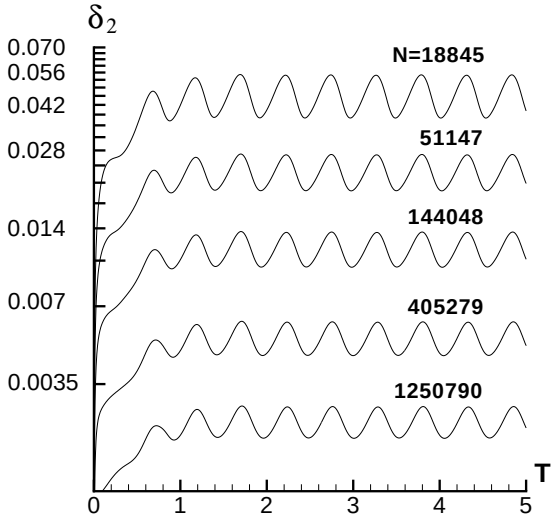


Рис. 3.

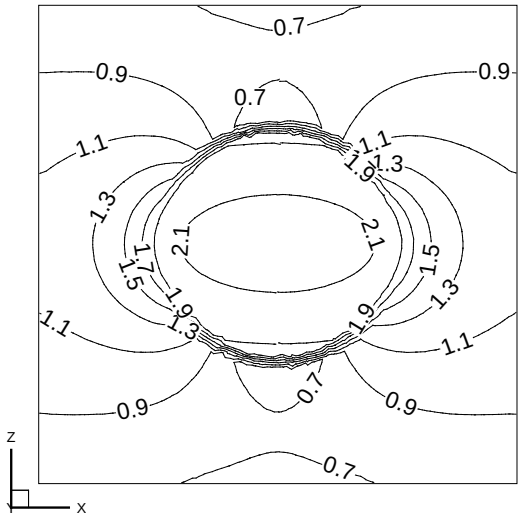


Рис. 4.

5.2. Тест 2

В качестве второго теста рассмотрим задачу о взаимодействие плоской электромагнитной волны с диэлектрической сферой. Разрыв диэлектрической проницаемости ε проходит по криволинейной поверхности $r \equiv \sqrt{x_1^2 + x_2^2 + x_3^2} = a$.

$$\varepsilon = \varepsilon(r) = \begin{cases} \varepsilon_1 = n_1^2, \text{ где } 0 \le r \le a, \\ \varepsilon_2 = n_2^2, \text{ где } r > a. \end{cases} \tag{34}$$

Волна распространяется в направлении x_3 и имеет компоненты электромагнитных полей E_1 и H_2. Для этой задачи система уравнений Максвелла имеет аналитическое решение [14] но в силу его громоздкости приводить его здесь не будем.

В качестве расчётной области выбирался куб. Расчёты проводились на сетке состоящей из 1426139 тетраэдров. Сетка была построена таким образом чтобы разрыв диэлектрической проницаемости проходил по граням тетраэдров.

Расчёты проводились с использованием последовательной и параллельной реализации предложенного алгоритма. Расчёты проводились на кластере Новосибирского Государственного Университета. Кластер построен на базе блэйд-серверов HP BL460c, имеющих

по 16 ГБ оперативной памяти и по два 4х-ядерных процессора Xeon 5355 (2.66 ГГц). В качестве коммуникационной среды использовался InfiniBand. В параллельной реализации использовалась декомпозиция расчётной области на 2,4,8,16,32,64,128 частей. Результаты всех расчётов с использованием параллельной версии и расчёта последовательной версии совпадали. На **Рис. 4** показаны изолинии распределения первой компоненты электрического поля E_3 в момент времени $T = 2.00$ в сечении $x_1 = 0.0$. Время счёта было различным. В таблице **2** приводятся затраты времени на проведения расчёта в зависимости от числа процессов. Видно что достигается ускорение близкое к линейному, что говорит о высокой эффективности параллельной реализации.

Таблица 2.

Число процессов	Время счёта сек.
1	90179
2	48512
4	25100
8	15243
16	7052
32	3333
64	1715
128	939

6. Заключение

В статье был предложен метод конечных объёмов для решения нестационарных уравнений Максвелла с разрывной диэлектрической проницаемостью на неструктурированных сетках. Проведённые тестовые расчёты подтверждают второй порядок аппроксимации по времени и пространству предлагаемой схемы а также эффективность параллельной реализации для систем с разделённой памятью с помощью интерфейса передачи сообщений MPI. В будущем планируется разработать параллельную реализацию для систем с общей памятью используя OpenMP. Необходимость в такой реализации возникнет всвязи с увеличением количества ядер в процессорах.

Литература

1. Yee K.S. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media // IEEE Trans. Antennas Propagat. 1966. V. 14 P. 585—589.

2. Taflove A. Advances in Computational Electrodynamics: the Finite-Difference Time-Domain Method // Boston, Artech House, 1998.

3. Taflove A. and Hagness S.C. Computational Electrodynamics: the Finite-Difference Time-Domain Method // Boston, Artech House, 2000.

4. Sullivan D.M. Electromagnetic Simulation Using the Finite-Difference Time-Domain Method // New York, IEEE, 2000.

5. Hermeline F. Two coupled particle-finite volume methods using Dalaunay-Voronoi meshes for applorimation of Vlasov—Poisson and Vlasov — Maxwell equations // J. Comput. Phys. 1993. V. 106 P. 1—18.

6. Cioni J.-P., Fzoui L., Issautier D. Higher order upwind schemes for solving time domain Maxwell equations // La Recherche Aĕrospatiale 1994. №. 5 P. 319—328.

7. Лебедев А.С., Федорук М.П., Штырина О.В. Решение нестационарных уравнений Максвелла для сред с неоднородными свойствами методом конечных объёмов // Вычисл. технологии. 2005. Т. 10, № 2. С. 60–73.

8. Лебедев А.С., Федорук М.П., Штырина О.В. Конечно-объёмный алгоритм решения нестационарных уравнений Максвелла на неструктурированной сетке // ЖВМ и МФ. 2006. Т. 47 №. 7. С. 1286—1301.

9. Cioni J.-P., Fzoui L., H. Steve A parallel time-domain Maxwell solver using upwind schemes and triangular meshes // IMPACT Comput. Sci. Eng Academic Press, Orlando, FL, USA 1994. V. 5 P. 215—247.

10. Geuzaine C. and Remacle J.-F. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities// Int. J. Numer. Meth. Engng. 2009. V. 3 P. 1–24.

11. Hang Si Adaptive tetrahedral mesh generation by constrained Delaunay refinement // Internat. J. Numer. Methods Engrg. 2008. V. 75 P. 856–880.

12. J. Schoberl. NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. // Comput.Visual.Sci. 1997. V.1 P. 41–52.

13. Okamoto K. Fundamentals of Optical Waveguides // London, Academic Press, 2000.

14. Bohren C.F. and Huffman D.R. Absorption and scattering of light by small particles // Wiley, 1998.

Технологии ГРИД в вычислительной химии

В.М.Волохов, Д.А.Варламов, А.В.Пивушков, Г.А.Покатович, Н.Ф.Сурков

Рассмотрены основные варианты применения ГРИД технологий в области вычислительной химии, а также основные достижения авторов в использовании подобных технологий на базе ресурсного ГРИД сайта ИПХФ. Описаны основные типы задач, использованные технологии, а также основные методы решения задач в распределенных средах, включая разработку прикладных программных интерфейсов различного уровня для запуска в распределённых вычислительных средах, методы формирования и запуска «пучков» независимых задач на распределенных ресурсах, создания пользовательских WWW интерфейсов, методы динамического формирования среды выполнения и т.п.

Введение

Вычислительная химия в своем современном состоянии невозможна без использования сверхмощных параллельных и распределенных вычислительных ресурсов, которые требуются для решения задач самых разных классов. Вычислительная и квантовая химия являются одними из наиболее заинтересованных в ГРИД вычислениях отраслей науки.

Следующие основные тематические научные направления наиболее заинтересованы в использовании ГРИД технологий:

- изучение строения вещества;
- строение молекул и структура твердых тел;
- создание материалов с заранее заданными свойствами;
- создание биологически активных веществ и лекарственных препаратов, биотехнологии;
- кинетика и механизм сложных химических реакций;
- химическая физика процессов горения и взрыва;
- газодинамика экстремальных состояний;
- химическая физика процессов образования и модификации полимеров;
- химическая физика биологических процессов и систем;
- предсказательное моделирование наноструктур;
- нанотехнологии;
- общие проблемы химической физики;

и многое другое…

Для проведения крупномасштабных вычислений в области вычислительной и квантовой химии и сопряженных областей науки требуется проведение высокоинтенсивных параллельных и распределенных расчетов. Например, некоторые задачи оптимизации молекулярных структур требуют выполнения до 10^9 отдельных расчетов. Подобные расчеты требуют вычислительных ресурсов, которые не может предоставить ни один из доступных вычислительных центров. Для таких задач необходимо развитие и применение ГРИД технологий в области вычислительной и квантовой химии для организации распределенных вычислений.

Крупномасштабные квантово-химические расчеты – одно из основных научных направлений работы вычислительного центра ИПХФ РАН [1,2]. Эти расчеты выполняются с использованием авторских программ, "open source" пакетов (GAMESS-US, CPMD, Dalton-2, NAMD, ABINIT и др.), а также лицензионных программ (Gaussian-98,-03, VASP, Mopac2002, MolPro). Институт располагает богатейшей в России библиотекой параллельных квантово-химических и молекулярно-динамических программ. Работы с системами распределенных вычислений в ИПХФ РАН были начаты в 2004-2008 годах по программам Президиума РАН и Федеральным целевым научно-техническим программам и продолжаются в настоящее время в рамках Программы фундаментальных исследований Президиума РАН № 1 на 2009-2011 годы «Проблемы создания национальной научной распределенной информационно-вычислительной среды на

основе развития ГРИД технологий и современных телекоммуникационных сетей», а также в рамках программы Союзного Государства «СКИФ-ГРИД».

Основными задачами авторов стало развитие двух основных направлений: 1) адаптация наиболее востребованного прикладного ПО в области вычислительной (прежде всего квантовой) химии к работе в инфраструктуре ГРИД и обеспечение широкого доступа пользователей к работе с ним с использованием самых различных методов и технологий; 2) развитие ресурсного ГРИД сайта (для нескольких распределенных сред), выступающего как в роли полигона для проведения вычислительных экспериментов в данной области, так и в роли средства для решения реальных фундаментальных и научно-практических задач. Выбор данных направлений был обусловлен основной стратегией развития инфраструктуры ГРИД как в России, так и в мире, и позволяет наилучшим образом «приблизить» конечного пользователя (прежде всего – ученого-химика) к широкомасштабному использованию распределенных вычислительных ресурсов и обеспечить возможность решения задач, принципиально трудно разрешимых в настоящее время на единичных вычислительных комплексах.

Основные типы и классы задач вычислительной химии

Почему же так важны квантово-химические расчеты и насколько велики могут быть востребованные ими вычислительные ресурсы? Подобные расчеты являются важнейшим звеном при проведении исследований в области строения вещества, наноматериалов, физики твердого тела, биофизики и всех научных дисциплин, связанных с исследованием электронной структуры вещества и его строения.

Наш многолетний опыт проведения подобных расчетов [1-4] позволяет разделить большинство квантово-химических задач на два основных вычислительных типа:

1) задачи, распадающиеся на совокупность практически независимых заданий, число которых зависит обычно от количества параметров задачи или от «сетки» разбиения искомой области данных;
2) задачи, представляющие собой единый вычислительный процесс, как правило, требующий единовременного выделения большого количества ресурсов (количество CPU, оперативная память на ядро, дисковые массивы).

Задачи первого типа наиболее применимы к работе в ГРИД средах, поскольку, распределяя независимые задания на множество небольших кластеров (каждый кластер – 10-20 процессоров, задание исполняется на нем как параллельное), можно добиться высокой эффективности использования вычислительных ресурсов. При этом возможно использование весьма больших вычислительных полигонов (до 10^3–10^4 процессоров) как в локальном варианте (в гетерогенных вычислительных средах типа Condor), так и в условиях распределенных сред (совокупность удаленных кластеров – ресурсных узлов). Хорошим примером является траекторные расчеты химических реакций. Характерной системой для подобных расчетов является реакция H_2 + O_2 (рис.1). Расчет представляет собой компьютерное моделирование с помощью классических траекторий элементарного акта столкновения. Как правило, расчет одной траектории занимает не более нескольких минут. Для расчета полного сечения подобной реакции для набора необходимой статистики следует разыграть: по два угла взаимной ориентации для каждой молекулы, начальные колебательные и вращательные квантовые числа, параметр столкновения, относительную энергию столкновения. Последовательный перебор указанных параметров приводит к необходимости расчета десятков миллионов траекторий, что приводит к нереальности решения подобных задач на локальных ресурсах и перехода к их решению на распределенных полигонах. Аналогичным способом могут решаться многочисленные многопараметрические задачи химии, для которых свойственен перебор многомерных «сеток» входных параметров, что ведет к увеличению числа независимых расчетов до 10^8 – 10^9.

Задачи второго типа представляют собой существенную проблему, т.к. эффективность их решения непосредственно связана с эффективностью распараллеливания вычислительного процесса и высокими требованиями к ресурсам узла. Например, для программы Gaussian известно эмпирическое правило: масштабируемость пропорциональна кубическому корню из числа процессоров. Для программы GAMESS масштабируемость существенно лучше и для нескольких десятков процессоров остается практически линейной. Аналогично возможно масштабирова-

ние программ типа VASP. Таким образом, для характерных задач исследования наноструктур и молекулярных кристаллов возможно использование до нескольких тысяч процессоров с потреблением процессорного времени порядка месяца, т.е. желательно использование кластеров терафлопного уровня. При этом возможно использование ГРИД сред для запуска подобных задач на удаленных ресурсах, при этом с точки зрения пользователя отличается от обычного удаленного запуска заданий (например, через SSH) в сторону упрощения выбора необходимого (и доступного!) вычислительного ресурса.

Характерные примеры задач первого и второго вычислительных классов приведены на рис.1 и 2.

Рис.1 Исследование траекторных расчетов сечений химических реакций (кислород + водород). Время расчета одной траектории от долей минуты до нескольких часов, общее время расчета – до нескольких лет работы единичного CPU

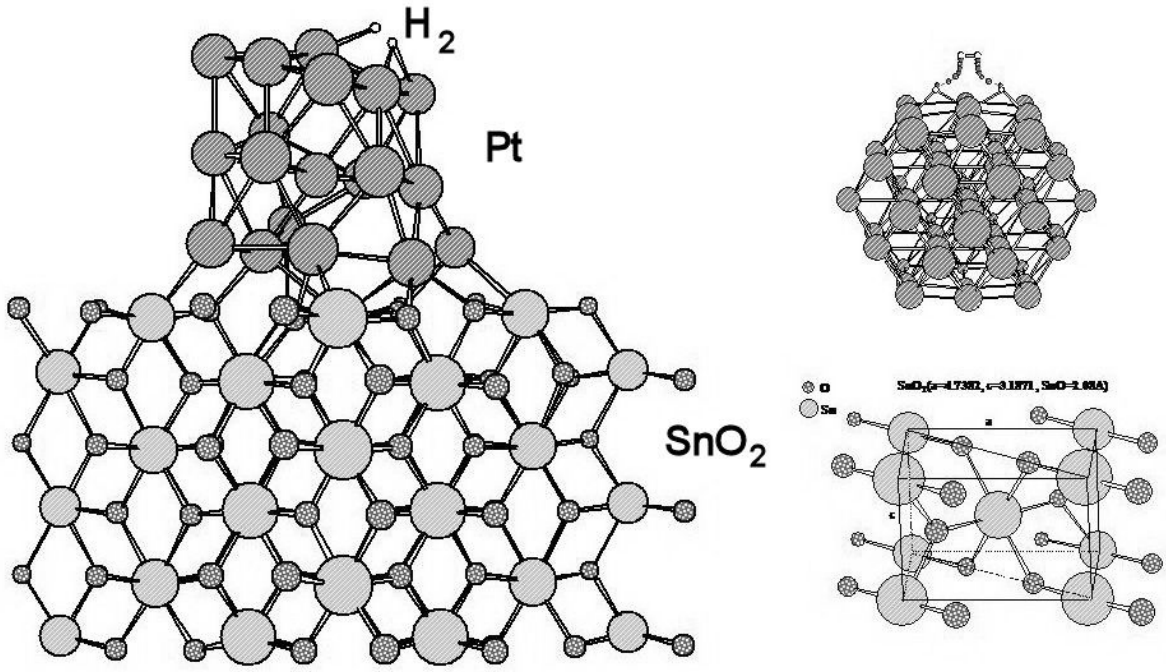


Рис.2 Адсорбция водорода (H_2) на октаэдрический кластер Pt_{19}, лежащий на поверхности SnO_2. При расчетах на СКИФ-МГУ «Чебышёв» (программа VASP) использовано до 200 CPU, затрачено15 часов, рассчитано всего 10 шагов оптимизации, при необходимых 100-200 шагах

С теоретической же точки зрения большинство задач вычислительной химии можно разбить на два класса:

1. Стационарные и важнейшие из них квантово-химические задачи и задачи на нахождение собственных функций и значений уравнения Шредингера (задача Штурма-Лиувилля),
2. Нестационарные задачи, исследующие временное поведение молекулярных систем или связанные с использованием траекторных методов расчета сечений реакций.

Первый класс достаточно обычен с точки зрения вычислений и, бо́льшей частью, относится ко второму типу вычислений. В среде ГРИД данные задачи могут запускаться как отдельные задания на удаленных узлах в зависимости от требуемых ресурсов (память, количество CPU и т.п.), а также наличия предустановленных необходимых прикладных пакетов квантово-химического ПО на этих узлах.

Второй же класс куда более интересен с точки зрения применимости ГРИД технологий. Нестационарные задачи в области теоретической химии и физики, безусловно, являются передним краем современной науки, и все исследования в этой области являются новыми. Как показывают последние конференции по современным методам в области вычислительной химии все ведущие научные центры, проводящие теоретические исследования химических превращений сфокусировали свое внимание на проводящихся в среде ГРИД вычислениях подобного типа. Задачи этого класса представляют собой передний край мировой науки. В настоящее время в крупнейших мировых научных центрах, специализирующихся в теоретической физике и теории химических превращений, ведутся активные работы по разработке вычислительных процедур в распределенных и параллельных вычислительных средах, позволяющих исследовать развитие химического превращения во времени. Кратко физическую суть этих работ можно описать так. Волновая функция химической системы в начальный момент времени представляется в виде разложения по полному набору функций, например, по функциям Гаусса. В результате создается пакет, во многом аналогичный волновой функции, строящейся в квантово-химической программе Gaussian. Однако, в отличие от Gaussian, где построение такой волновой функции и является конечным результатом, в нестационарном подходе возможно развитие ее во времени вдоль классических траекторий. При этом вдоль траекторий распространяются компоненты пакета с учетов интерференции между ними. Таким образом, строится волновая функция молекулярной системы в любой момент времени. Иными словами, компьютерные вычисления позволяют построить пропагатор, описывающий развитие квантово-химической системы во времени. Основная вычислительная процедура, которая используется для построения пропагатора – это метод Монте-Карло, идеально подходящий для распределенных ГРИД систем, т.к. опять же представляет собой большое количество независимых заданий.

Ресурсный центр ИПХФ РАН, структура и возможности

Для проведения всех работ с распределенными средами основой стал реализованный ранее на основе вычислительного комплекса ИПХФ РАН ресурсный ГРИД сайт, включающий узлы распределенных сред gLite и Unicore. Целью создания ресурсных узлов стало (помимо постоянного проведения текущих расчетов) формирование опытного полигона по проведению вычислительных экспериментов в российском ГРИД сегменте по вычислительной и квантовой химии. Основными задачами в рамках этого направления стали:

1) развитие инфраструктуры центра, его вычислительных мощностей и функциональности;
2) разработка и использование системы запуска **исходящих** задач (т.е. запускаемых пользователями на удаленных ресурсных узлах) различного типа в распределенных средах;
3) обеспечение проведения вычислительных экспериментов и расчета реальных **входящих** задач на собственном ресурсном узле ИПХФ (выступающем в роли удаленного распределенного ресурса и тестового полигона).

Существующий ресурсный сайт ИПХФ был достаточно детально описан авторами ранее [1,2,4], поэтому лишь коротко охарактеризуем его здесь. В настоящее время интегрированная

вычислительная мощность распределенной (локальной) вычислительной сети в Институте достигла 1,5 TFLOPS (пиковая) с совокупным дисковым пространством до 20 TB. Количество процессоров (ядер) основного вычислительного ресурса, кластера на процессорах Intel Xeon (с поддержкой 64-битных приложений), достигло 130, что позволяет апробировать работу программного обеспечения (системного и прикладного) в условиях работы достаточно больших вычислительных систем и предоставить значительную долю ресурсов для распределенных вычислений. Более детально собственно вычислительный комплекс ИПХФ описан здесь: http://cc-ipcp.icp.ac.ru.

В состав ресурсного ГРИД сайта входят:

1. ресурсный узел консорциума EGEE-RDIG (Enable GRID for E-sciencE и Russian Data Intensive GRID, http://www.egee-rdig.ru) на основе распределенной среды gLite (http://glite.web.cern.ch), работы ведутся в рамках виртуальной организации (ВО) RGSTEST;
2. ресурсный сайт категории «А» для работы в рамках созданного в России крупномасштабного вычислительного полигона СКИФ-Полигон (http://skif-grid.botik.ru) на базе промежуточного ПО Unicore (http://www.unicore.eu).
3. Для облегчения доступа пользователей сформирован WWW портал (http://grid.icp.ac.ru, Grid Enabled Chemical Physics – GECP), включающий высокоуровневые пользовательские WWW интерфейсы для работы с рядом задач (GAMESS, многопараметрические задачи) в условиях распределенных сред.

В состав ресурсного сайта входит комплекс интерфейсов различных уровней для взаимодействия прикладного ПО с распределенными ГРИД средами, позволяющий запускать целый ряд задач вычислительной химии на распределенных вычислительных ресурсах с возможностью формирования, запуска на удаленных ресурсах, мониторинга заданий и сбора результатов и статистики.

Работа в рамках ВО RGSTEST обеспечивает доступ к вычислительным мощностям порядка до 800 процессоров (без учета их возможной многоядерности) и дисковым массивам порядка 8-15 терабайт в нескольких географических зонах (Москва, Протвино, Харьков, Черноголовка и др.). Разнородность узлов данной ВО позволяет достаточно легко варьировать параметры запускаемых задач, ориентируясь на различные типы ресурсов. Использование подобного полигона обеспечивает проведение достаточно масштабных вычислительных экспериментов как научного, так и прикладного характеров.

Созданный ресурсный сайт для среды Unicore позволяет выполнять входящие задачи сертифицированных пользователей СКИФ-Полигона, производить мониторинг задач и передавать полученные результаты пользователям. Обеспечена возможность мониторинга состояния сайта извне. Настроен клиентский интерфейс, проведены успешные запуски исходящих задач через внешний брокер ресурсов на собственном ресурсном сайте ИПХФ в роли удаленного ресурса (https://unicorgw.icp.ac.ru:8080) и ресурсных узлах СКИФ-Полигона – ИПС РАН, Cyberia (Томский ГУ, https://cyberia.tsu.ru), СевКазГУ (https://skif-poligon.ncstu.ru), Нижегородского ГУ (https://85.143.3.48) и др., что продемонстрировало работоспособность как ресурсного сайта, так и клиентского интерфейса среды Unicore. Сайт позволяет проводить вычислительные эксперименты на полигонах, поддерживающих среду Unicore, с использованием входящих и исходящих распределенных задач вычислительной химии.

Составная часть ресурсного центра – ГРИД портал, объединение ГРИД и Web сервисов. Создан наиболее современный интерфейс, позволяющий более эффективно использовать все преимущества ГРИД, продуктивно работать, экономить время и ресурсы. Это среда, которая позволяет пользователям получить доступ к ГРИД ресурсам и сервисам, вызывать и настраивать их с помощью web-браузера. Портальный интерфейс также очень важен потому, что с помощью него пользователь даже начальной подготовки и уровня знаний может без особых проблем начать работу в ГРИД. Архитектура ГРИД портала основана на идее, что портальная система является контейнером для пользовательских интерфейсов (инструментов, клиентов), обеспечивающих работу с ГРИД службами. Преимущество данной архитектуры в том, что она достаточно легко позволяет встраивать в портал интерфейсы новых ГРИД служб и изменять существующие. Портальные сервисы контролируют и визуализируют пользовательский интерфейс.

В настоящее время Портал объединяет интерфейсы приложений двух видов :

1. Квантово-химический комплекс GAMESS для теоретического исследования свойств химических систем, *ab initio*, методы которого могут использовать параллельные вычисления;
2. Вычисление многопараметрических функций, под которой следует понимать целый класс нераспределенных задач химической физики, обладающих свойством параллелизма по данным (Data Parallel).

Данные интерфейсы позволяют определять входные параметры и условия (включая загрузку данных и конфигурационных файлов), формировать сложные первичные файлы запуска, производить (при условии сертификации пользователя) запуск данного ПО в распределенной среде, осуществлять мониторинг выполнения заданий и сбор результатов. Интегрирована также технология работы через web-интерфейс с «пучками» независимых заданий на «нарезаемых» областях данных. Заметим, что основная часть программного кода web-интерфейсов не связана напрямую с выбранной распределенной средой, поэтому они могут подключены к нескольким вариантам таковых сред. Данные интерфейсы значительно снижают трудоемкость работы пользователя в части формирования задач и работы с первичными данными. Они также значительно облегчают работу с пакетами в распределенных средах, особенно для неподготовленного пользователя.

Основные варианты использования ГРИД технологий в химии

Адаптация к работе в распределенных средах прикладных пакетов ПО вычислительной химии и авторских программ

Нами проводилась экспериментальная проверка и апробация возможности использования ГРИД ресурсов для реальных расчетов на стандартных прикладных пакетах программ (в том числе и параллельных), используемых в вычислительной химии, а также различных авторских программ, разработанных в ИПХФ и НЦЧ РАН. Особый интерес имеет адаптация этих программ для распределенных вычислений на максимуме доступных ресурсов российской и международной ГРИД инфраструктуры.

Для адаптации в распределенных вычислительных средах (см. выше) были выбраны следующие, наиболее востребованные пользователями ИПХФ прикладные программные пакеты:

- GAMESS-US (http://www.msg.ameslab.gov/GAMESS) - одна из самых популярных программ для теоретического исследования свойств химических систем, уступает по известности лишь комплексу Gaussian, позволяет рассчитывать энергию, структуры молекул, частоты их колебаний, а также разнообразные свойства молекул в газовой фазе и в растворе, как в основном, так и в возбужденных состояниях. Основное направление – развитие методов расчета сверхбольших молекулярных систем;
- VASP (Vienna University, http://cms.mpi.univie.ac.at/vasp) – Программный комплекс VASP предназначен для моделирования объема и поверхности твердых тел в рамках неэмпирических подходов, основанных на применении функционалов плотности с использованием периодических граничных условий с базисами на плоских волнах. VASP позволяет проводить оптимизацию структуры и выполнять моделирование в рамках молекулярной динамики. Программный комплекс VASP необходим для моделирования процессов на поверхности и в объеме твердых тел (прежде всего катализа и ионной проводимости).
- Gaussian-03 (http://www.gaussian.com/) - самое популярное средство выполнения квантово-химических расчетов среди основной массы химиков. Основные причины этого - широта охвата реализованных квантово-химических методик, высокая эффективность и удобный интерфейс пользователя. Современные версии комплекса программ Gaussian расширением спектра поддерживаемых квантово-химических методов и их модификаций. Комплекс программ Gaussian позволяет рассчитывать энергию, структуру молекул, частоты их колебаний, а также разнообразные свойства молекул в газовой фазе и в растворе, как в основном, так и в возбужденных состояниях. Основное направление, в котором развиваются версии, это развитие методов расчета сверхбольших молекулярных систем. Однако, использование пакета в распределенных средах затруднено лицензионными ограничениями.

- Dalton-2 (http://www.kjemi.uio.no/software/dalton/dalton.htm) – позволяет рассчитывать синглет-синглетные возбуждения, а также электронные структуры, вращательные и колебательные спектры молекул, учитывать релятивистские эффекты и эффект сольватации;
- CPMD (http://www.cpmd.org) – расчеты в области молекулярной динамики;
- NAMD (University of Illinois at Urbana-Champaign, Computational Biophysics Group, http://www.ks.uiuc.edu/Research/namd/) – хорошо масштабируемая молекулярно-динамическая программа. Одна из наиболее быстрых при параллельном вычислении на большом числе процессоров. Программа активно используется в ИПХФ РАН для расчетов мицеллы (micelle - коллоидная частица, несущая электрический заряд и объединяющая в себе несколько крупных молекул);
- Авторские программы (разработки ИПХФ), включающие многопараметрические задачи из области квантовой химии и молекулярной динамики.

Для всего выбранного ПО был проведен детальный анализ модульной структуры квантово-химического кода и изучены особенности работы различных реализаций однопроцессорных и параллельных версий, определены стратегии реализации выбранных типов квантово-химических вычислений применительно к распределенным средам.

Для большинства выбранных прикладных пакетов созданы и протестированы на реальных задачах низкоуровневые интерфейсы для запуска их в распределенных вычислительных средах (в основном для среды gLite, в меньшей степени – для среды Unicore). Данные интерфейсы включают набор скриптов по формированию исходящих заданий, запуску через брокер ресурсов на удаленных узлах, мониторингу выполнения задач, возвращению полученных результатов с удаленных ресурсов и «сборку» окончательных результатов на интерфейсе пользователя. Реализованы интерфейсы для однопроцессорных и параллельных (SMP, сокетные, MPI-1,2) вариантов указанного ПО. На ресурсном ГРИД узле ИПХФ, использованном в качестве удаленного распределенного ресурса, проведены запуски указанного прикладного ПО через инфраструктуры ВО RGSTEST (EGEE-RDIG) и СКИФ-Полигона. Запуски всего адаптированного ПО проводились в разных режимах и конфигурациях (с разным количеством востребованных процессоров и использованием разных вариантов параллельных расчетов). Были изучены варианты совмещения различных вариантов распараллеливания (например, SMP+MPI) вычислений применительно к некоторым прикладным пакетам (пакеты Dalton-2 и CPMD). После ряда вычислительных экспериментов была проведена коррекция созданных низкоуровневых интерфейсов и окончательная оптимизация их для распределенных сред. Были скорректированы проблемы запуска и работы параллельных (SMP, сокетные, MPI-1,2) вариантов указанного ПО на различных типах ресурсных узлов (разные пакетные системы PBS и параллельные среды).

Следует отметить, что большинство указанных прикладных пакетов вычислительной химии отличаются сложностью конфигураций и повышенными требованиями к среде выполнения, особенно для проведения параллельных расчетов. Обычно эта проблема решается путем создания *виртуальных организаций*, т.е. объединением через распределенные среды во многом однотипных (по установленному программному обеспечению и настройкам) вычислительных ресурсов. Для них выбранные прикладные пакеты (вместе со средствами конфигурирования и настройки) распространяются из единого репозитория (как, например, для прикладных пакетов ЦЕРНа – Atlas, CMC, Alice и т.п.). В большинстве же случаев неподготовленный ресурсный сайт не имеет нужного заранее установленного прикладного ПО или хотя бы не сконфигурирован должным образом, поэтому запуск непредустановленных сложных прикладных пакетов обычно для таких ресурсов оканчивается неудачей. Поэтому в общем случае необходима ручная или полуавтоматическая перенастройка ресурсных узлов распределенных сред, включающая установку собственно пакетов, конфигурирование центрального узла и расчетных узлов (настройка переменных окружения, общих NFS ресурсов, PBS очередей), установка дополнительных системных библиотек и исполняемых файлов (включая параллельные среды типа Mpich-2). При условии этого возможны запуски пакетов на распределенных узлах.

Для частичного решения данной проблемы авторами был разработан метод создания виртуальных перемещаемых программных «контейнеров». «Контейнер», включающий собственно прикладной пакет, набор необходимых системных файлов и библиотек, скрипты по развертыванию и настройке среды исполнения, файлы данных и конфигурационные файлы, доставляется на удаленный ресурсный узел ГРИД среды стандартными средствами распределенного

middleware. Применение таких «контейнеров» позволяет передавать заранее настроенную среду как единое задание, не требующее дополнительного конфигурирования и сложной процедуры установки и настройки, производимых, как правило, вручную администратором кластеров. «Контейнер» по прибытии на ресурсный узел производит развертывание пакета и необходимых системных библиотек, настройку среды исполнения (включая параллельную среду), запуск задания, по его окончании проводится отправка результатов на пользовательский интерфейс и «очистка» среды исполнения, т.е. приведение ресурса в первоначальное состояние. Так могут быть решены проблемы установки, настройки, несовместимости с операционной системой и другими программами, разрешаются конфликты одинаковых приложений. Более детально этот метод описан в статье авторов в этом же сборнике трудов (Варламов и др., «Виртуализация вычислительной среды в ГРИД»).

Работа с «пучками» формально независимых заданий

Для решения части задач «первого» вычислительного типа (например, широкого класса многопараметрических задач вычислительной химии) с использованием ГРИД технологий был создан метод запуска “пучков” независимых заданий для использования всех доступных ресурсов распределенной среды. Как уже говорилось, в области химической физики существует класс задач, требующих перебора большого количества параметров. При этом полная задача разбивается на огромное количество независимых подзадач (каждая определяется группой значений совокупности параметров). Задача автоматизации процесса разбиения полной задачи на фрагменты важна и определяет удобство пользования системой. Типичный пример - фундаментальная задача в теории элементарных химических процессов: туннельные реакции под воздействием электромагнитного излучения. Параметрами являются частота и амплитуда излучения. Задача имеет высокую вычислительную сложность, однако вычисления в каждой точке сетки в ней происходят независимо друг от друга, поэтому оказалось возможным разбить область вычислений на множество непересекающихся подобластей и для каждой из них запускать задачу на различных процессорах.

Была разработана методика запуска задач и получения результатов методом запуска “пучков” заданий на всех доступных ресурсах выбранной распределенной среды. На языке Perl написан комплекс программ для запуска “пучков” заданий и получения результатов счета с использованием пользовательских интерфейсов (UI) сред gLite и Unicore. Для решения многопараметрических задач квантовой химии были разработаны методы формирования «пучков» независимых заданий с варьирующими параметрами – до 10^4, в перспективе до 10^7 «атомарных» заданий на задачу. Для выбранных областей данных авторскими скриптами производится «нарезка» областей данных, формирование пулов независимых заданий, создание очередей запуска и отправки заданий на брокер ресурсов. После запуска периодически запускаемые (средствами ОС, например по cron) скрипты ведут мониторинг выполнения заданий, контроль таймаутов, перезапуск неудачных заданий и сбор результатов выполненных заданий (с использованием базы данных и таблиц в ней, контролирующих состояние заданий – «ожидание», «запуск», «выполнение» и т.д.). По окончании расчетов проводится сборка «атомарных» результатов в единый выходной файл. Для части задач (требующих значительного числа параллельных независимых расчетов) дополнительно созданы авторские механизмы по разбиению областей данных (или расчетов) на большие независимые подсетки или независимые задания, передачи всех их интерфейсам распределенных сред с последующим запуском на параллельных узлах и «сборки» финальных результатов из множества полученных независимых. Были сформированы и направлены на распределенные ресурсы ВО RGSTEST и СКИФ-полигона "пучки" заданий, осуществлен мониторинг их выполнения (с использованием комплекса скриптов и базы данных MySQL), "сборка" результатов с различных ресурсов. Для ВО RGSTEST было задействовано до 400 процессоров на различных ресурсных узлах (Москва, Протвино, Харьков, Черноголовка), для СКИФ-Полигона – доступные CPU узлов, указанных выше (до 120). С использованием данных методов был решен ряд реальных научных задач, включая: а) поведение низкотемпературных химических реакций под сильным электромагнитным воздействием; б) первичный расчет установок по росту кристаллов; в) ряд газодинамических задач.

Заключение

Авторами описаны использование некоторых технологий ГРИД вычислений применительно к приложениям вычислительной химии. Наши работы позволили создать в рамках технологий ГРИД вычислительную среду для проведения крупномасштабных расчетов в области вычислительной химии. Это позволило достигнуть нового уровня расчетов в области вычислительной химии:

- создан комплекс адаптированных к различным ГРИД средам (gLite, Unicore) прикладных программных пакетов вычислительной химии с интерфейсами различного уровня (от низкоуровневых интерфейсов вплоть до Web-портала),
- разработаны новые методики вычислений (методы формирования «пучков» независимых заданий, метод «виртуальных контейнеров» и т.д.) в распределенных и параллельных средах применительно к прикладному ПО вычислительной химии;
- создан ресурсный центр (включающий ресурсные узлы полигонов EGEE-RDIG и СКИФ-Полигона, а также web-портал) для проведения вычислительных экспериментов в этой предметной области, объединяющий как ресурсы для решения входящих заданий в средах gLite и Unicore, так и пользовательские интерфейсы к этим распределенным средам для решения исходящих задач;

В результате выполнения всего проекта создан вычислительный центр, позволяющий проводить масштабные расчеты в области вычислительной химии в распределенных средах на крупномасштабных полигонах (в перспективе до 10^4 CPU на узлах многотерафлопного масштаба). На ряде реальных задач продемонстрирована применимость созданных ресурсов для решения крупномасштабных химических задач на высокопроизводительных вычислительных полигонах. Это позволяет ставить и решать вычислительные задачи фундаментального и прикладного характера в области химических наук, ранее не доступные из-за ограниченности возможностей вычислительных ресурсов. Основные научные области применения – химическая физика, квантовая химия, исследование наноструктур, молекулярная динамика, фармацевтика, разработка топливных элементов и прочие близкие отрасли наук.

Литература

1. В.М.Волохов, Д.А.Варламов, А.В.Пивушков, Н.Ф.Сурков, Г.А.Покатович ГРИД и вычислительная химия // "Вычислительные методы и программирование", М.: МГУ, 2009, т.10, № 2, с.78-88
2. Варламов Д.А., Волохов В.М., Пивушков А.В., Сурков Н.Ф., Покатович Г.А. Распределенные и параллельные вычисления в области химии на ресурсном узле ГРИД ИПХФ РАН // “Distributed Computing and Grid-Technologies in Science and Education: Extended Proceedings of the 3rd Intern.Conf.” (Dubna, June 30-July 4, 2008). – Dubna: JINR, 2008, с.127-130
3. В.М. Волохов, Д.А. Варламов, А.В. Пивушков Крупномасштабные задачи химии на параллельных и распределенных вычислительных полигонах: современное состояние и перспективы // “Научный сервис в сети Интернет: решение больших задач», Всероссийская научная конференция, (г. Новороссийск, 22-27 сентября 2008) – М.; Изд-во МГУ, 468 с., с.210-212
4. С.М. Алдошин, В.М. Волохов, Д.А. Варламов, А.В. Пивушков Вычислительная химия в среде GRID: параллельные и распределенные вычисления // Вторая международная конференция «Суперкомпьютерные системы и их применение» SSA’2008, Минск, октябрь 2008; Минск, ОИПИ НАН Беларуси, с.114-118

Виртуализация вычислительной среды в ГРИД

Д.А.Варламов, Н.Ф.Сурков, В.М.Волохов, А.В.Пивушков

Данная статья посвящена описанию метода динамического формирования виртуальной среды выполнения для запуска сложно сконфигурированных прикладных пакетов вычислительной химии (на примере пакета GAMESS-US) в условиях параллельных сред на произвольных ГРИД ресурсах различных полигонов. Метод включает создание «виртуального контейнера» с образом среды исполнения, запуск его как исходящего ГРИД задания, развертывание на удаленном узле, настройку среды, исполнение задания, сбор результатов и «очистку» узла. Данный метод позволяет расширить возможности работы со сложными пакетами ПО в ГРИД средах.

Введение

Решение многих задач вычислительной химии, таких как моделирование молекулярных структур и их динамики, поведение и энергетика сложных химических реакций, наномоделирование невозможно без применения ГРИД вычислений. Например, время расчета поведения нанотрубок, легированных металлами, состоящих из $n*10^3$ атомов может достигать 3-4 лет для однопроцессорной системы. Детальнее особенности проведения квантово-химических расчетов в различных ГРИД средах описаны авторами ранее [1-4], а также в статье в данном сборнике трудов (Волохов и др., «Технологии ГРИД в вычислительной химии»)

На основе опыта работы в различных распределенных вычислительных средах авторами был сделан вывод, что наиболее существенными препятствиями на пути применения ГРИД технологий в вычислительной химии (а в общем случае – для любых сложно сконфигурированных прикладных пакетов ПО) стали следующие проблемы:

- разнородность доступных распределенных вычислительных ресурсов (на уровне архитектур процессоров, различных операционных систем, сетевых настроек, используемых параллельных сред и т.п.);
- необходимость создания для многих ресурсоемких параллельных приложений целой системы из конфигурационных настроек, дополнительных служб, специфичных параллельных сред, хранилищ данных и прочих компонентов информационно-вычислительной инфраструктуры;
- невозможность (или избыточная трудоемкость) перенастройки существующих вычислительных ресурсов (особенно класса “production farms”) для целей распределенных вычислений (что связано с особенностями операционных систем и используемого прикладного ПО, архитектуры, требований безопасности и т.д.) или под нужды конкретных прикладных пакетов.

Одним из способов решения данных проблем может стать применение интенсивно развиваемых в последнее время технологий виртуализации, включающих: (а) создание распределенных ресурсов и сервисов на базе виртуальных машин, (б) формирование виртуализованных «контейнеров-приложений» как единых распределенных задач; (в) создание полнофункциональных виртуальных машин, выступающих в роли исходящих/входящих распределенных заданий. Появление новых типов и архитектур процессоров (с поддержкой виртуализации на уровне ядра, интегрированных гипервизоров – на уровне материнских плат и т.п.), новых типов ПО уровня операционных систем и middleware со встроенными средствами виртуализации, развитие технологий быстрой передачи данных, удешевление Интернет-трафика дают основание предполагать, что технологии виртуализации займут ведущих мест в работе ГРИД ресурсов и инфраструктур.

Термин «виртуализация» используется авторами в двух основных смыслах: виртуализация вычислительных ГРИД ресурсов и сервисов и виртуализация вычислительного объекта (ОС, приложение и т.д.), перемещаемого в ГРИД среде. Потребность в виртуализации для распределенных вычислительных сред продиктована необходимостью создания и поддержки стандартных механизмов взаимодействия между пользователями и вычислительными ресурсами (серви-

сами), одинаковых со стороны ресурса (поставщика сервисов) и со стороны пользователя (вернее, используемого им интерфейса).

В 2009 году в рамках Программы фундаментальных исследований Президиума РАН № 1 на 2009-2011 годы «Проблемы создания национальной научной распределенной информационно-вычислительной среды на основе развития ГРИД технологий и современных телекоммуникационных сетей» авторами были продолжены исследования по применимости различных методов виртуализации для ГРИД сред. Среди прочих задач проекта была поставлена цель изучить и применить (на реальных расчетах в области химии) ряд технологий виртуализации, включая:

- создание и применение виртуальных машин на существующих ресурсных узлах различных распределенных сред с целью расширения их функциональности (решение различных прикладных задач на базе разных программных архитектур, разделение ресурсов, повышение безопасности, вычислительные эксперименты) и отработки устойчивости узлов;
- адаптация прикладного программного обеспечения для работы в роли приложений в составе виртуальных динамически формируемых параллельных сред;
- В перспективе – создание образов виртуальных машин (с встроенными прикладными пакетами сложных конфигураций) для запуска их как заданий на ГРИД ресурсах.

Основные результаты в области виртуализации ГРИД ресурсов и приложений были описаны авторами ранее [5,6], в данной же статье детально описан метод динамического формирования виртуальной среды исполнения на *не подготовленном* удаленном ГРИД ресурсе с использованием технологии «виртуального контейнера».

Анализ среды выполнения ГРИД заданий в разных средах

Сегодня для комфортной работы многих приложений даже на уровне локального кластера требуется создание целой системы из приложений, служб, сетей, хранилищ данных и прочих компонентов современной информационно-вычислительной инфраструктуры, которые зачастую плохо совместимы с режимами работы ресурсного узла в целом. Для ряда пакетов прикладного ПО и сервисов нужно создавать комплексные среды с необходимым набором приложений и политиками безопасности. Ключевыми требованиями являются скорость и простота предоставления таких сред, их тщательная изоляция друг от друга, квотирование вычислительных ресурсов для каждой среды, независимость от базовых настроек узла. Зачастую все это необходимо делать без прерывания работы узлов и остановки вычислительной среды, особенно в рамках «production farms», т.е. ресурсных узлов, не допускающих остановок и переконфигурирования системы.

Другой фундаментальной проблемой решения задач (особенно параллельных) в условиях распределенных вычислений является необходимость виртуализации программных сред для исходящих задач. Например, для проведения параллельных вычислений требуется наличие установленной на ресурсных узлах какой-либо системы параллельного программирования (например, MPI, OpenMP и др.) или предустановленных специфичных математических библиотек. Широко используемые в настоящее время пакеты прикладных программ (ППП) вычислительной химии (GAMESS, Gaussian, NAMD и др.), как, впрочем, и большинство инженерных пакетов, отличаются сложностью конфигураций и повышенными требованиями к среде выполнения, особенно для проведения параллельных расчетов. Они требуют обязательной настройки большого количества переменных окружения операционной системы до запуска параллельного приложения на каждом из использующихся процессоров. Такая настройка, как правило, осуществляется в два этапа:

(1) при ручной (или полуавтоматической) установке ПО системным администратором на каждом узле ресурсного сайта на уровне операционной системы (например, при формировании сайтов виртуальной организации, требующей единых настроек ПО);

(2) при настройке соответствующих скриптов запуска задания для каждого пользователя, согласно требованиям как приложения, так и системы параллельного программирования.

При этом традиционный подход со статическим линкованием необходимых библиотек (не говоря уже о динамическом варианте) к исполняемому модулю (пакету) часто не способен создать *полностью работоспособное* параллельное задание на произвольном ресурсе среды ГРИД,

поскольку на подобном ресурсе могут отсутствовать необходимые системные файлы.

Для решения данной проблемы авторами был проведен анализ процедуры исполнения типичного параллельного задания на ресурсном узле ГРИД (для сред gLite и Unicore), позволивший определить требования к создаваемому виртуальному образу среды исполнения, а также принципиальную возможность формирования динамической среды исполнения для тестируемых ресурсов. Также был сделан анализ систем параллельного программирования для выбора оптимального виртуального образа среды исполнения параллельного приложения на ГРИД ресурсах. В результате в качестве базового пакета для разработки виртуального образа среды исполнения параллельного приложения был выбрана среда Mpich-2.

В настоящее время в качестве распределенных ресурсов ГРИД используются, как правило, узлы в виде Linux-кластеров рабочих станций с операционной системой и некоторым набором приложений, настроенных для работы в среде ГРИД и специфичных для каждого из этих распределенных ресурсов. Единые стандарты на установку определенных типов программного обеспечения отсутствуют (если нет таковых в рамках какой-либо ВО). Поэтому на ресурсных узлах ГРИД среды, как правило, можно ожидать наличия только библиотек стандарта MPI-1. Добавим, что использование параллельных приложений в настоящее время в среде ГРИД ограничено как возможностями брокера ресурсов, который часто не распознает тип параллельного задания, так и отсутствием предустановленных на кластерах необходимых Run-time библиотек, а также отсутствием стандартов на размещение таких библиотек. Поэтому запуск параллельного сложно сконфигурированного задания на не указанном явно ресурсном узле распределенной среды обычно неэффективен, либо неудачен.

Проведенная в 2008-2009 годах работа была направлена на преодоление этих недостатков и преодоления специфичных особенностей среды ГРИД для запуска параллельных приложений.

Был проведен анализ предоставляемых псевдопользователям распределенных сред (так называемым «mapped users») прав доступа к ОС расчетного узла, которые определяют в свою очередь возможности работы внешнего задания с локальной файловой системой (обычно NFS), другими приложениями, исполняемыми модулями и утилитами. Для проведения такого анализа был самостоятельно разработан ряд оригинальных тестовых примеров, испытания которых позволили определить порядок запуска поступающих от брокера ресурсов заданий, в том числе:

- присвоенные заданиям имена пользователей и их права;
- создаваемые временные директории и требуемые файловые иерархии, в том числе на NFS ресурсах;
- доступность различных системных средств исполнения заданий (параллельные среды, доступ к очередям PBS и т.п.)

Это позволило определить требования к создаваемому виртуальному образу среды исполнения со стороны ОС ресурсного узла, а также принципиальную возможность динамической организации среды исполнения для тестируемых ресурсов.

Был сделан анализ имеющихся систем параллельного программирования для выбора оптимального виртуального образа среды исполнения параллельного приложения, в результате чего был выбран стандарт MPI (Message-Passing Interface, т.е. интерфейс для передачи сообщений), который предоставляет стандартные спецификации для библиотек передачи сообщений. В ИПХФ РАН длительное время используется реализация Mpich (свободно распространяемый Open Source Project, http://www.mcs.anl.gov/research/projects/mpich2), разработанная в Argonne National Laboratory.

Ранее на его основе в Суперкомпьютерном центре ИПХФ РАН для кластера на базе операционной системы ScientificLinux из исходных текстов была скомпилирована (после модификации участниками проекта) библиотека MPI. Основная причина использования менее распространенной версии стандарта MPI-2 (а не MPI-1) заключается в том, что использование стандарта MPI-1 не позволяет создать полностью однородную среду исполнения задания на узлах кластера, несмотря на ряд дополнительно разработанных сторонних программных пакетов. Стандартом MPI-2 однородная среда исполнения задания на всех процессорах рассматривается как базовая. Более того, с 2002 года стандарт MPI-1 не поддерживается Argonne National Laboratory, и была выпущена новая версия пакета Mpich, соответствующая стандарту MPI-2, в котором коренным образом изменена система запуска параллельных заданий и устранены указанные недостатки. Перед запуском задания осуществляется запуск кольца серверов mpd («mpd

ring»), одна из задач которых состоит в выравнивании среды окружения на главном и подчиненных узлах. В более общем случае, кольцо серверов может запускаться пользователем root, а остальные пользователи могут использовать это глобальное кольцо, однако, обычно на узлах ГРИД ресурса пользователю root запрещен любой удаленный доступ между узлами, и, следовательно, каждый псевдопользователь ГРИД должен запускать собственное локальное кольцо серверов (основанное на использовании непривилегированных портов).

Поэтому в качестве базового пакета для разработки виртуального образа среды исполнения параллельного приложения на распределенных ресурсах был выбран Mpich-2. Данный пакет представляет собой переносимую реализацию полных спецификаций MPI для широкого класса параллельных вычислительных средств, включающих кластеры рабочих станций и блоки массивно-параллельных процессоров (MPPs). Mpich-2 содержит, наряду с самими библиотеками MPI, программные средства для работы с программами MPI. Программные средства включают в себя переносимый стартовый механизм, несколько профилирующих библиотек для изучения производительности программ MPI.

При создании виртуального образа среды исполнения параллельного приложения на первых стадиях были введены ряд ограничений для ресурсных узлов:

- использована аппаратная архитектура x86, в настоящее время проводятся также работы с 64-битной версией;
- на расчетных узлах ГРИД ресурса используется операционная система Linux (клоны на базе RedHat – собственно RedHat, ScientificLinux, Fedora и т.п.), что связано с особенностями размещения системного ПО, хотя принципиальных ограничений на использование других ветвей Linux дистрибутивов нет;
- по стандарту настройки ресурсных узлов ГРИД для коммуникации между расчетными узлами используется интерфейс TCP/IP и беспарольный доступ по ssh (включая копирование файлов), возможна поддержка NFS ресурсов;
- Некоторые версии пакетов с целью повышения производительности вычислений имеют привязку к сетевым продуктам конкретных производителей и используют поставляемые этими производителями низкоуровневые драйверы. Нами пока такие версии, несмотря на их высокую эффективность, использоваться не будут.

В процессе тестирования было выбрано 2 потенциальных статических места инсталляции библиотек – по месту загрузки исполняемого приложения (т.е. в home директории mapped-user) и использование общедоступной на большинстве расчетных узлов директории /tmp. Также рассмотрена (для кластеров с бесдисковыми рабочими узлами) возможность использования общих NFS ресурсов, доступных (на запись) ГРИД пользователю.

Разработка системы динамического компилирования приложения на ресурсном узле и инсталляции дополнительных библиотек на данном этапе работ не рассматривалась.

Создание прототипа виртуального «контейнера» с образом среды исполнения параллельного приложения

После анализа процедуры выполнения ГРИД задания (в разных средах) и выбора среды параллельных вычислений была разработана технология создания динамически формируемых образов исполняемых сред, или виртуальных «контейнеров»

В соответствии с определенными выше требованиями ресурсных узлов был сформирован перемещаемый программный пакет MPI-2, тестирование которого на ГРИД кластере ИПХФ РАН с использованием тестовых примеров показало его полную работоспособность. Полученный пакет в дальнейшем использовался в качестве *базового* прототипа для разработки виртуального образа среды исполнения конкретных параллельных приложений, в том числе сложных прикладных пакетов.

В качестве первичного тестового приложения был использована программа вычисления числа π (‘cpi.c’) из пакета Mpich-2, правильность работы которой легко проверяется в параллельной среде с различным количеством узлов. Исходная тестовая программа была доработана с учетом особенностей запуска прикладных приложений на ГРИД узлах, был получен ее исполняемый модуль и скрипты запуска с использованием библиотек MPI-2. Тестовый модуль и

перемещаемый пакет MPI-2 были собраны и упакованы в единый «контейнер», для запуска которого в средах ГРИД (для сред gLite и Unicore) была разработана серия низкоуровневых скриптов пользовательского интерфейса.

Была принята следующая схема запуска: на удаленный ресурсный узел сети ГРИД через брокер ресурсов (или непосредственно – как, например, в Globus) передается главный скрипт и упакованный «контейнер», содержащий исполняемые файлы, необходимые системные библиотеки, файлы конфигурации и данных. Далее главный скрипт реализует следующую последовательность шагов по месту исполнения:

1. Сбор начальной информации о текущем ресурсном узле ГРИД
2. Распаковка «контейнера» в рабочей директории псевдопользователя ГРИД и перемещение библиотек в локальную директорию /tmp на текущем локальном узле (ТЛУ) или в доступную NFS область.
3. Создание файла mpd.conf (на базе шаблона) на ТЛУ для запуска MPI-2 сервера mpd.
4. Переопределение на ТЛУ текущего значения ряда переменных среды окружения для ГРИД пользователя.
5. Запуск сервера mpd (с правами mapped-user) на стартовом узле и проведение его run-time тестирования.
6. Сбор информации о доступных узлах и их текущем состоянии (обычно опрос PBS сервера). Список свободных узлов собирается в файл mpd.hosts, необходимый для запуска «кольца» серверов mpd.
7. Распределение необходимых библиотек по списку свободных узлов по протоколу ssh (или в определяемую NFS директорию)
8. Запуск «кольца» серверов mpd на ресурсном узле ГРИД и его тестирование.
9. Запуск параллельного приложения и его работа как обычного распределенного задания с последующей передачей результатов на брокер ресурсов и затем – на пользовательский интерфейс
10. Удаление всех библиотек и созданных временных файлов со всех узлов или из NFS папки

Отметим, что возможно распространение необходимых файлов на доступные расчетные узлы как через ssh, так и путем организации единой папки на общем NFS ресурсе.

Работа тестового варианта пакета была отлажена на ресурсном узле ГРИД ИПХФ РАН (использовался как удаленный ресурс) в условиях сред gLite и Unicore. Дальнейшее тестирование было проведено на ресурсных узлах RDIG в рамках ВО RGSTEST (узлы НИИЯФ МГУ).

Создание виртуального «контейнера» на примере параллельного квантово-химического приложения GAMESS-US

Для тестирования разработанного метода на примере конкретного прикладного пакета был выбран квантово-химический пакет GAMESS-US.

GAMESS-US (http://www.msg.ameslab.gov/GAMESS) – одна из популярных программ для теоретического исследования свойств химических систем, уступает по известности лишь комплексу Gaussian, позволяет рассчитывать энергию, структуры молекул, частоты их колебаний, а также разнообразные свойства молекул в газовой фазе и в растворе, как в основном, так и в возбужденных состояниях. Основное направление – развитие методов расчета сверхбольших молекулярных систем. Основные программные модули GAMESS-US поддерживают параллельный режим вычислений как на многопроцессорных компьютерах, так и на кластерах рабочих станций UNIX. Пакет отличается сложностью установки и конфигурации, а также требует нестандартных настроек параллельной среды вычислений.

Работы по распараллеливанию GAMESS-US начались еще в 1991 году. Однако использование методов передачи сообщений MPI получило применение только с 1999 г., когда в пакете GAMESS-US была реализована модель интерфейса с распределенным размещение данных (DDI – Data Distributed Interface). Последняя версия интерфейса DDI, которая была оптимизирована для многопроцессорных SMP-архитектур общего вида, особенно работающих с памятью в стиле System V, была выпущена только в мае 2004 г. В настоящее время практически все ab initio методы, включенные в пакет GAMESS, могут использовать параллельные вычисления.

Интерфейс DDI использует в качестве базовой *сокетную* TCP/IP модель межпроцессорных коммуникаций. Использование такого метода распараллеливания для работы на локальном кластере достаточно эффективно и довольно просто в конфигурации, но при работе в ГРИД средах возникает ряд принципиальных проблем: а) необходимо заранее явно указывать используемые расчетные узлы (что обычно нереально); б) неправильно оценивается загруженность расчетных узлов (учитывается только первый расчетный узел); в) отсутствует возможность контроля выполнения удаленной задачи средствами распределенного middleware; (г) на ряде современных кластеров (например «Чебышёв» в НИВЦ МГУ) сокетная модель неработоспособна из-за политик безопасности кластера.

Конфигурации же GAMESS-US с использованием библиотеки MPI авторами пакета разработаны только для ряда мейнфреймов известных производителей (Cray, IBM, SGI). В общем случае конфигурации с MPI не рекомендуются, и желающим предлагается экспериментировать с такими конфигурациями самостоятельно.

Для работы с пакетом GAMESS-US в среде ГРИД на ресурсных узлах ИПХФ РАН первоначально была установлена последняя наиболее широко распространенная версия Mpich-1.2.7 (см.выше), которая является реализацией стандарта MPI-1. Достоинством данной версии является то, что она явно включает интерфейс Globus-2, основанный на Globus Runtime System, что было бы эффективно для запуска Globus заданий. Однако получить работоспособную конфигурацию пакета GAMESS-US для MPI-1 не удалось по двум основным причинам:

- во-первых, из-за особенностей запуска исполняемого задания GAMESS-US, которая осуществляется скриптом, активизирующем более 150 переменных окружения. На главном узле среда создается правильно, но механизм передачи переменных окружения на подчиненные узлы в библиотеке Mpich-1 стандартно отсутствует. В пакет Mpich был включен безопасный сервер (“secure server”), одной из задач которого являлась ликвидация этого недостатка. Но из-за неполной совместимости с операционной системой ScientificLinux эту функцию безопасного сервера использовать не удалось. При запуске задания на локальном узле пакет Mpich-1 использует команды оболочки такие, как ‘.’ , `eval`, `exec`, которые не наследуют среду окружения запускающего процесса, что ведет к краху дочерних процессов;
- во-вторых – особенности реализации команды запуска параллельных заданий `mpirun` пакета Mpich-1. Запуск заданий на главном и подчиненных узлах существенно различаются — строки команды удаленного запуска (`rsh` или `ssh`) на подчиненных узлах дополняются служебными переменными. В результате стандартное расположение строчных аргументов задания GAMESS-US нарушается и не распознается, что ведет к краху запуска.

В ИПХФ РАН с целью расширения функциональности применения пакета GAMESS-US в сети ГРИД была поставлена задача разработки оригинальной конфигурации и сборки из исходных текстов исполняемого файла пакета GAMESS-US с использованием библиотеки MPI-2.

После установки библиотек MPI стандарта 2.0 (версия 1.0.3 пакета Mpich2) была проведена соответствующая модификация конфигурационных скриптов пакета GAMESS-US (`compddi`, `comp`, `compall`, `lked`), а также программных модулей `ddi_init.c` и `ddi_base.h`. Был полностью переписан соответствующий раздел в запускающем скрипте `rungms`, который сначала запускает кольцо серверов `mpd`, а затем уже и само задание. После сборки исполняемого файла было проведено его тестирование на включенных в пакет GAMESS-US примерах файлов данных и получено совпадение результатов. Запуск параллельных заданий осуществляется командой `mpiexec`, которая не имеет указанных выше недостатков команды `mpirun` (библиотеки MPI-1).

Использование модифицированной авторами библиотеки MPI позволило впервые из отредактированных исходных текстов свободно распространяемого квантово-химического пакета GAMESS-US получить исполняемое задание для работы в параллельной среде под управлением MPI. Выбранный авторами подход по созданию виртуального образа среды исполнения на основе перемещаемого пакета MPI-2 показал свою продуктивность и в этом случае. Была проведена компиляция модифицированных исходных кодов GAMESS-US с использованием библиотек MPI-2 и получен бинарный пакет. Затем была создана система компоновки необходимых системных файлов (библиотеки, исполняемые системные файлы), собственно модифицированного GAMESS-US, сопутствующих конфигурационных файлов и настроечных скриптов,

файлов данных в единый «контейнер», выступающий в роли исходящего задания распределенной среды. Запуск подобного контейнера аналогичен описанному выше для прототипа.

Серия первичных запусков (с использованием внутренних тестовых примеров собственно пакета GAMESS-US) вплоть до получения положительного результата тестов (равно-значность поведения сокетных и MPI вариантов) была проведена на ресурсном сайте ГРИД ИПХФ РАН (grid-ce.icp.ac.ru) для сред gLite и Unicore (узлы использовались как удаленные ресурсы, т.е. запуск задач шел через ГРИД инфраструктуру). Дальнейшее успешное тестирование было проведено на удаленных ресурсных узлах RDIG в рамках ВО RGSTEST (узлы НИИЯФ МГУ, lcg38.sinp.msu.ru, среда gLite). Были проведены успешные запуски пакета GAMESS-US с применением данной технологии (рассчитаны тестовые примеры молекулярных структур из дистрибутива GAMESS, например, серия *ab initio* расчетов по оптимизации геометрии в 15-атомной системе ($P_3O_9H_3$) на уровне HF/6-31G*), подтвердившие полную работоспособность разработанной технологии.

Ввиду того, что на ряде кластеров (например, в Курчатовском РНЦ) запрещено или затруднено использование скриптовых языков, нами проведены работы по переводу всех действий по развертыванию и настройке подобных «контейнеров» в полностью бинарные исполняемые программы, которые действуют схожим образом, но не требуют доступа к shell языкам. Таким образом, впервые разработана технология запуска GAMESS-US в ГРИД среде в виде единого откомпилированного бинарного файла. При этом входящая задача порождает единственный процесс, который распаковывает библиотеки и бинарные системные файлы, собственно прикладной пакет, файлы данных, настраивает среду исполнения, в том числе mpich2, запускает параллельные процессы GAMESS-US, собирает полученные результаты, удаляет «мусор» и отправляет выходные данные на пользовательский интерфейс ГРИД среды.

В результате пользователь получает единое *виртуальное* приложение, которое в виде «виртуального контейнера» доставляется на ресурсный узел вместе со всеми конфигурационными настройками, относящимися к операционной системе, и поддержкой необходимых параллельных протоколов, не требуя процедуры предварительной установки и настройки. Далее «виртуальный контейнер» самостоятельно разворачивается на всех выделенных узлах ресурса ГРИД, подготавливая среду для исполнения параллельного приложения с последующим его запуском. При этом отсутствуют конфликты приложения с другими, уже установленными на узле программами и даже с другими экземплярами этого же приложения. Суть виртуализации приложения заключается в создании персональной копии необходимой части системных файлов и настроек операционной системы и доставке приложения совместно с этой информацией с последующим запуском в изолированном «контейнере». Проведенные авторами эксперименты в этой области показали, что так могут быть решены проблемы установки, настройки, несовместимости с операционной системой и другими программами, разрешаются конфликты одинаковых приложений. Заметим, что данная технология применима для запуска подобных приложений и в условиях локальных кластеров без необходимости настройки расчетных узлов.

В перспективе более общим вариантом данных технологий является использование (по аналогии с описанным «контейнером») виртуальных машин как исходящих распределенных заданий, что позволит гарантировать пользователю необходимое качество обслуживания, не затрагивающее при этом работу основных служб ресурсных узлов. Таким образом, пользователю распределенной среды может быть предоставлена полностью изолированная виртуальная вычислительная среда, по своим свойствам не уступающая физическому серверу, в которой может быть предоставлен любой его собственный вычислительный сервис. Приложения, реализованные в ВМ, в этом случае абсолютно не зависят от операционной системы и окружения, в котором ВМ выполняется. Пользователь получает возможность создать образ виртуальной машины с предустановленной операционной системой и полностью сконфигурированными приложениями, нацеленной на решение конкретной задачи. Этот образ затем передается на распределенный ресурс и исполняется там как ГРИД приложение, не требуя настройки данного узла под конкретные задачи. Это существенно облегчает адаптацию прикладного ПО для работы в распределенных средах. Дополнительным плюсом служит то, что данные технологии в принципе позволяют запускать образы виртуальных машин с операционными системами, отличными от установленных на ресурсах (например, Windows ВМ на Linux-кластере). Следует отметить потенциальные недостатки данного метода. Прежде всего – это размер передаваемых

заданий (может достигать первых гигабайтов) и «накладные» расходы на виртуализацию (до 15-20% от мощности ресурса, при оптимальном конфигурации они могут быть снижены до уровня 5-7%).

Заключение

Разработан метод создания виртуальных перемещаемых «контейнеров», которые содержат: «персональные» копии необходимых системных файлов и библиотек, скрипты по настройке операционной системы, необходимые файловые «деревья», собственно приложение, файлы данных и т.п.. После динамического создания «контейнера» он средствами распределенной среды как обычное ГРИД задание доставляется на удаленный ресурсный узел, «разворачивается», настраивает среду узла «под себя» и запускается как обычное параллельное приложение. По окончании работы приложения происходит «очистка» среды выполнения и возврат результатов на пользовательский интерфейс. Созданы два варианта «контейнеров»: с использованием скриптовых языков и как единого бинарного задания. В настоящее время этот метод применим для исполнения на узлах, поддерживающих ОС системы Linux, т.е. типичных кластерах, интегрированных в ГРИД среды.

В качестве примера использован классический квантово-химический пакет GAMESS-US, для которого создан работоспособный «виртуальный контейнер» для сред gLite и Unicore. Нет принципиальных ограничений для создания подобных «контейнеров» для других прикладных пакетов, требующих специфических параметров окружения и нестандартных настроек параллельных сред выполнения.

Применение данного метода виртуализации позволит существенно расширить круг доступных ГРИД ресурсов для выполнения на них сложно сконфигурированных прикладных пакетов.

Литература

1. В.М.Волохов, Д.А.Варламов, А.В.Пивушков, Н.Ф.Сурков, Г.А.Покатович ГРИД и вычислительная химия // "Вычислительные методы и программирование", М.: МГУ, 2009, т.10, № 2,с.78-88
2. С.М. Алдошин, В.М. Волохов, Д.А. Варламов, А.В. Пивушков Вычислительная химия в среде ГРИД: параллельные и распределенные вычисления // Вторая международная конференция «Суперкомпьютерные системы и их применение» SSA’2008, Минск, октябрь 2008; Минск, ОИПИ НАН Беларуси, с.114-118
3. Варламов Д.А., Волохов В.М., Пивушков А.В., Сурков Н.Ф., Покатович Г.А. Распределенные и параллельные вычисления в области химии на ресурсном узле ГРИД ИПХФ РАН // Сб. науч.тр. “Distributed Computing and Grid-Technologies in Science and Education: Extended Proceedings of the 3rd Intern.Conf.”, Дубна, изд-во ОИЯИ, 2008, с.127-130
4. В.М. Волохов, Д.А. Варламов, А.В. Пивушков Крупномасштабные задачи химии на параллельных и распределенных вычислительных полигонах: современное состояние и перспективы // “Научный сервис в сети Интернет: решение больших задач», Всероссийская научная конференция, (г. Новороссийск, 22-27 сентября 2008) – М.; Изд-во МГУ, 2008, с.210-212
5. В.М. Волохов, Д.А. Варламов, Н.Ф. Сурков, А.В. Пивушков Виртуальные вычислительные среды: использование на ГРИД полигонах // Вестн. ЮУрГУ, серия «Математическое моделирование и программирование», 2009, № 17 (150), вып. 3, с.24-35.
6. Д.А.Варламов, В.М.Волохов, А.В.Пивушков, Н.Ф.Сурков Технологии виртуализации ресурсов и приложений вычислительной химии для использования на ГРИД полигонах // «Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность», Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г. Новороссийск). – М.: Изд-во МГУ, 2009. – 524 с., с.378-381

Применение суперкомпьютеров для молекулярно-динамического моделирования процессов в конденсированных средах *

А.В. Янилкин, П.А. Жиляев, А.Ю. Куксин, Г.Э. Норман, В.В. Писарев, В.В. Стегайлов

В работе обсуждается использование метода классической молекулярной динамики на суперкомпьютерах. Исследована эффективность распараллеливания пакета LAMMPS вплоть до 8тыс. ядер на МВС-100К МСЦ РАН. Выработаны рекомендации по его эффективному использованию, обсуждается выбор числа частиц и времени расчета. Представлен набор моделей и подходов для молекулярно-динамического моделирования фазовых превращений и разрушения. Проведен анализ предсказательности моделирования на основе сравнения результатов расчетов с экспериментом.

1. Введение

Область применение суперкомпьютеров для расчётов методом классической молекулярной динамики (МД) определяется возможностью распараллеливания и развитием МД моделей различных процессов. В данной работе исследуется параллельная эффективность пакета LAMMPS [1] вплоть до 8тыс. ядер (МВС-100К МСЦ РАН) и применение его для исследования процессов в конденсированных средах. В качестве примеров применения представлены результаты расчетов пластической деформации и разрушения.

Для распараллеливания классических МД задач используются хорошо зарекомендовавшие себя алгоритмы декомпозиции по пространству. На сегодняшний день классические молекулярно-динамические программы позволяются рассматривать системы, состоящие из 10^{12} атомов [2]. Архитектура суперкомпьютеров накладывает свои ограничения на проведения крупномасштабных расчетов. На МВС-100К МСЦ РАН проведены тестовые расчеты с целью выработки набора рекомендаций для наиболее эффективного проведения расчетов и анализа данных для различных задач, решаемых методом классической МД, для описания которых требуется большое число частиц. Тесты включали в себя расчеты с 1 млрд. атомов и полной загрузкой суперкомпьютера. Определены границы максимального числа частиц и область эффективного использования.

В работе исследуются механизмы и скорости пластического деформирования и разрушения металлов и сплавов при высокоскоростном деформировании (10^6 $с^{-1}$ и выше), имеющем место в ударно-волновых явлениях, при импульсном воздействии лазерного излучения или корпускулярных пучков. Изучение проводится на основе моделирования методом молекулярной динамики элементарных процессов, а именно: образование, движение и объединение дислокаций и других дефектов, обеспечивающих неупругое деформирование кристаллической решетки (пластичность); зарождение и рост полостей (разрушение) [3–5].

На примере актуальной задачи физики высокоскоростного разрушения проведен крупномасштабный расчет разрыва или вскипания растянутой жидкости. Для моделирования использовано 64 млн. атомов и загрузка около 3500 ядер суперкомпьютера МВС-100К. Приводится сопоставление с данными о динамической прочности и пластичности из ударно-волновых экспериментов [6].

*Расчеты выполнены в МСЦ РАН и на кластере МФТИ-60 кафедры информатики МФТИ (ГУ). Работа поддержана по гранту РФФИ 09-08-12161-офи-м, программам фундаментальных исследований РАН № 1, 2 и контракту с Sandia National Laboratories в рамках U.S. DOE/NNSA Advanced Simulation and Computing program.

2. Выбор оптимального числа процессоров и времени расчета

2.1. Эффективность распараллеливания МД задач

Для каждого фиксированного размера системы определялось время расчета 100 шагов, запускаемой на различном числе ядер (PEs) от 1 до 1000 в случае потенциала LJ и от 1 до 7200 в случае алюминия. Эффективность параллелизации определяется ускорением вычислений, которое рассчитывалась как отношение времени расчета на одном процессоре t_1 к реальному времени расчета t_{PEs}. Результаты представлены на рис.1. Для небольших размеров систем (N=32000) наблюдается значительное снижение эффективности для небольшого числа ядер (<50), а при PEs > 100 с дальнейшим увеличением числа ядер время счета практически не меняется. Аналогичные расчеты были проведены для других размеров систем 256000, $2 \cdot 10^6$, $1.6 \cdot 10^7$, $1.3 \cdot 10^7$, $1.3 \cdot 10^8$. Поскольку проводить расчеты в области насыщения не имеет смысла, в тестовых задачах рассматривалась область с параллельной эффективностью более 60%. Наблюдается хорошая масштабируемость вплоть до расчетов на 7тыс. ядрах.

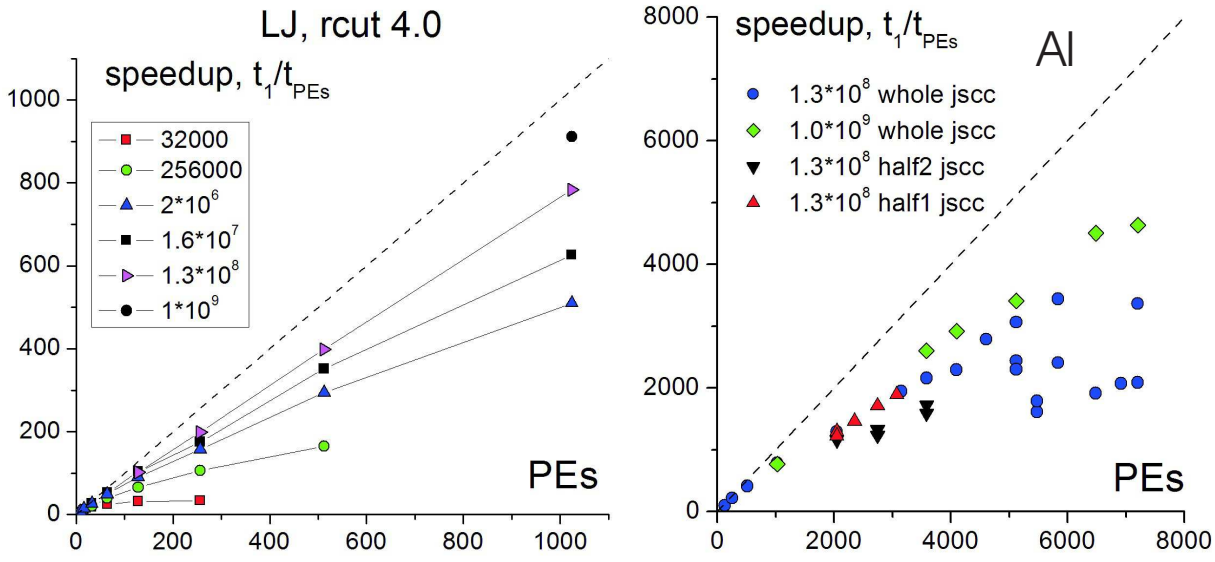


Рис. 1. Зависимость ускорения от числа ядер для различных размеров систем: точки - алюминий (EAM), пунктир - идеальное ускорение.

Как уже было отмечено, использовать суперкомпьютер для расчета в области насыщения, где время расчета практически не меняется с увеличением числа ядер, нецелесообразно, поэтому необходимо ввести критерии эффективного использования. В данном случае мы использовали критерий, основанный на времени обмена $t_{comm}/t_{full} = 20\%$, где t_{comm} - время обмена, t_{full} - общее время расчета. В соответствии с этим критерием для данного размера системы (числа атомов) определена область эффективного использования рис.2.

Таким образом, кривая эффективности определяет нижнюю границу по используемому числу атомов. С другой стороны увеличение числа частиц требует все больший объем оперативной памяти. В связи с этим возникает вторая кривая, ограничивающая число атомов сверху. В нашем случае 1Гбайт оперативной памяти позволяет рассчитывать примерно 1 млн. атомов. Таким образом, возникает полуостров в координатах число частиц - число ядер, в котором можно проводить эффективные расчеты.

На графике приведен рекордный на сегодняшний день расчет: 10^{12} атомов, взаимодействующих по потенциалу LJ. Расчет был проведен на суперкомпьютере BlueGene. Сравнение показывает, что существующая архитектура позволяет проводить подобные расчеты

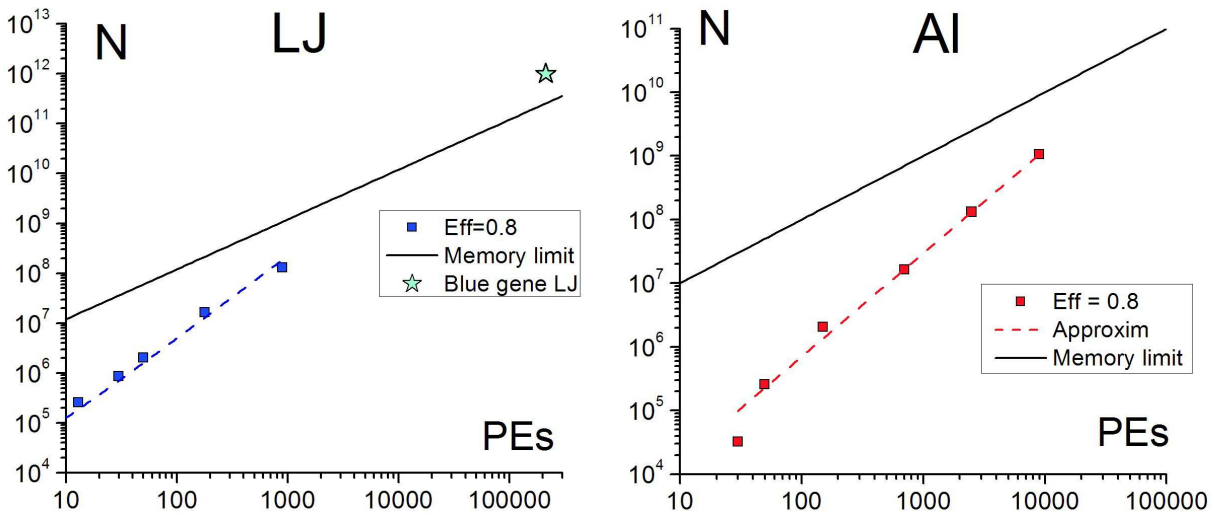


Рис. 2. График число частиц - число ядер. Точками показаны расчеты с эффективность 80%. Сплошная кривая - ограничение по оперативной памяти. Для сравнения приведет максимальный на сегодняшний день расчет на суперкомпьютере BlueGene.

с высокой эффективностью, но требует увеличения объема оперативной памяти.

2.2. Время расчета

Отдельный интерес для планирования расчетов представляет время расчета одного шага интегрирования (рис.3).

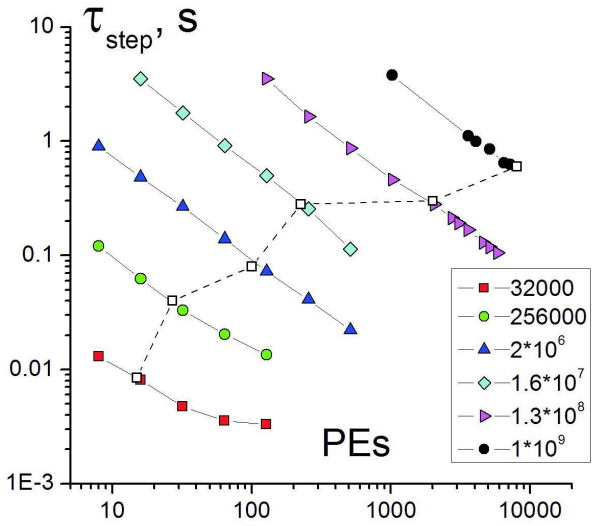


Рис. 3. Зависимость времени расчета одного шага от числа используемых ядер для различных размеров систем. Открытыми точками и пунктиром показана граница области эффективности расчета 80%.

На основании такого графика можно определить характерные физические времена, которые можно рассчитать в условиях полной загрузки кластера за разумное время. В качестве используемого размера системы рассмотрим 1 млрд. атомов. Время одного шага составляет $\tau_{step} = 1$ сек. Большинство физических процессов, рассматриваемых с помощью классического метода молекулярной динамики, протекают за времена, значительно больше 1 пс, поэтому будем рассматривать эту величину как наименьшую. Длительность стан-

дартного шага интегрирования составляет 1 фс или 0.001 пс. Тогда за один час машинного времени (3600 сек) при полной загрузке можно посчитать 3.6 пс для 1 млрд. частиц. Тогда за один день можно посчитать 86 пс для 1 млрд. или почти 0.86 нс для 100 млн. частиц. Эти оценки зависят от потенциала взаимодействия и могут изменяться как в ту, так и в другую сторону, но для большинства задач по порядку величины они дают правильное значение.

2.3. Выбор числа частиц

Выбор значения N определяется масштабами пространственных и временных корреляций, характерных для поставленной задачи. В системе существует иерархия корреляций $r_{c1} < r_{c2} < r_{c3} < ...$, которой соответствует иерархия $N_1 < N_2 < N_3 < ...$, где $N_i = n{r_{ci}}^3$, где n - концентрация частиц, r_c - область расстояний, которая исследуется. Иными словами можно сказать, что, выбирая то или иное значение N, мы тем самым обрываем ряд корреляций, которые можно будет исследовать в данном МД расчете. Выбирая $N = nL^3$, мы также ограничиваем длины волн $\lambda < L$ равновесных флуктуаций, т.е. фиксируем диапазон волновых векторов, для которого можно будет рассчитать дисперсию колебаний плотности: фононов в конденсированных средах, плазменных волн в неидеальной плазме, колебаний биомолекул и т.п. Подобным же образом выбор N ограничивает область исследуемых характеристик таких кооперативных явлений как дислокации, образование трещин и др.

Наряду с иерархией радиусов корреляций, существует иерархия времен корреляций $\tau_{c1} < \tau_{c2} < \tau_{c3} < ...$ Выбор L обрывает этот ряд, в частности, двумя неравенствами. Во-первых, $6D\tau_{ci} < L^2$, где D - коэффициент диффузии. Во-вторых, $a_s\tau_{ci} < L$, где a_s - скорость звука.

Общий вывод заключается в том, что выбор размера системы (числа частиц) ограничивает предельные значения r_c, τ_c, λ и т. п., а также круг явлений и процессов, которые можно исследовать. Выше мы говорили об однородной системе с периодическими граничными условиями. Очевидные варианты требований возникают при моделировании поверхностей, фазовых равновесий и т.п. При переходе к исследованию релаксационных процессов следует учитывать возможность появления дополнительных пространственных и временных характерных масштабов и соответствующих требований на выбор N.

Физически обоснованный выбор числа частиц в сочетании с тестированием эффективности распараллеливания (см. выше) позволяет установить оптимальное соотношение количество частиц – число вычислительных ядер и проведение исследования выбранного свойства, явления и процесса именно в рамках этого соотношения.

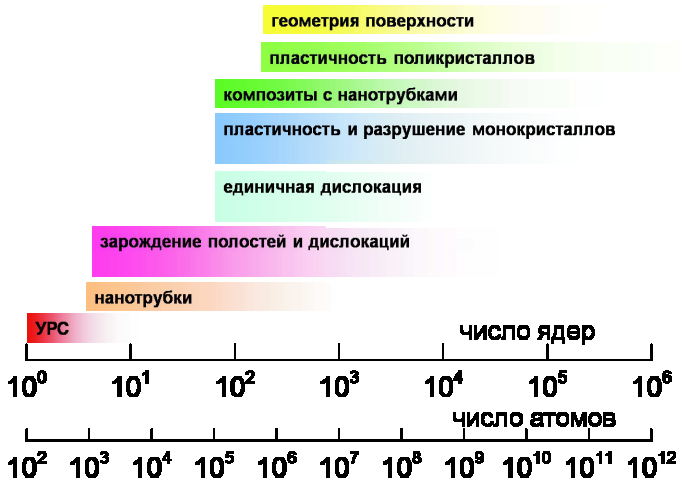


Рис. 4. Число частиц для различных структур и процессов.

Выбор числа частиц определяется рассматриваемым физическим явлениями и структурой. Применительно к физическим задачам, рассматриваемым в данной работе, можно

выделить следующие требования на размер системы рис.4

3. Применение высокопроизводительных расчетов в физике пластичности и прочности

3.1. Пластическое деформирование металлов

Основным механизмом пластической деформации кристаллических материалов является движение дислокаций (рис.5). Поэтому одним из способов упрочнения материалов является создание препятствий, которые бы задерживали дислокации. Примером такого материала является хорошо известный сплав алюминия и меди - дюралюминий. Метод МД успешно использован для исследования взаимодействия движущейся дислокации с включениями и полостями нанометрового размера. Рассмотрим пример подобного расчета для монокристаллического Al с нановключениями Cu. Для анализа используется МД модель, представляющая собой монокристалл с единичной краевой дислокацией. Визуальная картина взаимодействия дислокации и препятствия показана на рис.5. Под действием приложенных напряжений часть дислокационной линии проходит вперед, а часть задерживается на нановключении меди, препятствия дальнейшему движению. При увеличении приложенных напряжений дислокация отрывается от медного включения. Такой механизм наблюдается для препятствий в виде полости.

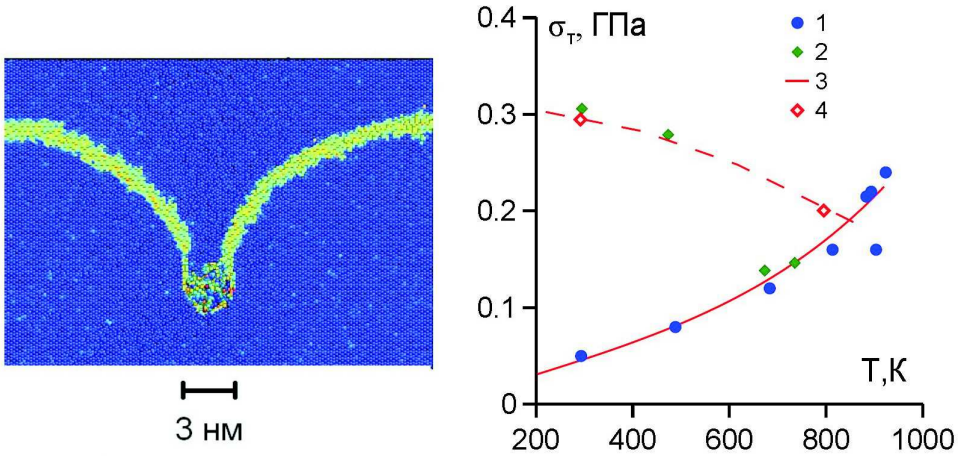


Рис. 5. Показан изгиб дислокации (светлые атомы) в МД модели при ее захвате нановключением в процессе сдвиговой деформации. Размер системы 2 млн. атомов. На графике показана зависимость динамического предела текучести от температуры. Экспериментальные данные: 1 – монокристалл Al, 2 – сплав Д16Т. Данная работа: 3 – монокристалл Al, 4 – сплав Al – 4%Cu.

Оценка напряжений, необходимых дислокации для преодоления кластеров меди и полостей, показала, что создание препятствий нанометрового размера (0.5 - 3 нм) существенно повышает предел текучести материала при низких температурах. Характерное расстояние между препятствиями, рассмотренное в расчетах, составляет 10-60 нм. Для кластеров с диаметрами в десятки нанометров с ростом температуры наблюдается существенно меньшее по величине снижение предельного напряжения. Таким образом, процесс преодоления препятствия носит термофлуктуационный характер. Оценки для Al с включениями Cu (4%) постоянного размера 1.5 нм приведены на рис.5 и согласуются с экспериментальными данными для неотожженного сплава аналогичного состава.

Большая доля материалов не является отдельными сплошными монокристаллами, а состоит из большого числа кристаллитов (зерен), соединенных между собой межзеренными

границами. Такие вещества называются поликристаллическими. Было обнаружено, что механические свойства поликристаллических материалов с характерным размером зерна до нескольких сотен нанометров существенно отличаются от свойств обычных крупнозернистых материалов. Было показано, что уменьшение размера зерна приводит к значительному увеличению прочности материала (эффект Холла-Петча). Причиной является то, что в результате различной ориентации зерен дислокациям затруднительно переходить из одного зерна в другое, т.е. межзеренные границы являются препятствиями для движения дислокаций.

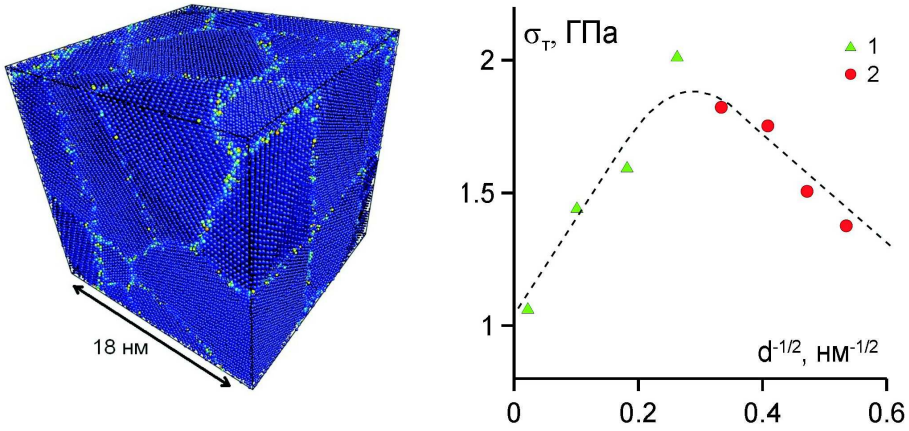


Рис. 6. Показан пример расчетной ячейки МД модели нанокристаллической меди. Светлый цвет атомов выделяет межзеренные границы. На графике показана зависимость предела текучести от размера зерна: 1 – эксперимент, 2 – данная работа (максимум предела текучести при d 10 нм).

Атомистическое моделирование позволяет исследовать механизмы и закономерности рассмотренного явления. Результаты МД моделирования свидетельствуют, что в нанокристаллической меди в рассмотренном диапазоне размеров зерна (3.5-13.5 нм) пластическая деформация обусловлена как дислокационным движением внутри зерен, так и зернограничным проскальзыванием (перемещением атомов вдоль межзеренных границ). Получена зависимость предела текучести от среднего размера зерна, которая хорошо дополняет экспериментальные данные, и дает оценку максимального предела текучести (рис.6).

3.2. Разрыв жидкости

Произведено моделирование разрыва растянутой жидкости (рис.7). Рассмотрен режим растяжения с постоянной скоростью деформирования. Расчеты проводились с использованием пакета LAMMPS. Межатомные взаимодействия описывались потенциалом Леннарда-Джонса (радиус обрезания потенциала 4σ, скин-радиус для списка Верле 0.3 σ, число соседей в сфере взаимодействия 125). Параметры потенциала подобраны, чтобы моделировать жидкий гексан: $\epsilon/\mathrm{k} = 413$ K, σ=5.9Å. Моделируемая система содержала 64 млн. частиц.

Важным требованием для моделирования разрыва является то, чтобы за время нарастания напряжения в системе происходили зарождение и рост множества полостей. Концентрация полостей в этом случае обратно пропорциональна скорости растяжения. Оценки показывают, что для скорости растяжения 10^9 c^{-1} в системе, содержащей 512 тыс. частиц, успевает зародиться ≈100 полостей. Чтобы такое же число полостей образовалось при скорости растяжения $2 \cdot 10^8 c^{-1}$, необходима система, содержащая более 60 млн. частиц. Характерное время, за которое развивается разрыв, также обратно пропорционально скорости растяжения и составляет около 300 пс при скорости растяжения $5 \cdot 10^8 c^{-1}$. Таким образом, для моделирования разрыва жидкости при скоростях растяжения $10^8 - 10^9 c^{-1}$ необходимы

молекулярно-динамические расчёты очень длинных траекторий (сотни тысяч шагов при шаге интегрирования ≈1 фс) для очень большого числа частиц.

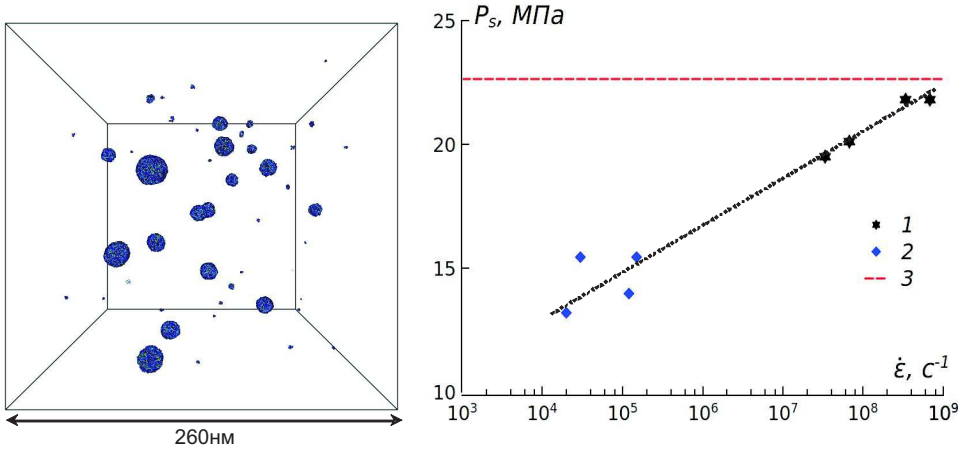


Рис. 7. Снимок расчётной ячейки (показаны только частицы на поверхности полостей). Зависимость прочности жидкого гексана от скорости растяжения. 1 - МД, 2 - эксперимент, 3 - спинодаль жидкость-пар.

Расчеты проводились в выделенном режиме на суперкомпьютерном кластере МВС-100К МСЦ РАН. Для расчетов использовались до 3400 вычислительных ядер. Расчёт 20 тыс. шагов занимал около 6000 сек. при использовании 3000 ядер. 1 ядро, таким образом, выполняет около 60 тыс. атом-шагов. Эффективность расчета составила 85% при использовании 3000 ядер, 83% при использовании 3200 ядер и 80% при использовании 3400 ядер (время обмена составило 15%, 17% и 20% от времени расчета, соответственно). При расчётах с таким числом частиц проблемой является большой размер файлов конфигурации. Так, в наших расчётах бинарный файл, из которого считывалась начальная конфигурация (координаты и скорости частиц), занимал около 5 ГБ. Доступное дисковое пространство позволяет хранить полную конфигурацию не более чем на 3 шагах интегрирования, тогда как для диагностики требуется знать состояние системы каждые 1000 шагов. Поскольку интерес представляли только атомы, лежащие на границах полостей, каждые 1000 шагов в текстовый файл выводились координаты частиц, имеющих избыточную потенциальную энергию. Это позволило выводить в текстовые файлы координаты не более 20 тыс. частиц на каждом шаге, файлы при этом имеют размер несколько мегабайт для каждого запуска (каждый файл содержит конфигурацию на 20 различных шагах интегрирования). Полная конфигурация сохранялась только в двух файлах, которые использовались для запуска новых расчётов.

4. Заключение

В заключение обращается внимание на то, что молекулярное моделирование является движителем развития высокопроизводительной вычислительной техники. Изложенные подходы позволяют оценить число частиц, необходимое для исследования того или иного явления или процесса и указать, ради изучения каких явлений или процессов следует строить все более и более многоядерные системы, и их перспективную архитектуру. Обсуждается предсказательная сила молекулярного моделирования и его соотношение с натурными испытаниями в инженерной практике.

Литература

1. Plimpton S.J. Fast parallel algorithms for short-range molecular dynamics. // J.Comp.Phys. -1995. -Vol. 117, -P. 1-19.

2. Germann T.C., Kadau K. Trillion-atom molecular dynamics becomes a reality. // International Journal of Modern Physics C -2008. -Vol. 19, -P. 1315-1319.

3. Куксин А.Ю., Стегайлов В.В., Янилкин А.В. Молекулярно-динамическое моделирование динамики краевой дислокации в алюминии // Доклады Академии Наук. -2008. -Т. 420, -Р. 467-471.

4. Куксин А.Ю., Янилкин А.В. Кинетическая модель разрушения при высокоскоростном растяжении на примере кристаллического алюминия // Доклады Академии Наук. -2007. -Т. 413, -С. 615-619.

5. Kuksin A. Yu., Norman G. E., Stegailov V. V., and Yanilkin A. V. Molecular Simulation as a Scientific Base of Nanotechnologies in Power Engineering // Journal of Engineering Thermophysics. -2009. -Vol. 18, -P. 197-226.

6. Pisarev V.V., Kuksin A.Yu., Norman G.E., Stegailov V.V., Yanilkin A.V., Microscopic theory and kinetic model of fracture of liquids // In Shock Compression of Condensed Matter - 2009, Ed. By M.D. Furnish et al. American Institute of Physics, New York. 2009.

Об экзапроблемах математического моделирования

В.П. Ильин

1. Введение

Развитие математического моделирования происходит под воздействием трех основных факторов: появление новых актуальных практических задач, в том числе междисциплинарных, обратных и оптимизационных, требующих экстремальных ресурсов; достижения теоретической и вычислительной математики по созданию и обоснованию современных моделей и эффективных алгоритмов; разработка высокопроизводительных компьютерных технологий на базе бурно прогрессирующих многопроцессорных вычислительных систем (МВС).

Стоящие на повестке дня проблемы изучения динамических процессов и явлений включают понятия хаоса, странных аттракторов, фрактальных сред, синергетической самоорганизации сложных систем, инициирующие новые принципы моделирования и требующие сверхвысокого разрешения.

Последние десятилетия ознаменованы созданием новых математических подходов и постановок реализуемых или ожидающих своего воплощения в вычислительных алгоритмах и технологиях, см. [1]–[5]. Здесь можно назвать комплексные согласованные модели механики сплошных сред, единообразно описывающие свойства упругих, пластических, жидких, газообразных и многофазных сред, а также переходы из одних состояний в другие, использование теории групп, дифференциальных форм, распределений, дискретных эйлеровых и лагранжевых представлений, и т.д.

Кардинально меняется понятие “большой” задачи, предполагающей уже применение сеток с миллионами, десятками и сотнями миллионов узлов, для решения которой численные методы стали принципиально параллельными и ориентированными на МВС с сотнями и тысячами процессоров. Проблема обеспечения высокой производительности и масштабируемости крупномасштабных вычислений требует углубленного узучения структуры алгоритмов и их отображения на архитектуру ЭВМ, а также создание и эффективное использование современных программных технологий, см. [6]–[9].

Хотя первый в мире компьютер петафлопной производительности запущен только в 2008 г., уже объявлено грядущее появление в 2019 г. “экзафлопника” с быстродействием 10^{18} операций в секунду (1000 петафлоп!, [10]). А поскольку закон Мура успешно “работает” более 40 лет, данный прогноз представляется вполне реальным, что означает создание в недалеком будущем МВС с миллионами вычислительных ядер. В связи с этим по инициативе Дж.Донгарры и П.Бекмана сформирован международный Exascale Software Project, миссией которого является выработка новых концепций и моделей программирования для компьютеров экстремального и экзафлопного уровня (авторами [10] такое программное обеспечение названо X – stack). Именно глобальные проблемы программного обеспечения, включая операционные системы, компиляторы, базы данных и всевозможные инструментарии, являются главными на ближайшее десятилетие, поскольку существующее “софтверное” оснащение вычислительных систем своими корнями связано с последовательными вычислениями на фон-неймановской машине, а наступающая эра массового параллелизма неизбежно требует перехода количества в качество.

Основной заявленный в [10] принцип построения нового программного обеспечения – создание открытых (Open Source) библиотек для виртуальных машин. Эта стратегия

естественным образом соответствует исторически складывающейся методологии развития прикладного программного обеспечения задач математического моделирования. Чтобы понять возникающие на текущем этапе проблемы, необходимо оценить тенденции возникающих практических задач, эволюцию современных вычислительных методов и новые технологические требования, возникающие из особенностей развития компьютерных архитектур.

2. Прикладные задачи и математические модели

Актуальные практические проблемы из различных сфер человеческой деятельности, требующие все более глубокого понимания протекающих процессов и явлений, являются главным двигателем прогресса в математическом моделировании. Если говорить о естественно-научных направлениях, то практически все они охватываются следующим списком: материаловедение и нанотехнологии, энергетика, машиностроение и металлургия, физика плазмы и высоких энергий, электроника и связь, астрофизика и астрономия, науки о Земле (атмосфера, океан, твердь), биология и медицина.

В терминах уравнений математической физики, или механики сплошных сред (или теоретической физики) эти задачи составляют физику твердого тела, т.е. упругости, пластичности и разрушения, классическую и магнитную гидрогазодинамику, тепломассоперенос в многофазных и пористых средах, электромагнетизм. Каждая из этих дисциплин представляет собой фундаментальное научное направление, изучение которого требует глубокого погружения и профессионализма.

Однако особенность реальных требований в серьезных технических разработках, в наукоемких производствах и в исследовании природных явлений заключается в необходимости междисциплинарного подхода к моделированию с обязательным учетом факторов самой различной природы. Примером может служить проблема долгосрочного прогноза погоды и климата [11], которая включает циркуляции атмосферы и океана, учет мезометеорологических структур и морфологии приземной поверхности, солнечной радиации и ионосферных процессов, распространения примесей и антропогенных воздействий. Математическая модель такой суперзадачи представляет собой начально-краевую задачу для сложнейшей системы нелинейных дифференциальных уравнений или эквивалентных вариационных соотношений со многими неизвестными функциями, имеющими многочисленные разномасштабные во времени и пространстве особенности. Зачастую коэффициенты решаемых уравнений, определяющие важные физические факторы (вязкость, теплопроводность и т.д.), известны с большой долей неопределенности, которую надо устранить путем сопоставления расчетных данных с результатами натурных измерений. Таким образом возникает проблема идентификации параметров модели – одна из типичных обратных задач, для нахождения ответа которой, как правило, требуется многократное решение прямых задач (которые проще хотя бы потому, что в них задается вся информация, необходимая для получения искомого результата). В целом моделирование динамики атмосферы, океана и климата – это работа для профессиональной команды, которая фактически является коллективным естествоиспытателем.

Иллюстрация совсем другого рода – это комплексное моделирование технологических процессов в алюминиевом электролизере [12], представляющем собой в определенном смысле сложный организм, длительное и успешное функционирование которого требует тщательного внимания и управления. Здесь подаются огромные токи и загружается сырье, из которого выплавляется периодически отгружаемый жидкий металл. Конечная практическая проблема в данном случае –

оптимизация производственного режима, обеспечивающего минимизацию целевого функционала, т.е. отношение эксплуатационных расходов (стоимость электроэнергии, материалов, ремонтных работ и т.д.) к стоимости получаемого продукта, характеризуемой итоговым объемом алюминия определенного качества. При этом необходимо удовлетворять различным эксплуатационным ограничениям, например, на температурные напряжения, во избежание разрушения конструкции. Такая обратная задача включает решение множества прямых задач, каждая из которых описывается джентльменским набором уравнений математической физики: Максвелла (электромагнетизм), Ламе (упругость и прочность), Навье–Стокса (циркуляции жидкого металла и электролита), теплопереноса с фазовыми переходами, химической кинетики. Решать эти взаимосвязанные уравнения необходимо в реальных трехмерных областях сложной конфигурации. Достаточно сказать, что компьютерная модель электролизера (среднего уровня детализации) насчитывает более 500 геометрических объектов и 30 различных материалов (анодные и катодные узлы, графитовые блоки, футировочные материалы, корпус и т.п.). К этому следует добавить, что в заводском цехе работает "линейка" из электрически взаимосвязанных более чем 100 электролизеров, магнитные поля которых существенно влияют друг на друга.

Можно привести много других больших задач, решение которых с необходимой точностью (на практике постоянно ужесточающейся) требует привлечения вычислительных ресурсов пета- и экзафлопного масштаба (будем их называть "экстремального", или Х-ресурсов). Подчеркнем еще раз общую для них тенденцию: серьезные задачи являются междисциплинарными и обратными, а расчет какого-то одного поля (например, температурного) представляет собой частный или методический интерес.

Важно отметить, что при всем необозримом многообразии актуальных приложений, их математические постановки могут быть строго классифицированы и составлять конечный набор основных формулировок, который может при необходимости расширяться и углубляться, не выходя при этом за рамки унифицированной технологии. В целом формальное описание достаточно абстрактной начально-краевой задачи может быть представлено следующим образом.

Пусть $\vec{u} = \{u_\mu, \mu = 1,\ldots,m_f\}$ есть вектор-функция, у которой каждая скалярная составляющая u_μ зависит от вектора пространственных координат $\vec{x} = (x_1, x_2, x_3)$ и, возможно, от времени t. Система координат может быть декартовая, цилиндрическая или сферическая (или какая-то специальная), а количество независимых пространственных переменных d (размерность задачи) может уменьшаться с трех до двух или одного.

Обозначим через $\overline{\Omega} = \Omega \cup \Gamma$ замкнутую ограниченную область евклидового пространства с границей Γ, в которой определена функция $\vec{u}(\vec{x},t)$ для каждого момента времени $0 < t \le T < \infty$. Предполагается, что $\vec{u}(\vec{x},t)$ принадлежит некоторому функциональному пространству, в котором имеют смысл операторы L и l. Требуется в расчетной области $\overline{\Omega}$ найти решение системы дифференциальных уравнений (линейной или нелинейной) с известной функцией $\vec{f} = \{f_\mu, \mu = 1,\ldots,m_f\}$

$$L\vec{u} = \vec{f}(\vec{x},t), \quad \vec{x} \in \overline{\Omega}, \quad 0 < t \le T < \infty, \tag{1}$$

удовлетворяющее заданным граничным и начальным условиям

$$l\vec{u} = \vec{g}(\vec{x},t), \quad \vec{x} \in \Gamma, \quad \vec{u}(\vec{x},0) = \vec{u}^0(\vec{x}). \tag{2}$$

В качестве примера можно привести дифференциальный оператор второго порядка

$$L = A\frac{\partial}{\partial t} + \nabla B \nabla + C\nabla + D, \tag{3}$$

где ∇ есть градиент, а A,B,C,D – некоторые квадратные матрицы порядка m, элементы которых суть или постоянные, или функции $\vec{x},t$, или зависят от $\vec{u}$. В зависимости от конкретного вида матричных коэффициентов, математическая постановка (1)–(3) формально описывает стационарные или нестационарные, линейные или нелинейные процессы и явления в огромном числе приложений. Расчетную область будем всегда считать связной (хотя формально это и не обязательно), иначе задача (1)–(2) распадается на независимые подзадачи.

Простейшей иллюстрацией граничных условий является смешанная краевая задача, в которой граница области Γ состоит из частей Γ_D, Γ_N, на которых заданы условия разных типов:

$$u = g_D, \quad x \in \Gamma_D; \quad D_N u + A_N \nabla_n u = g_N, \quad x \in \Gamma_N, \tag{4}$$

где D_N и A_N – матрицы (в общем случае прямоугольные, т.к. на разных участках границы может быть задано разное число условий), а g_D, g_N – вектор-функции, элементы которых или известны, или зависят от искомых решений (∇_n – оператор дифференцирования по направлению внешней нормали к границе).

Вместо классического дифференциального описания (1)–(4) исходная задача может быть представлена в эквивалентной (в каком-то смысле) вариационной постановке для обобщенного решения. Кроме того, вместо дифференциальных могут фигурировать интегральные уравнения, а в общем случае – системы дифференциально-интегральных равенств и/или неравенств.

Приведенная выше формулировка определяет прямые задачи математического моделирования. Исходные данные каждой из них можно определить зависящими от вектора параметров $\vec{p} = (p_1, \ldots, p_{m_0})$, компоненты которого надо оптимизировать по условию минимума описанного заранее целевого функционала от определенного на параметризованном решении $\vec{u}(\vec{x},t,\vec{p})$ поставленной задачи (см. [13] и цитируемые там работы):

$$\Phi_0\left(\vec{u}\left(\vec{x},t,\vec{p}_{opt}\right)\right) = \min_{\vec{p}} \Phi_0\left(\vec{u}\left(\vec{x},t,\vec{p}\ \right)\right), \tag{5}$$

при заданных линейных или функциональных ограничениях $(m_1 + m_2 = m_0)$:

$$p_k^{\min} \le p_k \le p_k^{\max}, \quad k = 1,\ldots,m_1, \quad \Phi_l\left(\vec{u}\left(\vec{x},t,\vec{p}\right)\right) \le \delta_l, \quad l = 1,\ldots,m_2. \tag{6}$$

В этом случае описание прямой задачи фигурирует формально как дополнительное ограничение в виде уравнения состояния

$$L\vec{u}(\vec{p}) = \vec{f}, \quad \vec{p} = \{p_k\}. \tag{7}$$

Таким образом мы приходим к оптимизационной постановке обратной задачи, заключающейся в идентификации параметров модели математической проблемы, которая может включать и комплексное моделирование различного вида полей, когда рассматривается одновременно совокупность физических эффектов, а целевой функционал Φ_0 включает отклонение натурных и расчетных данных для всех видов измеряемых полей.

Расчетную область можно представить как объединение $N_D \ge 1$ замкнутых непересекающихся подобластей, т.е. $\overline{\Omega} = \bigcup_k \overline{\Omega}_k$, $\Omega_k \bigcap \Omega_{k'} = 0$, при $k \ne k'$, где $k,k' = 1,\ldots,N_D$, в каждой из которых могут задаваться свои коэффициенты или даже

различные типы решаемых уравнений. Граница *k*-ой подобласти может быть представлена как объединение смежных границ с примыкающими подобластями:

$$\overline{\Omega}_k = \Omega_k \bigcap \Gamma_k\,,\ \Gamma_k = \bigcup_{k'} \Gamma_{k,k'},\ \overline{\Omega}_k \bigcap \overline{\Omega}_{k'} = \Gamma_{k,k'} = \Gamma_{k',k} \neq 0\,. \qquad (8)$$

Если внешнее пространство (дополнение) по отношению к Ω обозначить формально как подобласть $\Omega_0 = R^d / \overline{\Omega}$, то внешняя граница расчетной области представляется в виде $\Gamma = \Gamma^{(e)} = \bigcup_k \Gamma_{k,0}$, а граница каждой подобласти

может быть разбита на ее внешнюю и внутреннюю части, т.е.

$$\Gamma_k = \Gamma_k^{(e)} \bigcap \Gamma_k^{(i)},\quad \Gamma_k^{(e)} = \Gamma_{k,0}\,,\quad \Gamma_k^{(i)} = \bigcup_{k' \neq 0} \Gamma_{k,k'}. \qquad (9)$$

Если подобласть Ω_k – трехмерный объект, имеющий свой ненулевой объем, то ее граница Γ_k в качестве меры измерения имеет площадь. Каждый из граничных фрагментов $\Gamma_{k,l}$ внешней или внутренней границы (на каждом из которых ставится какое-то краевое условие) будем считать кусочно-гладкой поверхностью, т.е. состоящей из совокупности гладких поверхностных сегментов, характеризуемых своим уравнением. Граница расчетной области Γ является в общем случае многосвязной, или конечносвязной, т.е. состоящей из конечного числа связных множеств. Граница Γ является односвязной, если сама расчетная область не имеет ``дыр''.

Каждый поверхностный сегмент имеет свою граничную линию, состоящую из конечного числа криволинейных (пространственных в общем случае) отрезков, характеризуемых парой своих концевых точек, длиной и уравнениями двух поверхностей,
пересечением которых он является. Такие граничные отрезки будем называть ребрами расчетной области, а их концевые точки, лежащие на пересечении ребер, будем называть вершинами. Совокупность ребер и вершин назовем каркасом области, который формально можно представить в виде многомерного графа.

3. Вычислительные методы и технологии

При всем многообразии математических задач и моделей, поддающихся тем не менее вполне обозримой систематизации, количество методов их решения, естественно, во много раз больше. Однако этапы вычислительного эксперимента также могут быть четко выделены и классифицированы, а алгоритмы их реализации – фрагментированы по модульному принципу, в зависимости от их назначения, применяемых подходов, эффективности, ресурсоемкости, параллелезуемости и т.д.

3.1. Дискретизация математической модели

Задачи математического моделирования в подавляющем своем большинстве описываются в терминах интегро-дифференциального исчисления над функциями непрерывного аргумента. Исключения составляют процессы масштаба межатомных расстояний, которые актуальны, например, в нанотехнологиях, где могут быть даже неприменимы понятия производных, но мы на них останавливаться не будем.

Дискретизация расчетной области, т.е. построение сетки, представляет особо сложную проблему в многомерных случаях с кусочно-гладкими криволинейными и, возможно, движущимися границами (для последних специальный класс составляют т.н. свободные границы, положение которых заранее неизвестно). Сама сетка определяется

совокупностью своих объектов различной размерности: узлы, ребра, грани, конечные объемы,– а также топологическими связями между ними. Между геометрической структурой расчетной области с подобластями и сеточной структурой существует большая аналогия, и они содержат по сути однотипные наборы объектов макро- и микро-уровня. Если математическая постановка является дифференциальной или в форме граничных интегральных уравнений, то отличие дискретизации заключается в том, что в последнем случае сетка строится только на граничных поверхностях расчетной области.

Существует большое количество критериев качества сетки, от которых в значительной степени зависит точность и экономичность численного решения. Одно из главных требований к сетке – адаптивность, означающая в том или ином смысле учет особенностей конфигурации границы и свойств искомого решения, которые определяются или теоретически априори или апостериори экспериментально, т.е. на основе предварительных расчетов.

В первую очередь требуется, чтобы вершины, ребра и граничные поверхности расчетной области составлялись из соответствующих сеточных объектов, или в крайнем случае аппроксимировались ими с возможно малой погрешностью. Второй аспект связан с возникающей сингулярностью, т.е. сильным ростом производных, в окрестности углов и ребер границы, что требует специального сгущения узлов в таких подобластях. А при наличии сильно меняющихся пространственно-временных особенностях решения сетки должны быть динамическими, т.е. регулярно или периодически перестраиваемыми.

Распространенные сеточные технологии включают триангуляции Делоне, ячейки Дирихле–Вороного и различные приемы контроля вырожденных случаев. Важным вычислительным средством являются многосеточные методы, основанные на построении последовательности вложенных сеток или на их локальном сгущении.

Наиболее просто конструируемыми и одновременно обеспечивающими больший порядок точности являются равномерные сетки, которые могут иметь различные типы конечных элементов: кубы или параллелепипеды, призмы, тетраэдры и т.д. Однако в реальных задачах сетки приходится строить неравномерные и нерегулярные, или неструктурированные, у которых для каждого узла номера его ближайших соседей можно задать только перечислением. Существуют компромиссные квазиструктурированные сетки, когда расчетная сеточная область состоит из сеточных подобластей, в каждой из которых сетка строится по своим правилам и может быть структурированной. Такие сетки могут быть несогласованными или согласованными – в последних случаях узлы на смежных границах раздела соседних сеточных подобластей являются совпадающими, т.е. общими.

Методы построения сеток отличаются большим разнообразием и имеют обширную профессиональную литературу. Здесь используются и квазиконформные отображения, и вариационные принципы, и специальные метрические пространства, и многочисленные эмпирические приемы. Соответствующее программное обеспечение, или генераторы сеток, существует в широком ассортименте, или в свободном доступе через Интернет, или в качестве коммерческих продуктов. Зачастую процедуры построения сеток являются неотъемлемой частью проблемно-ориентированных пакетов прикладных программ (ППП), но они также существуют и в автономной форме, позволяющей их встраивать в различные приложения.

Рассмотренные выше принципы дискретизации базируются на эйлеровом подходе. Однако при моделировании движущейся среды оказывается более удобно лагранжевая система координат, порождающая многочисленные варианты методов больших частиц. Широко распространены алгоритмы “частиц в ячейках”, использующие сочетание

эйлеровых сеток с дискретизацией потоков субстанции. Имеются также и “бессеточные” методы с чисто лагранжевым описанием процессов.

Все рассматриваемые выше подходы относятся к детерминистским алгоритмам, альтернативу которым составляют статистическое моделирование и методы Монте–Карло. Основанные на вероятностном принципе, они являются незаменимым инструментом исследования процессов и явлений со случайными исходными данными или описываемыми многомерными дискретными соотношениями. Они обладают высокой универсальностью и формально могут быть применены к решению практически любых дифференциальных и/или интегральных уравнений, однако вопрос о целесообразности их выбора – это уже компетенция конкретного квалифицированного пользователя.

3.2. Алгоритмы аппроксимации

Строго говоря, генерация сетки является первым этапом дискретизации, конечной целью которой является переход от исходных функциональных соотношений к конечно-мерным уравнениям, неравенствам или рекурсиям. При этом фактически задача алгебраизируется, что достигается путем аппроксимации функций, производных и интегралов. Основные подходы к построению сеточных соотношений – это методы конечных разностей, конечных объемов, конечных элементов (МКР, МКО, МКЭ), коллокаций и спектральные методы, связанные с разложениями в ряды Фурье. Не пытаясь делать обзора имеющегося огромного материала по данным вопросам, мы коротко остановимся главным образом на технологических аспектах наиболее универсальных МКО и МКЭ, позволяющих конструировать сеточные аппроксимации высоких порядков точности на различных типах конечных объемов для широкого класса задач математического моделирования.

Важным моментом этих двух близких подходов является поэлементная технология вычисления локальных матриц и векторов правых частей с последующей сборкой (ассемблированием) глобальных матриц и систем алгебраических уравнений (линейных или нелинейных – СЛАУ или СНАУ). Этот прием значительно упрощает программную реализацию и естественным образом обеспечивает эффективное распараллеливание данного вычислительного этапа.

В нестационарных проблемах пространственная аппроксимация осуществляется на каждом временном шаге, а при наличии нелинейностей такая процедура повторяется итерационно для всех шагов. Наиболее трудоемкими оказываются задачи с динамическими сетками, если их приходится перестраивать на каждой итерации для каждого шага по времени.

Совокупность аппроксимационных алгоритмов для типовых задач математической физики может быть систематизирована и классифицирована по следующим признакам:

- по типу аппроксимируемого члена дифференциального уравнения (градиент, дивергенция и т.д.) или его механического смысла; примером могут быть матрицы жесткости и матрицы масс;
- по виду конечных элементов или объемов: тетраэдры, параллелепипеды, призмы и т.д.; при этом могут выделяться частные случаи, которые наиболее экономично реализуются (например, правильные фигуры);
- по характеру базисных функций, которые могут отличаться своими носителями, порядками и структурными свойствами (лагранжевые и эрмитовые, скалярные и векторные, и т.п.);

- по конфигурации сеточного шаблона, т.е. совокупности узлов, участвующих в одном уравнении, получаемом в итоге формирования алгебраических систем; наиболее экономичными оказываются компактные схемы, в которых участвуют только наиболее близкие геометрически соседние узлы; как правило, повышение порядка аппроксимации ведет к " растягиванию" сеточного шаблона и расширению ленточной структуры итоговых матриц, так что с точки зрения общей эффективности алгоритмов приходится искать компромиссную "золотую середину".

3.3. Задачи вычислительной линейной алгебры

Если решаемая проблема в целом является нестационарной и нелинейной, все равно после этапов временной аппроксимации и квазилинеаризации мы приходим к задачам линейной алгебры, из которых наиболее типичны – это решение СЛАУ или проблемы собственных значений (как правило, частичной, т.е. вычисление нескольких собственных чисел и соответствующих собственных векторов).

Характерные матрицы, возникающие в сеточных методах решения дифференциальных задач, являются разреженными, ленточными и большими. Это означает, во-первых, что порядки N достигают десятков и сотен миллионов, а ненулевые элементы сосредоточены в некоторой полосе ширины m около главной диагонали, причем величина m и число ненулевых элементов в каждой строке не зависят от N. Дискретизированные алгебраические системы, возникающие из аппроксимации интегральных уравнений (определенных на границе или в объеме расчетной области) являются, наоборот, плотными, но зачастую имеют специальные структурные свойства (например, являются теплицевыми, квази- или блочно-теплицевыми).

Вычислительная линейная алгебра – хорошо продвинутая математическая дисциплина и содержит большое количество алгоритмов для решения задач с самыми разными типами матриц: вещественными и комплексными, квадратными и прямоугольными, эрмитовыми и неэрмитовыми, положительно определенными и знаконеопределенными. Имеется также соответствующее обширное программное обеспечение в виде библиотек или прикладных пакетов, как свободно распространяемых в Интернете, так и коммерческих. Данные алгоритмы и программы делятся в основном на два класса: ориентированные на плотные матрицы и на разреженные. Первый из них содержит в основном прямые методы, основанные на точной факторизации матрицы, а второй – предобусловленные итерационные алгоритмы или же быстрые прямые процедуры для специальных матричных структур (например, происходящих из многомерных краевых задач с частично или полностью разделяющимися переменными, где эффективно работают знаменитые методы быстрого преобразования Фурье, циклической редукции и некоторые другие).

Алгебраические задачи – "узкое горлышко" математического моделирования, поскольку потребляемые вычислительные ресурсы нелинейно растут с увеличением порядка N. Поэтому здесь особенно актуально распараллеливание алгоритмов на МВС с общей, разделенной и гибридной памятью. Ключевым принципом является в данном случае алгебраическая декомпозиция областей, или "разделяй и властвуй".

Существенным моментом программного обеспечения для разреженных матриц является оптимизация кода, поскольку здесь для хранения ненулевых элементов неизбежно применение сжатых форматов данных, которые позволяют кардинально сокращать объем хранимой информации, но существенно усложняют реализацию

доступа к матричным элементам, что особенно критично при многоуровневой неоднородности кэша и оперативной памяти.

3.4. Методы оптимизации и решения нелинейных уравнений

Алгоритмы оптимизации решения обратных задач, сводящихся к проблеме условной минимизации функционала, являются бурно развивающейся в последние десятилетия областью вычислительной математики. Здесь развиты модификации методов множителей Лагранжа и внутренних точек, являющихся развитием подходов со штрафными функциями, использование доверительных интервалов для регулировки последовательности шагов, а также варианты алгоритмов Ньютона и последовательного квадратичного программирования для решения возникающих на промежуточных этапах СНАУ.

В данной тематике имеется много актуальных и далеко не решенных вопросов, связанных с поиском глобального минимума и оптимального управления, применения методов теории возмущений и сопряженных уравнений, нахождения градиентов функционалов или функций чувствительности к вариациям исходных данных. Зачастую постановки требуют “штучного” исследования устойчивости и корректности задачи для поиска соответствующего регуляризационного подхода. К этой же области можно отнести изучение нелинейных динамических систем, связанных с эффектами бифуркаций, самоорганизации и хаоса, странных аттракторов и т.д., где зачастую еще остаются открытыми методологические принципы математического моделирования.

3.5. Постобработка и визуализация результатов

Непосредственная реализация наукоемких алгоритмов в многомерных задачах может составлять отнюдь не главную долю общего вычислительного процесса, поскольку анализ получаемых результатов требует их презентабельной визуализации с предварительной обработкой сеточных функций, что требует выполнения ресурсоемких технологических операций по формированию сечений, изолиний и изоповерхностей, различных графиков с возможной анимацией многоцветных изображений. Такие технологические проблемы особенно актуальных в системах принятия решений по результатам математического моделирования.

4. Конвергенция алгоритмических структур и компьютерных архитектур

Развивающиеся МВС остаются на текущий момент в рамках типовой кластерной архитектуры: соединенные общей шиной вычислительные узлы, состоящие из определенного количества процессоров и/или ядер с общей многоуровневой памятью. Основные программные средства распараллеливания алгоритмов – это системы MPI и OpenMP для работы с распределенной и общей памятью соответственно. Создаваемые так же интенсивно GRID-системы предназначены в основном для интеграции разнородных вычислительных ресурсов и не должны, по-видимому, рассматриваться как орудие эффективного распараллеливания алгоритмов.

Лет 20 и более назад в тематике компьютерных архитектур наблюдалось заметное многообразие: разрабатывались матричные ЭВМ, транспьютеры и различные виды

вычислительных сетей (гиперкубы и т.п.), обсуждались клеточные автоматы, нейросети и нейрочипы, исследовались специализированные процессоры. Но все они не выдержали конкуренции с универсальными компьютерами общего назначения.

Однако в последние годы ситуация начинает серьезно меняться. Появились многоядерные видеокарты, серьезно повышающие быстродействие формирование многоцветных и многоплановых графических изображений. А главное – благодаря современным кремниевым технологиям активно разрабатываются программируемые логические интегральные схемы (ПЛИС), которые делают реальной давнюю мечту математиков и программистов о создании компьютеров, ориентированных на экономичную реализацию конкретного класса задач или алгоритмов.

Говорить о возможном соответствии алгоритмических структур и компьютерных архитектур удобнее всего в терминах графов. Рассмотрим для примера блочную запись СЛАУ

$$\left(Au\right)_k \equiv A_{k;k}u_k + \sum_{l\in\omega_k} A_{k,l}u_l = f_k, \quad k \in \Omega_k, \tag{10}$$

где u_k и f_k подвекторы векторов u и f, Ω_k – совокупность номеров его индексов, $A_{k,l}$ – блочные подматрицы матрицы A, а ω_k – совокупность номеров ненулевых блоков $A_{k,l}$. В сеточных системах матрицы могут быть несимметричными ($A_{k,l} \neq A_{l,k}^t$, где t означает транспонирование), но всегда являются структурно симметричными, т.е. $A_{k,l}$ и $A_{l,k}$ могут равняться нулю только одновременно. Блочную структуру СЛАУ (10) можно изобразить в виде некоторого макрографа, вершинам которого соответствуют подвекторы и диагональные матричные блоки, а ребрам – соответствующие ненулевые блоки. Можно также с помощью полного матричного графа изобразить матричный портрет A, т.е. всю совокупность ее ненулевых элементов с указанием их расположения.

Соотношения (10) можно рассматривать также как схему алгебраической декомпозиции расчетной области. В этом случае Ω_k есть сеточная подобласть, $A_{k,l}$ и $A_{l,k}$ – матричные записи связей между Ω_k и Ω_l, которые бывают ненулевыми только для геометрически соседних подобластей, а диагональные блоки $A_{k,k}$ представляют подзадачи для соответствующих Ω_k (которые могут решаться независимо и одновременно, в чем и состоит суть распараллеливания). Иллюстрация двумерной прямоугольной декомпозиции приводится на рис. 1, где смежные границы $\Gamma_{k,l}$ соответствуют внедиагональным блокам $A_{k,l}$.

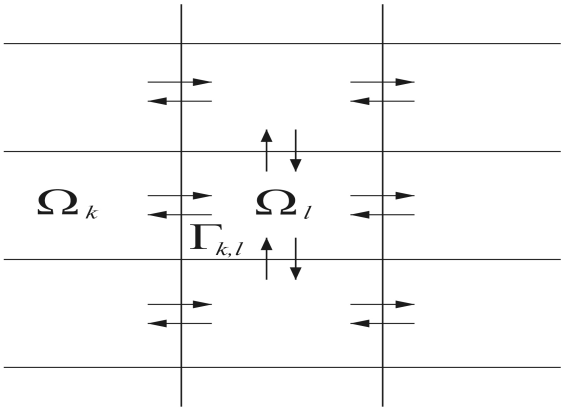


Рис 1. Иллюстрация двумерной декомпозиции

Очевидно, что топологические связи подобластей, в зависимости от способа разбиения расчетной области, могут быть представлены в виде сеточного макрографа – одномерного, двумерного или трехмерного.

А каждая сеточная подобласть, в свою очередь, может быть изображена с помощью сеточного двунаправленного графа (структурированного или неструктурированного), пример которого представлен на рис. 2, где каждое ребро соответствует матричным элементам $a_{i,k}$ и $a_{k,i}$ матрицы A.

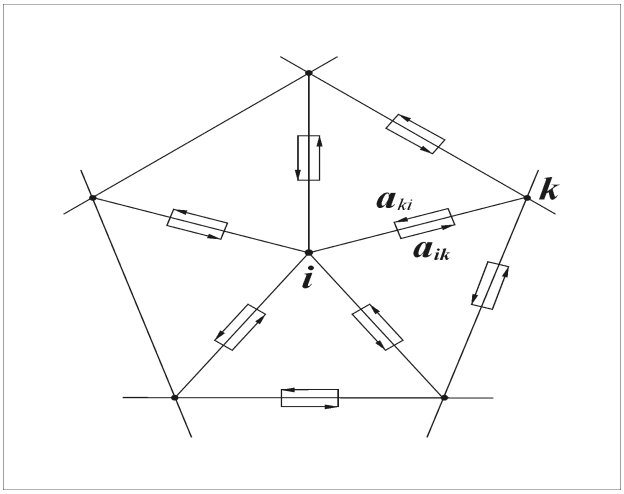


Рис. 2. Фрагмент пространственной сетки

Идеально отображение сеточных алгоритмов на архитектуру МВС достигается в том случае, когда последняя составляет вычислительную сеть, представимую в виде аналогичного сетевого графа, в котором узел представляет многоядерный процессор с собственной памятью, а ребро выполняет и вычислительные, и коммутирующие функции по соединениям соседних узлов, т.е. принадлежащих одному сеточному шаблону.

Оптимизация массивного распараллеливания в конкретных случаях требует анализа коэффициентов ускорения и использования

$$S_p = T_1 / T_p, \quad E_p = S_p / p, \tag{11}$$

где T_p – время решения задачи (или реализации алгоритма) на p процессорах. К сожалению, сделать это непросто в силу отсутствия адекватных моделей машинных вычислений на МВС с иерархической памятью.

Поскольку главное снижение производительности многопроцессорных компьютеров происходит из-за коммуникационных потерь, с точки зрения эффективности решения рассматриваемых задач перспективными являются следующие направления развития архитектур МВС:

- вычислительные сети (ВС) различной размерности (одно-, дву- и трехмерные);
- динамическая реконфигурация ВС с реализацией как структурированных, так и неструктурированных сеток;
- быстрые ближние связи и некоторые специальные коммуникации (типа гиперкуба);
- синхронизация обменов и вычислений, совмещение во времени двусторонних обменов;

- специализированные вычислительные устройства и гетерогенные МВС (например, аппаратная поддержка постобработки, визуализации и алгоритмов линейной алгебры).

С точки зрения концепции прикладного математического и программного обеспечения кардинальным представляется создание автономных библиотек алгоритмов и инструментариев, поддерживающих различные технологические этапы моделирования (генераторы сеток, аппроксиматоры) на основе разработки согласованных структур данных (геометрических, сеточных, алгебраических, графических). Такой модульный принцип позволит осуществить независимую реализацию, развитие и переиспользование продуктов коллективов разработчиков, а также их адаптацию на вновь появляемым платформам МВС. Кроме того, эта модель ориентирована на сборку конкретных приложений из представительного набора программных блоков наподобие детского конструктора, что должно обеспечить такие трудно совместимые категории, как универсальность и эффективность. Разумеется, достижение этой цели требует координации усилий вычислительного сообщества, но примеры успешного сотрудничества такого рода уже имеются: в области линейной алгебры общепринятыми являются форматы и стили пакетов BLAS, SPARSE BLAS, а также коллекции типовых матриц для сравнительного тестирования.

Литература

1. Годунов С.К., Роменский Е.И. Элементы механики сплошных сред и законы сохранения.–Новосибирск, Научная книга, 1998.
2. Hiptmair R. Finite elements in computational electromagnetism.–Acta Numerica, Cambridge Univ. Press, 2002, 237-339.
3. Дородницын В.А. Групповые свойства разностных уравнений.–М., Физматлит, 2001.
4. Франк А.М. Дискретные модели несжимаемой жидкости.–М., Физматлит, 2001.
5. Чиркунов Ю.А. Групповой анализ линейных и квазилинейных уравнений.–Новосибирск, НГУЭУ – НИНХ, 2007.
6. Воеводин В.В. Математические основы параллельных вычислений.–М., МГУ, 1991.
7. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления.–С.-Пб., “БХВ-Петербург”, 2002.
8. Asanovic K., et. al. The Landscape of Parallel Computing Research: A View from Berkeley. Techn. Rep. UCB/EECS – 2006–183.
9. Asanovic K., et. al. The Parallel Computing Laboratory at U.C.Berkeley: A Research Agenda Based on the Berkeley View. Techn. Rep. UCB/EECS – 2008–23.
10. Dongarra J., Beckman P., et. al. – IESP: International Exascale Software Project. Read Map, 18 Nov., 2009, www.exascale.org.
11. Марчук Г.И., Дымников В.П. и др. Математическое моделирование общей циркуляции атмосферы и океана.–Л., Гидрометеоиздат, 1984.
12. Голосов И.С., Горбенко Н.И., Гурьева Я.Л., Ильин В.П. и др. Комплексное моделирование технологических процессов в алюминиевом электролизере.–В сб.: “Актуальные научно-технические проблемы алюминиевой промышленности России”.–М., изд. ИГЕМ РАН, 2003, 72-80.
13. Ильин В.П. О численном решении прямых и обратных задач электромагнитной георазведки.–Сиб.Ж.Выч. Мат., т. 6, № 4, 2003, 381-394.