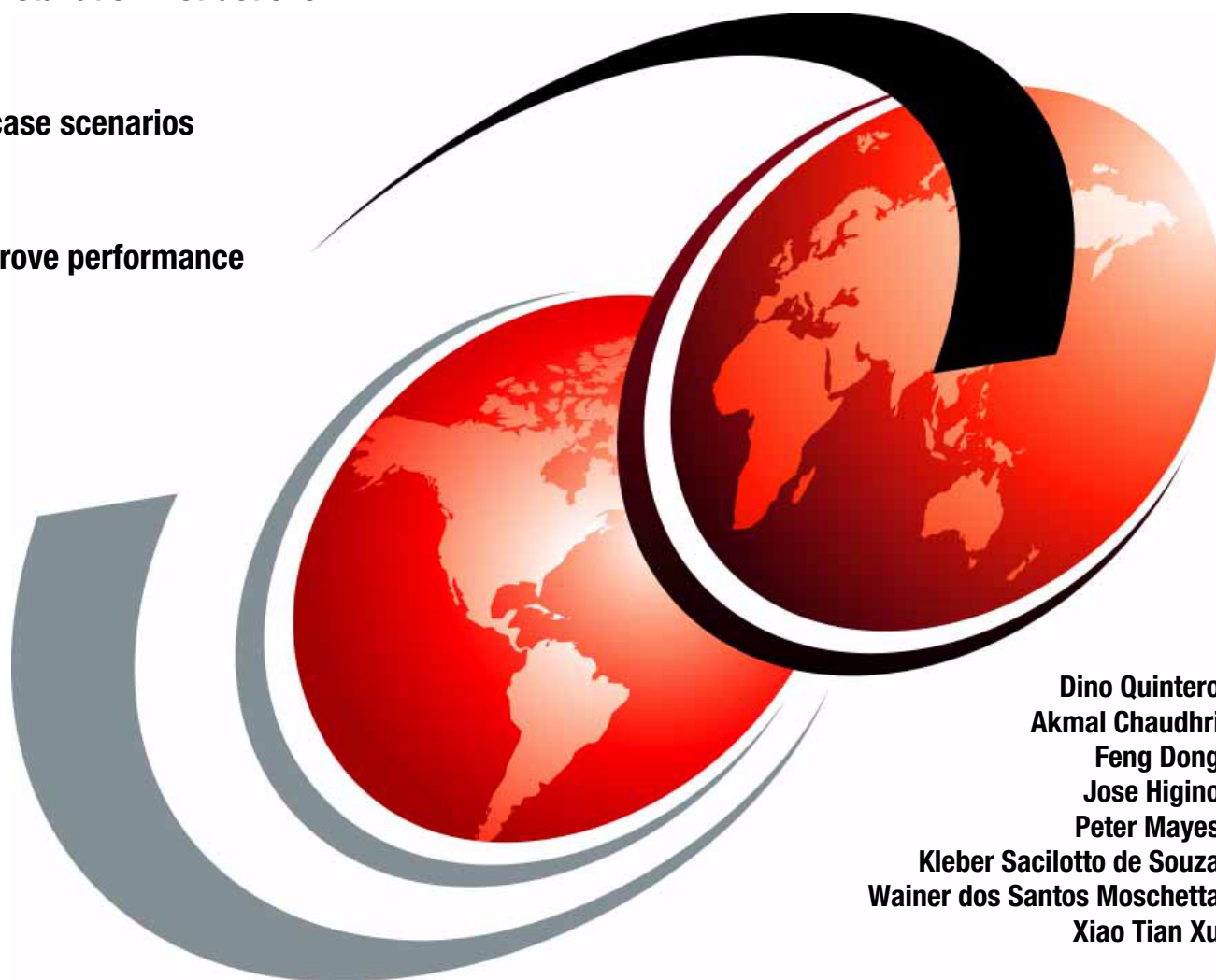


IBM Parallel Environment (PE) Developer Edition

Provides installation instructions

Includes case scenarios

Helps improve performance



Dino Quintero
Akmal Chaudhri
Feng Dong
Jose Higino
Peter Mayes
Kleber Sacilotto de Souza
Wainer dos Santos Moschetta
Xiao Tian Xu

Redbooks



International Technical Support Organization

IBM Parallel Environment (PE) Developer Edition

February 2013

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (February 2013)

This edition applies to the following environments:

AIX:

IBM AIX 7.1 TL1 SP5, IBM GPFS 3.5.0.4, IBM TWS LoadLeveler 5.1.0.7, IBM Parallel Environment 1.1.0.8, IBM Parallel Environment Developer Edition 1.2.0.0, IBM XL Fortran Compilers 14.1.0.1, IBM XL C/C++ Compilers 12.1.0.1, git-4.3.20-4 and readline-4.3-2 from AIX Toolbox.

RHEL (on Power):

RHEL 6.2, IBM GPFS 3.5.0.3, IBM TWS LoadLeveler 5.1.0.10, IBM Parallel Environment 1.2.0.7, IBM Parallel Environment Developer Edition 1.2.0.0, IBM XL Fortran Compilers 14.1, IBM XL C/C++ Compilers 12.1, git-1.7.1-2.el6_0.1.ppc64, environment-modules-3.2.7b-6.el6.ppc64. All required dependencies downloaded from Red Hat Enterprise Linux 6.2 repositories.

SLES (on x86):

SLES 11 SP1 (x86_64), IBM GPFS 3.4.0-11, IBM TWS LoadLeveler 5.1.0.11, IBM Parallel Environment 1.2.0.7, IBM Parallel Environment Developer Edition 1.2.0.0, git-core-1.6.0.2-7.26, compiled source of environment-modules-3.2.8 with GCC 4.3-62.198 (in SLES 11 SP1). All required dependencies downloaded from SUSE Linux Enterprise Server 11 SP1 repositories.

© Copyright International Business Machines Corporation 2013. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team who wrote this book	ix
Now you can become a published author, too!	xi
Comments welcome	xi
Stay connected to IBM Redbooks	xi
Chapter 1. Introduction	1
1.1 Overview	2
1.2 Features	3
1.3 Supported operating systems (software)	3
1.4 Supported hardware	4
1.5 Eclipse basics	4
1.5.1 What is Eclipse?	4
1.5.2 Eclipse architecture	4
1.5.3 Eclipse terms and concepts	5
1.6 Project types	6
Chapter 2. Component tour	9
2.1 Parallel Environment Runtime Edition components	10
2.1.1 Parallel Operating Environment (POE)	10
2.1.2 IBM Message Passing Interface (IBM MPI)	11
2.1.3 IBM Parallel Active Messaging Interface (PAMI)	11
2.1.4 Low-level application programming interface (LAPI)	12
2.1.5 Command line parallel debugger (pdb)	12
2.2 Parallel Environment Developer Edition components	13
2.2.1 Eclipse, PTP, CDT and Photran	13
2.2.2 IBM specific add-ons in the IBM PE Developer Edition	14
Chapter 3. Scenarios	17
3.1 Cluster design overview	18
3.1.1 HPC scenario based on IBM POWER Systems platform	18
3.1.2 HPC scenario based on x86 platform	24
3.1.3 IBM additional other software components for HPC solution	28
3.2 Developing parallel applications	33
3.2.1 Programming models	34
3.2.2 Most used frameworks and libraries	35
Chapter 4. Server installation	47
4.1 Software requirements	48
4.2 PEDE packaging considerations	48
4.2.1 Package contents	49
4.2.2 Additional software	50
4.3 Installation	52
4.3.1 AIX 7.1	53
4.3.2 RHEL 6 (on IBM POWER)	53
4.3.3 SLES 11 SP2 or RHEL 6.2 (x86_64)	53

4.4	Post-installation set up	54
4.4.1	Quick Parallel Environment Runtime tuning	54
4.4.2	GPFS tunable parameters affecting HPC performance	56
4.4.3	HPC Cluster verifications	56
4.4.4	Customizing the environment	56
Chapter 5. Managing projects using the Eclipse and PTP framework		61
5.1	Available project scenarios	62
5.1.1	Synchronized	62
5.1.2	Remote	64
5.1.3	Local	65
5.2	Creating a new parallel application	66
5.3	Importing an existing parallel application	70
5.4	Building and running an application	71
5.4.1	Building (using targets)	73
5.4.2	Running	78
5.5	Edit features of Eclipse	85
5.6	Debugging using Eclipse	91
5.7	Integrating external applications	91
Chapter 6. Parallel Environment Developer Edition tools		95
6.1	Tuning tools	96
6.1.1	Preparing the application for profiling	100
6.1.2	Creating a profile launch configuration	104
6.1.3	Hardware Performance Monitoring	107
6.1.4	MPI profiling and trace	111
6.1.5	OpenMP profiling	115
6.1.6	I/O profiling	117
6.1.7	X Windows Performance Profiler	130
6.2	Debugging	133
6.2.1	Parallel Static Analysis	133
6.2.2	Eclipse PTP Parallel debugger	138
Chapter 7. Application profiling and tuning		141
7.1	Profiling and tuning hints for sPPM	142
7.1.1	A glance at the application sPPM	142
7.1.2	Use gprof to view the profiling data	142
7.1.3	Using binary instrumentation for MPI profiling	145
7.1.4	Use OpenMP profiling to identify if the workload for threads	147
7.2	Profiling and analyzing Gadget 2	149
7.3	Analyzing a ScaLAPACK routine using the HPCT Toolkit	163
7.3.1	The Structure of ScaLAPACK	163
7.3.2	The build process	164
7.3.3	CPU profiling	164
7.3.4	HPM profiling	169
7.3.5	MPI profiling	173
Appendix A. HPC Toolkit environment variables		179
	General environment variables	180
	Variables related to hardware performance counters	180
	Variables related to MPI profiling	181
	Variables related to I/O profiling	182
	Variables relating to OpenMP profiling	182

Related publications	185
Online resources	185
Help from IBM	185

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Active Memory™	IBM®	PowerPC®
AIX®	iDataPlex®	PowerVM®
BladeCenter®	Informix®	POWER®
Blue Gene/L®	Intelligent Cluster™	PureFlex™
Blue Gene/P®	LoadLeveler®	Redbooks®
Blue Gene®	Power Systems™	Redbooks (logo)  ®
GPFS™	POWER6®	System x®
IBM Flex System™	POWER7®	Tivoli®
IBM PowerLinux™	PowerLinux™	

The following terms are trademarks of other companies:

Intel Xeon, Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

This publication helps strengthen the position of IBM® software solutions and enables for High Performance Computing (hardware, software, and tools) with a well-defined and documented deployment model within an IBM environment. As a result, customers receive a planned foundation for dynamic infrastructure for parallel High Performance Computing (HPC) applications.

This IBM Redbooks® publication addresses topics to take advantage of the strengths of IBM PE Developers Edition for HPC applications. The objective is to solve customer's challenges and maximize systems' throughput, performance, and management. This publication examines the tools, utilities, documentation, and other resources available to help the IBM technical teams provide solutions and support for IBM HPC solutions in an IBM hardware environment.

This IBM Redbooks is targeted toward technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) responsible for providing HPC solutions and support.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Dino Quintero is an IBM Senior Certified IT Specialist with the ITSO in Poughkeepsie, NY. His areas of knowledge include enterprise continuous availability, enterprise systems management, system virtualization, and technical computing, and clustering solutions. He is currently an Open Group Distinguished IT Specialist. Dino holds a Master of Computing Information Systems degree and a Bachelor of Science degree in Computer Science from Marist College.

Akmal Chaudhri is Senior IT Specialist in IBM United Kingdom (UK). Akmal has worked with database technology since 1988. He worked in a variety of roles, covering development, consulting, and product strategy with Reuters, Logica, Computer Associates, IBM Informix®, and IBM. He published and presented widely about Java, XML, and Database-related topics. He also edited and co-edited ten books and is now working on a new book about NoSQL technology. He worked extensively with the academic community on a world-wide basis through community outreach programs at Informix and IBM. He holds a Bachelor of Science(1st Class Hons.) in Computing and Information Systems, Master of Science in Business Systems Analysis and Design, and a PhD in Computer Science. He is a Member of the British Computer Society (MBCS) and a Chartered IT Professional (CITP).

Feng Dong is a Senior IT Specialist in IBM China. He has six years of experience in the High Performance Computing field. He has worked at IBM for 14 years. His areas of knowledge include System x, BladeCenter®, and IBM Power Systems™, rich experience for Linux HPC cluster deployment and performance tuning. From 2010, he work with IBM World Wide Benchmark Team, focus on HPC benchmark testing. He has a Bachelor degree of Compute Science from Beijing University of Technology, in China.

Jose Higino is a Infrastructure IT Specialist doing AIX/Linux support and services for IBM Portugal. His areas of knowledge include System x, IBM BladeCenter® and Power Systems planning and implementation, management, virtualization, consolidation and clustering (HPC

and HA) solutions. He is currently the only person responsible for Linux support and services in IBM Portugal. He completed the Red Hat Certified Technician level in 2007, became a CiRBA Certified Virtualization Analyst in 2009, and completed certification in the KT Resolve methodology as a Subject Matter Expert (SME) in 2011. José holds a Master of Computers and Electronics Engineering degree from Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia (UNL - FCT), in Portugal.

Peter Mayes is a Senior IT Specialist in the United Kingdom (UK). He has 26 years of experience in the field of high-performance computing. He has a Masters degree in Mathematics, Masters of Science in Mathematical Modeling and Numerical Analysis, and DPhil in Engineering Mathematics, all from the University of Oxford. His areas of expertise is FORTRAN programming, particularly for high-performance and parallel computers.

Kleber Sacilotto de Souza is a Software Engineer working at the IBM Linux Technology Center (LTC) in Brazil. He has more than five years of experience in Linux development and tests and is currently working with Linux device drivers for I/O devices for the IBM Power Systems. He holds a Bachelor of Science degree in Computer Science from State University of Campinas (UNICAMP), Brazil. His areas of expertise include C development, Linux device drivers, and tests automation.

Wainer dos Santos Moschetta is a Staff Software Engineer at the IBM Linux Technology Center (LTC) in Brazil. He has more than six years of experience in several areas of Linux and open source ecosystems. He is a Certified Associate in Project Management (PMI-CAPM) and currently leads the development of the IBM SDK for PowerLinux™. He holds a Bachelor of Science degree in Computer Science from the University of São Paulo (USP), Brazil. His areas of expertise are IBM PowerLinux™, automated test/build systems, Linux device drivers, development of tools for application migration, and construction of modern Integrated Development Environments (IDE) leveraging open source.

Xiao Tian Xu is a Software Engineer in IBM China. He has two years of experience in parallel computing and High Performance Computing fields, worked on the code tuning for the numerical weather predicting project of Chinese meteorological office, and did system testing on the IBM HPC software stack side for BGQ, SuperMUC, and Percs projects. He has a degree in High Performance Computing from Edinburgh Parallel Computing Center, the University of Edinburgh, UK. Thanks to the following people for their contributions to this project:

Ella Buslovic, Richard Conway
International Technical Support Organization, Poughkeepsie Center

Beth Tibbitts, Brian Watt, Chulho Kim, Dave Wootton, Gregory Watson, John Robb, and
KaTrina Love
IBM USA

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at: ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>



Introduction

This chapter provides an introduction to the IBM Parallel Environment (PE) Developer Edition (DE).

In this chapter, the following topics are discussed:

- ▶ Overview
- ▶ Features
- ▶ Supported operating systems (software)
- ▶ Supported hardware
- ▶ Eclipse basics
- ▶ Project types

1.1 Overview

IBM PE Developer Edition provides an Integrated Development Environment (IDE) that combines open source tools, and IBM-developed tools to fulfill the following requirements:

- ▶ Editing, compiling, running, and debugging an application
- ▶ Performing static analysis of an application to find programming problems
- ▶ Analyzing the performance of serial and parallel applications
- ▶ Interactively running and debugging parallel applications
- ▶ Submitting batch serial or parallel jobs to IBM LoadLeveler®

IBM PE Developer Edition consists of two major integrated components:

- ▶ A client workbench that runs on a desktop or notebook computer
- ▶ A server that runs on select IBM Power systems, IBM PureFlex™ Systems servers, IBM System x® systems and iDataPlex® servers

The client workbench is built-upon the Eclipse platform and is designed to provide a set of tools to improve the productivity of high performance and technical computing developers.

The client component contains the Eclipse IDE, Parallel Tools Platform (PTP), and client code for the IBM High Performance Computing (HPC) Toolkit. The server component contains instrumentation libraries and tools for the IBM HPC Toolkit and X11-based GUI tools that can be used, if necessary, in place of the client-side IBM HPC Toolkit plug-in.

Figure 1-1 shows the overall architecture of the IBM PE Developer Edition.

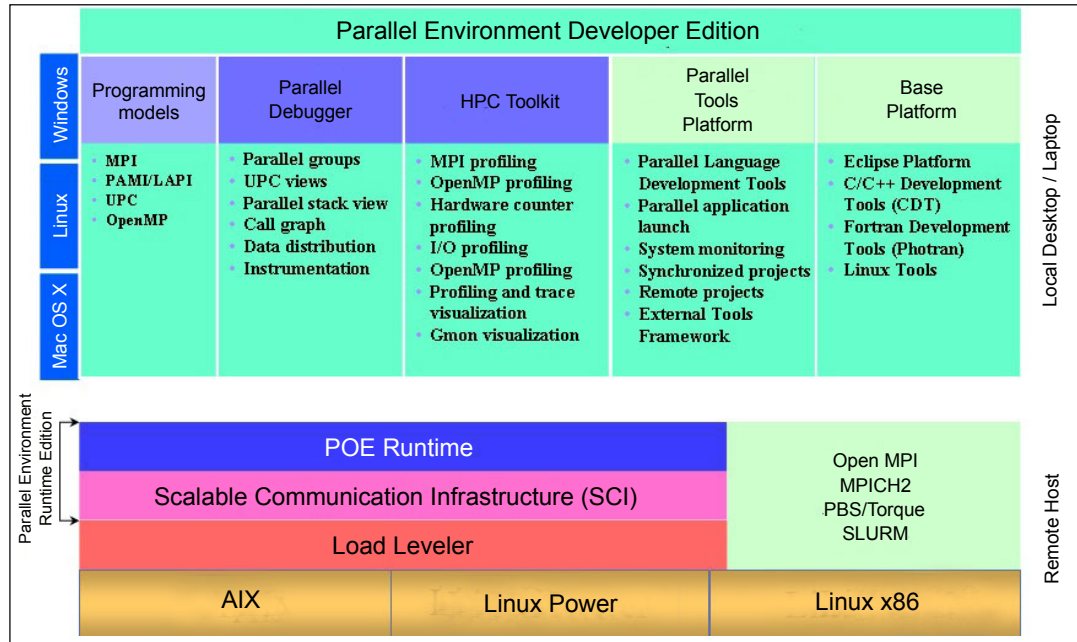


Figure 1-1 Architecture of IBM PE Developer Edition

We refer to many of the components in Figure 1-1 throughout this book, and we use examples to show in detail how to use the client and server tools in later chapters.

A detailed discussion of Eclipse is beyond the scope of this book, and there are many good tutorials available that describe Eclipse in depth. However, a brief introduction to Eclipse,

which includes basic terminology, is provided later in this chapter. Where Eclipse features are required to use IBM PE Developer Edition, these are described in detail in this book.

1.2 Features

IBM PE Developer Edition combines a number of tools to provide a complete development environment for high performance and technical computing. These tools support features that enable developers to perform the complete range of programming tasks, such as editing, compiling, running and debugging serial and parallel applications. Supporting these features directly within the workbench results in improved developer productivity.

Developers can use the following standard Eclipse features:

- ▶ Code assist
- ▶ Interactive code completion
- ▶ Syntax-directed highlighting of source code
- ▶ Built-in help information, including hover help, to assist developers as they are writing code

Developers can use the following PTP features:

- ▶ Edit and compile applications on remote systems
- ▶ Use source code cross referencing tools
- ▶ Perform static analysis of source code
- ▶ Run applications on remote systems using a variety of job schedulers and runtime systems

Developers can also use the following integrated performance analysis tools features:

- ▶ Analyze application code and locate performance problems
- ▶ Analyze resource usage
- ▶ Perform MPI profiling and tracing
- ▶ Perform Input/Output (I/O) profiling

We show examples of these features in later chapters of this book.

1.3 Supported operating systems (software)

IBM PE Developer Edition requires installation in two parts: on the server and on the client. In this section, we list the supported operating systems for the server and client components.

The software requirements for the IBM PE Developer Edition server component is one of the following:

- ▶ Red Hat Enterprise Linux (RHEL) v6.2
- ▶ SUSE Linux Enterprise Server (SLES) v11 SP1
- ▶ IBM AIX® v7.1

For Parallel Operating Environment (POE) jobs, IBM PE Runtime Edition for Linux on x86 Architecture v1.2 is required. Additionally, for batch jobs, LoadLeveler for Linux on x86 Architecture v5.1 is required.

The software requirements for the IBM PE Developer Edition client component are one of the following items:

- ▶ Microsoft Windows XP, 32 bit
- ▶ Microsoft Windows 7, 32 bit or 64 bit
- ▶ RHEL 5, 32 bit or 64 bit
- ▶ RHEL 6, 32 bit or 64 bit
- ▶ SLES 10, 32 bit or 64 bit
- ▶ SLES 11, 32 bit or 64 bit
- ▶ Apple Mac OS X 10.6, 64 bit
- ▶ Apple Mac OS X 10.7, 64 bit

1.4 Supported hardware

For server side systems, the supported hardware includes:

- ▶ IBM Power Systems or IBM PureFlex compute nodes, based on POWER7® technology, with supported adapters and switch
- ▶ Stand-alone POWER7 clusters or POWER7 clusters connected with a LAN supporting IP
- ▶ IBM System x iDataPlex
- ▶ IBM Intelligent Cluster™
- ▶ Select IBM System x servers

For clients systems, the supported hardware includes:

- ▶ Any hardware capable of running Java 1.6 or later on the supported operating systems listed in the supported operating systems (software) section with a minimum of 2 GB memory; 4 GB memory and 500 MB free disk space is recommended.

1.5 Eclipse basics

The Eclipse framework provides the foundation for the IBM PE Developer Edition. In this section, we provide a brief introduction to Eclipse.

1.5.1 What is Eclipse?

Eclipse is a technology platform, originally an IDE. Today, Eclipse is used as the foundation for many different types of tools. Eclipse is also an Open Source Project. The Eclipse Public License is Open Source Initiative (OSI) certified. The Eclipse community is also an important part of the platform, and there are hundreds of companies and thousands of programmers working to improve and extend Eclipse all the time.

1.5.2 Eclipse architecture

Figure 1-2 on page 5 shows a high-level architectural view of Eclipse. We can develop stand-alone Rich Client Platform (RCP) applications. We can also use many different development tools, such as the C Development Tools (CDT) that extend the Eclipse platform, and we can also develop additional plug-ins using the Plug-in Development Environment (PDE).

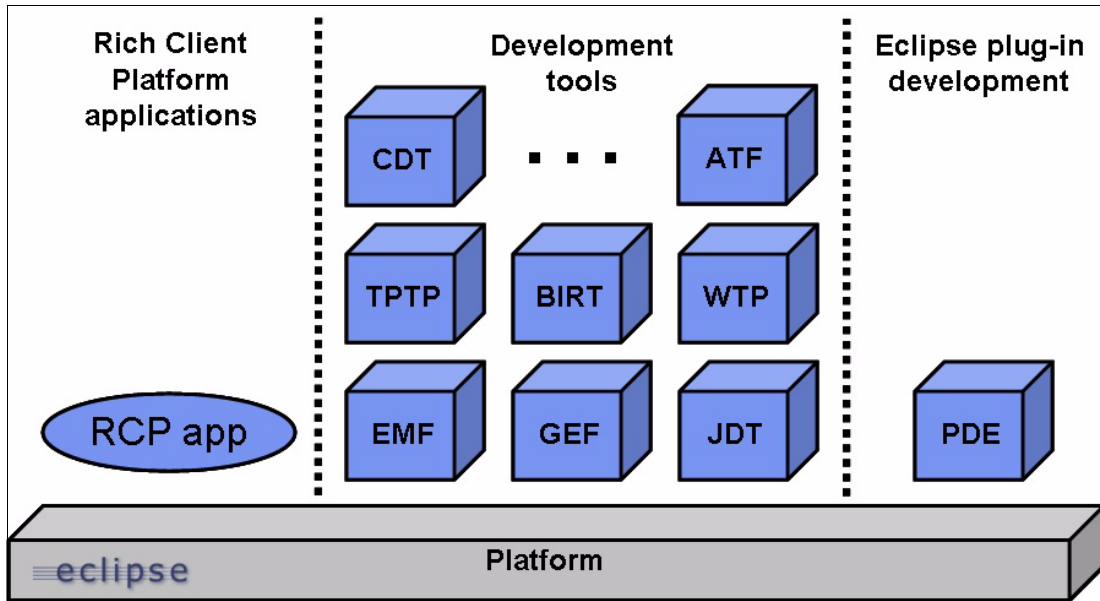


Figure 1-2 High-level Eclipse architecture

Figure 1-3 shows core components of the Eclipse platform, such as the workbench, help system, and team components. In later chapters, we document several plug-ins provided with the IBM PE Developer Edition.

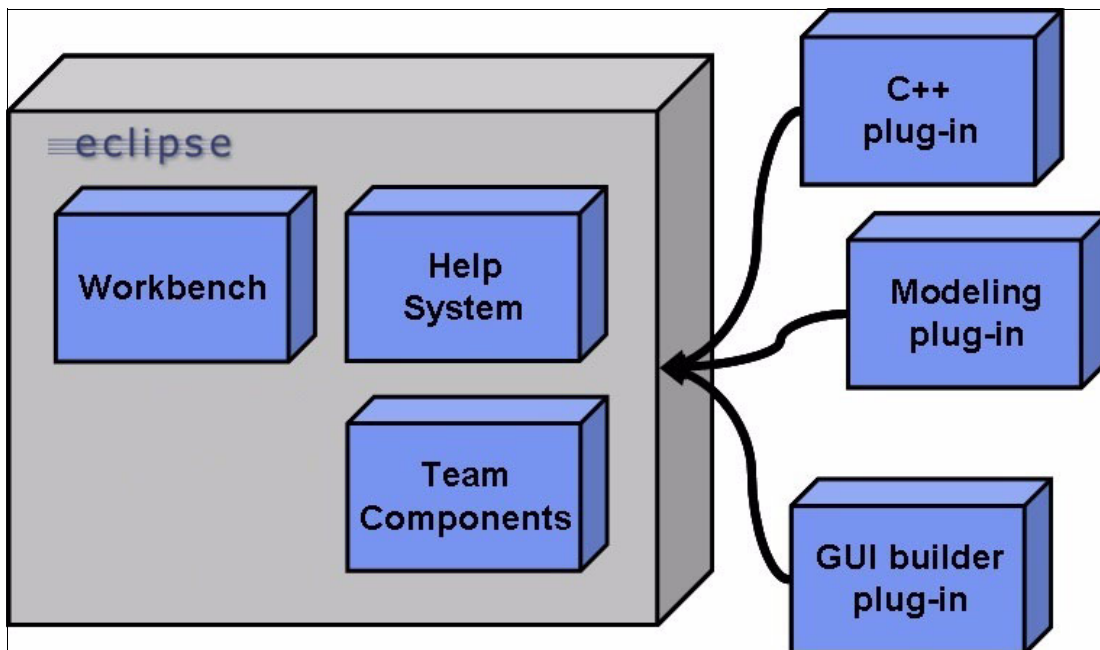


Figure 1-3 Eclipse components and some example plug-ins

1.5.3 Eclipse terms and concepts

In this section, we describe some key Eclipse concepts:

Workspace

The workspace is a physical location on a computer. It contains projects, folders, and user files. For IBM PE Developer Edition users, it contains their application source files, makefiles, and so on. A project

is a collection of folders and files for a particular task. A folder is a subdirectory and can contain folders and other files. A file is just a file. In other words, a workspace is just a directory. The first time we run Eclipse, we are asked to choose a workspace. We can also create many different workspaces. To switch to another workspace, use **File** → **Switch Workspace** in the Eclipse main menu.

Workbench	The workbench is the Eclipse environment. When we open Eclipse, we see a workbench that displays the resources in a particular workspace. A resource is a file, such as an application source file. A workbench contains perspectives, views, and editors.
Perspective	A perspective is a set of views organized for a particular task. If we are writing C/C++ or Fortran code, certain views are useful. If we are debugging C/C++ or Fortran code, other views are useful. A perspective lets us arrange the views we need. IBM PE Developer Edition comes with several perspectives installed. If we install plug-ins for various tasks, other perspectives might appear. We can customize a perspective in many ways. For example, we can change the views that are part of a perspective, or we can change how those views are arranged on the window. Use Window → Customize Perspective to define which menu items and toolbar buttons are available for a given perspective.
View	A view is a window on some resources in our workspace. For example, one view might list all the files in our workspace. We can open and close and arrange views any way we like. Two views might display the same thing in different ways. Use Window → Show View to open a view at any time. Use Window → Show View → Other to see a complete list of views. Views can also be stacked.
Editors	We use an editor to modify a resource. There are different kinds of editors for different kinds of resources. Like all views, editors can be stacked.
Preferences	Eclipse has a number of preferences that we can set. Preferences are settings, such as default values for options in Eclipse or preferences that can control the behavior of Eclipse components, such as the behavior of views. If we install a new plug-in, that plug-in can also have preferences. As a result, our Eclipse installation can have many settings. Choose Window → Preferences to get to the preferences dialog.

1.6 Project types

IBM PE Developer Edition supports three project types:

Local	A local project is where source code is held only on the local machine. With a local project, building an application and launching an application usually occurs only on the local machine.
Synchronized	A synchronized project is where source code is held on both the local and remote machines and synchronized between the two. Building an application and launching an application occur either locally or remotely. Since the source code is available locally, editing can use all the Eclipse workbench features, and response is fast. Work can

continue locally even if the network connection to the remote machine is lost.

Remote

A remote project is where source code is held only on the remote machine. Building an application and launching an application occur only on the remote machine. Source code is only brought to the local machine in an editor buffer when a file is edited. A reliable and consistent network connection is required to work easily with remote projects, but all the files do not have to be mirrored on the local machine.

In this book, we mainly focus on synchronized projects to demonstrate the flexibility of the IBM PE Developer Edition tools.

Furthermore, for C/C++ and Fortran projects, we can use two build types:

Makefile

A makefile project contains one or more makefiles for building an application. The user provides the makefile. Actually, any type of command can be used for the build. Scripts can call a collection of tools, including makefiles. A Makefile project just means that we control the build commands.

Managed

A managed project does not require a makefile, and Eclipse manages the application build process. Eclipse writes a makefile *under the covers*.



Component tour

In this chapter, we describe the client and server components that comprise the IBM Parallel Environment Developer Edition. We also describe the IBM Parallel Environment Runtime Edition, which is a companion to the Developer Edition.

This chapter contains the following sections:

- ▶ Parallel Environment Runtime Edition components
- ▶ Parallel Environment Developer Edition components

2.1 Parallel Environment Runtime Edition components

IBM Parallel Environment (PE) Runtime Edition is a capability-rich development and execution environment for parallel applications. IBM PE Runtime Edition offers parallel application programming interfaces and execution environment for parallel applications.

Parallel Environment Runtime Edition includes the following components:

- ▶ The Parallel Operating Environment (POE) for submitting and managing jobs.
- ▶ The IBM MPI, PAMI, and LAPI libraries for communication between parallel tasks.
- ▶ A parallel debugger (pdb) for debugging parallel programs.

2.1.1 Parallel Operating Environment (POE)

The IBM Parallel Operating Environment (POE) enables users to develop and execute parallel applications across multiple operating system images (nodes). POE includes parallel application compile scripts for programs written in C, C++, and Fortran, and a command-line interface to submit commands and applications in parallel. POE also provides an extensive set of options and additional functions to fine-tune the application environment to suit the execution of the application and system environment.

POE is used to set up the environment for the user's parallel program and to control and monitor job execution.

The Parallel Operating Environment provides:

- ▶ OS Jitter mitigation through co-scheduling capability: POE provides services for periodically adjusting the dispatch priority of a user's task between set boundaries, giving the tasks improved execution priority. As an alternative to the priority adjustment coscheduler, POE provides a run queue-based coscheduler for reducing operating system Jitter (OS Jitter). OS Jitter is operating system interference that is caused by the scheduling of daemon processes and the handling of asynchronous events, such as interrupts. This interference can have adverse effects on parallel applications running on large-scale systems.
- ▶ Affinity support of processes to CPU and memory: IBM PE Runtime Edition provides the capability to control the placement of the tasks of a parallel job using affinity to process memory or CPU placement during its execution. As a result, applications might see improved performance if the processor, the memory it uses, and the I/O adapter it connects to are in close proximity based on the affinity of the tasks to memory or CPUs it uses.
- ▶ User-controlled workflow/subjob support: IBM PE Runtime Edition provides support for launching and managing multiple parallel dynamic jobs (subjobs) using a single scheduler or resource management allocation of cluster resources.
- ▶ Lightweight core file support: Lightweight core file is designed to save CPU time, network bandwidth, and disk space that is required to generate standard core files. The lightweight format also provides the capability to easily examine the state of all threads in a parallel program at the time the event that caused the core file occurred.
- ▶ Serial and parallel job launch: Allows POE to be used not only for launching traditional MPI or other message passing programs, but also as a distributed shell to quickly obtain information about all nodes in the cluster, such as disk space, currently running jobs, and so on.
- ▶ Support running SPMD and MPMD programs: Single Process, Multiple Data (SPMD), Multiple Process, and Multiple Data (MPMD) gives users the flexibility to run the same

program on all nodes (SPMD, most common) or different programs on each node (MPMD). The MPMD function is useful for master/worker programs where the master program coordinates and synchronizes the execution of all worker tasks, where neither program can run without the other.

- ▶ Resource management: POE supports running without a separate scheduler or resource manager. If a scheduler or resource manager is not used or not available, POE can manage the node and adapter resources itself. The network resource tables is also loaded by POE to establish the communication mechanism needed for message-passing programs.

You can use the resource manager of your choice for submitting and managing batch or interactive parallel jobs. IBM PE Runtime Edition includes a set of resource management interfaces and data areas for configuring your resource manager to interact with POE.

Integration with IBM Tivoli® Workload Scheduler LoadLeveler: IBM PE Runtime Edition allows users to run POE jobs in interactive or batch mode with LoadLeveler managing the node, network, CPU, memory, and other key resources for optimum throughput and resource utilization.

2.1.2 IBM Message Passing Interface (IBM MPI)

The IBM MPI is a complete MPI 2.2 implementation, designed to comply with the requirements of the MPI standard. IBM MPI supports the MPI-2.1 process creation and management scheme. The IBM design is enabled using static resources allocated at job launch time.

The IBM MPI provides a number of nonblocking collective communications subroutines that are available for parallel programming. These subroutines are extensions of the MPI standard. Collective communications routines for 64-bit programs were enhanced to use shared memory for better performance. The IBM MPI collective communication is designed to use an optimized communication algorithm according to job and data size.

The IBM MPI provides a high scalability and low memory usage implementation. The IBM MPI library minimizes its own memory usage so that an application program can use as much system resources as possible. It is architected to support parallel job size of up to one million tasks.

IBM MPI runs over PAMI. This is achieved through exploiting the PAMI APIs.

Note: For Linux, PE Runtime Edition also includes an MPICH2 MPI implementation that can be used as an alternative to the IBM MPI implementation.

2.1.3 IBM Parallel Active Messaging Interface (PAMI)

IBM PAMI is a converged messaging API that covers both point-to-point and collective communications. PAMI exploits the low-level user space interface to the Host Fabric Interface (HFI) and TCP/IP using UDP sockets.

PAMI has a rich set of collective operations designed to support MPI and pGAS semantics, multiple algorithm selection, and nonblocking operation. It supports nonblocking and ad hoc geometry (group/communicator) creation and nonblocking collective allreduce, reduce, broadcast, gather(v), scatter(v), alltoall(v), reduce scatter, and (ex)scan operations. The geometry can support multiple algorithms, including hardware-accelerated (through HFI Collective Acceleration Unit, or Barrier Service Register) versions of broadcast, barrier, allreduce, and reduce.

2.1.4 Low-level application programming interface (LAPI)

The *low-level application programming interface* (LAPI) is a message-passing API that provides a *one-sided communication model*. In this model, one task initiates a communication operation to a second task. The completion of the communication does not require the second task to take complementary action.

The LAPI library provides basic operations to “put” data to and “get” data from one or more virtual addresses of a remote task. LAPI also provides an *active message* infrastructure. With active messaging, programmers can install a set of handlers that are called and run in the address space of a target task on behalf of the task originating the active message. Among other uses, these handlers can be used to dynamically determine the target address (or addresses) where data from the originating task must be stored. You can use this generic interface to customize LAPI functions for your environment.

Some of LAPI’s other general characteristics include:

- ▶ Flow control
- ▶ Support for large messages
- ▶ Support for generic non-contiguous messages
- ▶ Non-blocking calls
- ▶ Interrupt and polling modes
- ▶ Efficient exploitation of interconnect functions
- ▶ Even monitoring support (to simulate blocking calls, for example) for various types of completion events

LAPI is meant to be used by programming libraries and by power programmers for whom performance is more important than code portability.

MPI and LAPI provide communications between parallel tasks, enabling application programs to be *parallelized*.

MPI provides message passing capabilities that enable parallel tasks to communicate data and coordinate execution. The message passing routines call communication subsystem library routines to handle communication among the processor nodes.

LAPI differs from MPI in that it is based on an *active message style* mechanism that provides a one-sided communications model in which one process initiates an operation and the completion of that operation does not require any other process to take a complementary action. LAPI is also the common transport layer for MPI and is packaged as part of the AIX RSCT component.

2.1.5 Command line parallel debugger (pdb)

The parallel debugger (pdb) streamlines debugging of parallel applications, presenting the user with a single command line interface that supports most dbx/gdb execution control commands and provides the ability to examine running tasks. To simplify management of large numbers of tasks, dbx/gdb allows tasks to be grouped so that the user can examine any subset of the debugged tasks.

The pdb allows users to invoke a POE job or attach to a running POE job and place it under debug control. It starts a remote dbx/gdb session for each task of the POE job put under debugger control.

The pdb provides these advance features:

- ▶ Dynamic tasking support
- ▶ Multiple console display
- ▶ Output filtering

2.2 Parallel Environment Developer Edition components

IBM Parallel Environment Developer Edition is an Eclipse-based integrated set of application development tools that will help you develop, debug, and tune your parallel applications. It includes a set of standard Eclipse components and additional support for IBM environments.

2.2.1 Eclipse, PTP, CDT and Photran

IBM PE Developer Edition is based on the Eclipse 4.2 (Juno) platform and includes the following open-source components:

PTP

The Parallel Tools Platform (PTP) is an Eclipse-based application development environment that contains an integrated set of tools to help you edit, compile, run, debug, and analyze your parallel application written in C, C++, and Fortran. Advanced tools included with PTP include static analysis tools to locate errors before the code is compiled, refactoring tools to modify code while preserving behavior, and an integrated parallel debugger. PTP supports a broad range of architectures and job schedulers and provides the ability to easily add support for additional systems.

PTP also provides:

- ▶ Support for MPI, OpenMP, OpenACC, and UPC parallel programming models
- ▶ Support for a wide range of batch systems and runtime systems, including IBM LoadLeveler, IBM Parallel Environment, Open MPI, and MPICH2

IBM PE Developer Edition includes additional support for PAMI, LAPI, and OpenSHMEM libraries.

CDT

The C/C++ Development Tooling (CDT) provides a fully functional C and C++ IDE based on the Eclipse platform. Features include:

- ▶ Support for project creation and managed build for various toolchains
- ▶ Standard make build
- ▶ Source navigation
- ▶ Various source knowledge tools, such as type hierarchy
- ▶ Call graph
- ▶ Include browser
- ▶ Macro definition browser
- ▶ Code editor with syntax highlighting
- ▶ Folding and hyperlink navigation
- ▶ Source code refactoring and code generation
- ▶ Visual debugging tools, including memory, registers, and disassembly viewers

Photran

Photran is an IDE and refactoring tool for Fortran based on Eclipse and the CDT. Features include:

- ▶ Refactorings, such as rename, extract procedure, and loop transformations
- ▶ Syntax-highlighting editor
- ▶ Outline view
- ▶ Content assist
- ▶ Open declaration
- ▶ Declaration view and hover tips
- ▶ Fortran language-based searching
- ▶ Interactive debugger (gdb GUI)
- ▶ Makefile-based compilation
- ▶ Optional makefile generation
- ▶ Recognition of error messages from most popular Fortran compilers

2.2.2 IBM specific add-ons in the IBM PE Developer Edition

This section describes IBM specific add-ons in the IBM PE Developer Edition.

IBM HPC Toolkit

The IBM PE Developer Edition also includes the IBM HPC Toolkit (HPCT), which is a collection of tools that you can use to analyze the performance of parallel and serial applications that are written in C or Fortran, running the AIX or Linux operating systems on IBM Power Systems servers. Applications running on RedHat Enterprise Linux 6 on IBM System x with the Intel microarchitecture codename Nehalem, Westmere, and Sandy Bridge family of processors are also supported. The Xprof GUI also supports C++ applications. These tools perform the following functions:

- ▶ Provide access to hardware performance counters for performing low-level analysis of an application, including analyzing cache usage and floating-point performance.
- ▶ Profile and trace an MPI application for analyzing MPI communication patterns and performance problems.
- ▶ Profile an OpenMP application for analyzing OpenMP performance problems and to help you determine if an OpenMP application properly structures its processing for best performance.
- ▶ Profile application I/O for analyzing an application's I/O patterns and whether you can improve the application's I/O performance.
- ▶ Profile an application's execution for identifying hotspots in the application and for locating relationships between functions in your application to help you better understand the application's performance.

The IBM HPC Toolkit provides three primary interfaces. The first is the IBM HPC Toolkit Eclipse plug-in. This plug-in is an extension to the Eclipse IDE that you can use to run hardware performance counter analysis, MPI profiling and tracing, OpenMP profiling, and I/O profiling. The plug-in allows you to select the parts of your application that are to be instrumented, instrument those parts of the application, run the instrumented application, and view the resulting performance data. The plug-in allows you to sort and filter the data to help you find the performance problems in the application.

The second interface is peekperf, which is an AIX or Linux-based GUI that you can use to run hardware performance counter analysis, MPI profiling and tracing, OpenMP profiling, and I/O profiling. Peekperf allows you to select the parts of your application that are to be instrumented, instrument those parts of the application, run the instrumented application, and

view the resulting performance data. Peekperf allows you to sort and filter the data to help you find the performance problems in the application.

The third interface is Xprof, which you can use to view low-level profiling data for your application. Xprof allows you to view the performance data in gmon.out files, generated by compiling your application using the -pg compiler flag. You can view the profiling data, identify hotspots in the application, view relationships between functions in the application, zoom into areas of the application of greater interest, and sort and filter the data to help identify hotspots in the application.

The IBM HPC Toolkit also provides the hpcInst utility, which you can use to instrument the application without using the peekperf GUI. You can specify the types of instrumentation you want to use and the locations within the application that are to be instrumented. The hpcInst utility rewrites the application binary with the instrumentation you selected. Then, you can run the instrumented executable to obtain the same types of performance measurements that you can using peekperf.

Finally, the IBM HPC Toolkit provides commands to get an overview of hardware performance counters for an application and libraries that allow you to control the performance data obtained using hardware performance counters and by MPI profiling and tracing.

On x86 class machines running Linux, only the hardware performance counter tool, the MPI profiling tool, and the I/O profiling tool within the IBM HPC Toolkit are supported. The only supported model for these tools is where the application is linked with the supporting IBM HPC Toolkit runtime libraries. Accessing the hardware performance counters on x86 class machines is only supported on the Intel microarchitecture Nehalem, Westmere, and Sandy Bridge families of processors. There is no support for dynamic instrumentation using the **hpcInst** command, the peekperf GUI, or the IBM HPC Toolkit plug-in for Eclipse.

IBM PAMI and LAPI user assistance features

IBM PE Developer Edition provides features that augment the C/C++ Eclipse editor for ease of using PAMI and LAPI APIs, locating them in the source code, getting information about API usage and arguments, and so on.

IBM XLC and XLF compiler transformation report feedback viewer

IBM PE Developer Edition provides a source-linked view of items identified by the IBM XLC/XLF Compiler Transformation reports.



Scenarios

This chapter provides a general overview of High Performance Computing (HPC) clusters. This chapter also discusses and describes common components and introduces HPC solutions based on the IBM POWER® and x86 platforms including hardware and software stacks.

This chapter discusses the following topics:

- ▶ Cluster design overview
- ▶ Developing parallel applications

3.1 Cluster design overview

High Performance Computing clusters are designed to use parallel computing techniques to apply more processor power to develop a solution for a given problem. There are many examples of this in the scientific computing arena where multiple low-cost processors are used in parallel to perform a large number of operations. This is referred to as *parallel computing* or *parallelism*. In *How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters (Scientific and Engineering Computation)*, by Sterling, et al, parallelism is defined as “the ability of many independent threads of control to make progress simultaneously toward the completion of a task.”

An HPC cluster, as shown in Figure 3-1, is typically made up of a large number of nodes. All nodes are connected with each other through a high speed network, such as Gigabit Ethernet, 10-Gigabit Ethernet or InfiniBand networks, and shared parallel file system based on large capacity storage allows all nodes access to data simultaneously.



Figure 3-1 High Performance Computing Cluster

In this section, we introduce HPC environments based on IBM POWER Systems and an IBM iDataPlex (x86). We also introduce HPC scenarios based on these two platforms.

3.1.1 HPC scenario based on IBM POWER Systems platform

For many years IBM has provided High Performance Computing (HPC) solutions based on POWER platform. The IBM Power Systems platform is optimized for running highly parallel computationally intensive workloads and algorithms, such as weather and climate modeling, computational chemistry, and physics and petroleum reservoir modeling.

The next sections introduce IBM POWER Systems servers for HPC solution based on Power platform.

IBM Power 775

IBM announces the Power 775: A POWER7 processor-based POWER machine specially designed for large scale cluster computing, an extremely dense supercomputer node that boasts 256 POWER7 processor cores in just 2U of rack space. The Power 775 is the fourth generation of high-density compute nodes based on the POWER processor, following on from the original POWER5-based Power5 575 released in 2005. Target applications are highly parallel, high-performance computing (HPC) workloads, such as weather and climate modeling, computational chemistry, physics, computer-aided engineering, computational fluid dynamics, and petroleum exploration.

The Power 775 uses 8-core 3.84 GHz POWER7 processors packed four apiece into Quad Chip Modules (QCM). The system can support up to 256 processor cores and up to 2TB of memory in a slim 2U rack drawer. Each 2U drawer (CEC) contains eight QCMs so that each node has 256 cores. These QCMs are interconnected through copper switching technology. A maximum of twelve of these node drawers can be housed in custom water cooled rack along with optional 4U disk enclosures. Each rack can house 3072 POWER7 cores for a peak processing power of 96 TFLOPs. A Power 775 cluster can start with 512 cores (2 CEC) but is designed to scale out to hundreds of thousands of processors.

The system is logically configured into octant nodes with each octant containing thirty two (32) processor cores (one QCM). Each processor core runs at 3.84 GHz, and contains:

- ▶ Four Simultaneous Multithreading (SMT) execution threads per processor core.
- ▶ Two 128 bit VSX Vector units per processor core.
- ▶ Private 32 KB L1 cache, and private 256 KB L2 cache for each processor core.
- ▶ Private 4 MB L3 cache per processor core.
- ▶ Two integrated memory controllers and an Integrated I/O controller per processor chip.
- ▶ Integrated SMP coherence and data interconnect switch.
- ▶ Support logic for dynamic power management, dynamic configuration and recovery, and system monitoring.

In addition to the processors, each node also has 128 DIMM slots allowing for a total memory capacity of 2 TB using 16 GB DDR3 DIMMs and 16 PCIe Gen2-16x slots and one PCIe Gen2-8x slot. Disk storage is housed in optional 4U disk enclosures with up to 384 SAS and SSD 2.5" drives per drawer. Up to six disk enclosures can be included in a rack with two CEC drawers.

IBM Host Fabric Interface (HFI) is a new cluster interconnection network device designed for the IBM Power 775. The HFI provides the non-coherent interface between a POWER7 chip "quad" (QCM: 32-way SMP consisting of four POWER7 chips) and the clustered network. Four node drawers (8 node octants) were interconnected with HFI L-links, comprising a so called Super-node, while Super-nodes were interconnected with HFI D-links.

The IBM Power Systems 775 (9125-F2C) has several new features that make this system even more reliable, available, and serviceable:

- ▶ Fully redundant power, cooling and management, dynamic processor de-allocation and memory chip and lane sparing, and concurrent maintenance are the main reliability, availability, and serviceability (RAS) features.
- ▶ The system is mainly water-cooled, which gives a 100% heat capture. Some components are cooled by small fans, but the rear door heat exchanger captures this.
- ▶ Since most of the nodes are diskless nodes, the service nodes provide the operating system to them. The HFI network boots all diskless utility nodes.

The Power 775 Availability Plus (A+) feature allows processors, switching hubs, and HFI cables immediate failure-recovery because extra resources are available in the system. These extra resources fail in place and no hardware needs to be replaced until a specified threshold is reached.

The IBM Power 775 cluster solution provides High Performance Computing clients with:

- ▶ Sustained performance and low energy consumption for climate modeling and forecasting
- ▶ Massive scalability for cell and organism process analysis in life sciences
- ▶ Memory capacity for high resolution simulations in nuclear resource management
- ▶ Space and energy efficient for risk analytics and real time trading in financial services

Node

Each Power 775 planar consists of eight octants. Seven of the octants consist of 1x QCM, 1 x HUB, and 2 x PCI Express 16x. The other octant contains 1x QCM, 1x HUB, 2x PCI Express16x, and 1x PCI Express 8x:

- ▶ Compute power: 1 TF, 32 cores, 128 threads
- ▶ Memory: 512 GB max capacity, ~512 GBps peak memory BW (1/2 B/FLOP)
- ▶ I/O: 1 TB/s BW
- ▶ IBM Proprietary Fabric: On-board copper / Off-board Optical

One Power 775 Compute node physically represented by the drawer, also commonly referred to as CEC. A node consists of eight octants, including their local interconnect. Figure 3-2 shows the CEC drawer from the front.



Figure 3-2 CEC drawer front photo

Figure 3-3 on page 21 shows the CEC drawer rear view.

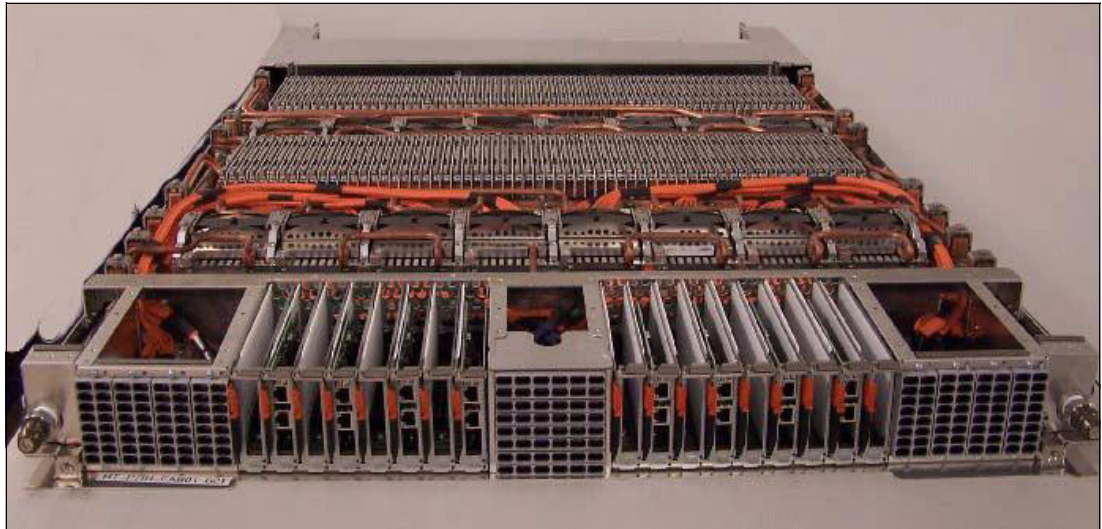


Figure 3-3 CEC drawer rear photo

IBM Flex System p260 and p460

This section provides an introduction to the IBM Flex System™ p260 and p460.

IBM Flex System overview

During the last 100 years, information technology moved from a specialized tool to a pervasive influence on nearly every aspect of life. To meet these business demands, IBM introduced a new category of systems. These systems combine the flexibility of general-purpose systems, the elasticity of cloud computing, and the simplicity of an appliance that is tuned to the workload.

IBM PureFlex System is a comprehensive infrastructure system that provides an expert integrated computing system. It combines servers, enterprise storage, networking, virtualization, and management into a single structure.

IBM PureFlex System is built from reliable IBM technology that supports open standards and offers confident road maps: IBM Flex System. IBM Flex System is designed for multiple generations of technology, supporting your workload today while being ready for the future demands of your business.

IBM Flex System Enterprise Chassis

The IBM Flex System Enterprise Chassis offers compute, networking, and storage capabilities far exceeding those currently available. The Enterprise Chassis is 10U height, support intermixing up to 14 compute nodes, and which based on POWER7 or Intel x86 chips, and the rear of chassis, accommodates four high speed networking switches. With interconnecting compute nodes, networking, and storage using a high performance and scalable mid-plane, Enterprise Chassis can support 40 Gb speeds.

The ability to support the workload demands of tomorrow's workloads is built in with a new IO architecture, which provides choice and flexibility in fabric and speed. With the ability to use Ethernet, InfiniBand, FC, FCoE, and iSCSI, the Enterprise Chassis is uniquely positioned to meet the growing I/O needs to the IT industry.

Figure 3-4 on page 22 shows the IBM Flex System Enterprise Chassis.



Figure 3-4 IBM Flex System Enterprise Chassis

Compute node

IBM Flex System offers compute nodes that vary in architecture, dimension, and capabilities. Optimized for efficiency, density, performance, reliability, and security, the portfolio includes the following IBM POWER7-based and Intel Xeon-based nodes:

- ▶ IBM Flex System x240 Compute Node, a two socket Intel Xeon-based compute node
- ▶ IBM Flex System x220 Compute Node, a cost-optimized two-socket Intel Xeon-based compute node
- ▶ IBM Flex System x440 Compute Node, a four-socket Intel Xeon-based server
- ▶ IBM Flex System p260 Compute Node, a two socket IBM POWER7-based compute node
- ▶ IBM Flex System p24L Compute Node, a two socket IBM POWER7-based compute node optimized for Linux
- ▶ IBM Flex System p460 Compute Node, a four socket IBM POWER7-based compute node

I/O modules

With a range of available adapters and switches to support key network protocols, you can configure IBM Flex System to fit in your infrastructure while still being ready for the future. The networking resources in IBM Flex System are standards-based, flexible, and fully integrated into the system, so you get no-compromise networking for your solution. Network resources are virtualized and managed by workload. These capabilities are automated and optimized to make your network more reliable and simpler to manage.

Here are the I/O Modules offered with the IBM Flex System:

- ▶ IBM Flex System Fabric EN4093 10 Gb Scalable Switch
- ▶ IBM Flex System EN2092 1 Gb Ethernet Scalable Switch
- ▶ IBM Flex System EN4091 10 Gb Ethernet Pass-thru
- ▶ IBM Flex System FC3171 8 Gb SAN Switch
- ▶ IBM Flex System FC3171 8 Gb SAN Pass-thru
- ▶ IBM Flex System FC5022 16 Gb SAN Scalable Switch
- ▶ IBM Flex System FC5022 24-port 16 Gb ESB SAN Scalable Switch
- ▶ IBM Flex System IB6131 InfiniBand Switch

IBM Flex System p260 and p460

The IBM Flex System p260 and p460 Compute Nodes are high-performance POWER7-based servers optimized for virtualization, performance, and efficiency.

IBM Flex System p260 Compute Node, 7895-22X, is a half-wide, Power Systems compute node with two POWER7 processor sockets, 16 memory slots, two I/O adapter slots, and an option for up to two internal drives for local storage.

IBM Flex System p460 Compute Node, 7895-42X, is a full-wide, Power Systems compute node with four POWER7 processor sockets, 32 memory slots, four I/O adapter slots, and an option for up to two internal drives for local storage.

The compute nodes offers numerous features to boost performance, improve scalability, and reduce costs:

- ▶ The IBM POWER7 processors, which improve productivity by offering superior system performance with Altivec floating point and integer SIMD instruction set acceleration.
- ▶ Integrated IBM PowerVM® technology, providing superior virtualization performance and flexibility.
- ▶ Choice of processors, including an 8-core processor operating at 3.5 GHz with 32 MB of L3 cache.
- ▶ Up to four processors, 32 cores, and 128 threads to maximize the concurrent execution of applications.
- ▶ Up to 16 (p260) or 32 (p460) DDR3 ECC memory RDIMMs that provide a memory capacity of up to 256 GB (p260) or 512 GB (p460).
- ▶ Support for IBM Active Memory™ Expansion, which allows the effective maximum memory capacity to be much larger than the true physical memory through innovative compression techniques.
- ▶ The use of solid-state drives (SSDs) instead of traditional spinning drives (HDDs), which can significantly improve I/O performance. An SSD can support up to 100 times more I/O operations per second (IOPS) than a typical HDD.
- ▶ Up to eight (p260) or 16 (p460) 10 Gb Ethernet ports per compute node to maximize networking resources in a virtualized environment.
- ▶ Includes two (p260) or four (p460) P7IOC high-performance I/O bus controllers to maximize throughput and bandwidth.
- ▶ Support for high-bandwidth I/O adapters, up to two in each p260 Compute Node or up to four in each p460 Compute Node. Support for 10 Gb Ethernet, 8 Gb Fibre Channel, and QDR InfiniBand.

Figure 3-5 on page 24 shows the IBM Flex System p260 compute node.



Figure 3-5 IBM Flex System p260 compute node

Figure 3-6 shows the IBM Flex System p460 compute node.



Figure 3-6 IBM Flex System p460 compute node

In HPC solutions, each compute node always needs high-speed communication adapters. For Flex System p260 and p460, the compute nodes do not have any networking integrated on the system. Table 3-1 lists the supported network adapters. All p260 and p460 configurations must include a 10 Gb (#1762) or 1 Gb (#1763) Ethernet adapter in slot 1 of the compute node.

Table 3-1 Supported network adapters

Feature code	Description	Maximum supported*
1762	IBM Flex System EN4054 4-port 10 Gb Ethernet Adapter	2 (p260), 4 (p460)
1763	IBM Flex System EN2024 4-port 1 Gb Ethernet Adapter	2 (p260), 4 (p460)
1761	IBM Flex System IB6132 2-port QDR InfiniBand Adapter	1 (p260), 3 (p460)

3.1.2 HPC scenario based on x86 platform

Before cluster-based computing, the typical supercomputer was a vector processor that typically cost over a million dollars because of the specialized hardware and software. With Linux and other freely available open source software components for clustering and improvements in commodity hardware, the situation now is quite different. You can build

powerful clusters with a small budget and keep adding extra nodes based on need, most of these type clusters based on x86 platform hardware.

IBM System x (x86) Servers help you stay ahead of your business and IT challenges by offering solutions with exceptional quality, value, performance, and ease of use.

We introduce IBM System x Servers for HPC solution in the next section.

IBM System x iDataPlex dx360 M4

IBM System x iDataPlex, a large-scale solution, can help you face constraints in power, cooling or physical space. The innovative design of the iDataPlex solution integrates intel processor-based processing into the node, rack, and data center for power and cooling efficiencies and required compute density:

- ▶ Flexible design for Internet-scale data centres
- ▶ Up to 40 percent power efficiency improvement
- ▶ Dramatically reduced cooling costs, air conditioning expense minimized or even eliminated
- ▶ Up to 5X compute density for efficient space utilization
- ▶ Front access cabling
- ▶ Easy to service and manage

A typical iDataPlex solution consists of multiple fully populated rack installations. The ground breaking iDataPlex solution offers increased density in a new rack design. It uses the dimensions of a standard 42U enterprise rack but can hold 100U of equipment, populated with up to 84 servers, plus sixteen 1U vertical slots for switches, appliances, and power distribution units (PDUs). This added density addresses the major problems that prevent most data centers today from reaching their full capacity: insufficient electrical power and excess heat.

The energy-efficient design of the iDataPlex servers and chassis can significantly reduce the incoming energy requirement compared with standard 1U servers. In addition, the optional liquid-cooled IBM Rear Door Heat eXchanger mounted to the back of the rack can remove 100% of the heat generated within the rack, drawing it from the data center before it exits the rack. It can even go beyond that to the point of helping to cool the data center itself and reducing the need for Computer Room Air Conditioning (CRAC) units. This allows racks to be positioned much closer together, eliminating the need for hot aisles between rows of fully populated racks.

With the iDataPlex chassis design, air needs to travel only 18 inches front to back, rather than the 30 plus inches of a typical enterprise server. This shallow depth is part of the reason that the cooling efficiency of iDataPlex servers is so high—shorter distance means better airflow. In addition, the new design uses four large 80 mm fans per 2U chassis for more efficiency and lower noise than the eight small 40 mm fans used in standard 1U servers. The increased air pressure resulting from the shorter distance through the rack and the larger fans makes for one of the most efficient air-cooled solutions on the market.

Unlike most conventional racks, which are often left largely empty due to power and cooling limitations, the iDataPlex Rack can be fully populated, while removing all rack heat from the data center (up to 100,000 BTUs or 30 kW), using the Rear Door Heat eXchanger. In addition, iDataPlex chassis uses highly-efficient (80 PLUS Platinum) power supplies, reducing energy draw and waste heat further.

IBM System x iDataPlex Rack has the following features:

- ▶ Shallow rack depth for efficient airflow and reduced cooling costs
- ▶ Horizontal space of 84U for installing servers (twice the density of a standard 42U rack)

- ▶ Vertical space of 16U for installing switches, power distribution units, and other optional devices
- ▶ Integrated cable-management hardware
- ▶ Front access to hard disk drives, system-board trays, I/O controls, cabling, and optional devices
- ▶ Front doors, side panels, and rear door
- ▶ Filler panels for unoccupied bays to help maintain proper cooling
- ▶ Hardware to bolt the rack cabinets together and form a suite
- ▶ Chassis and optional-device rails
- ▶ Casters and leveling feet
- ▶ Optional IBM Rear Door Heat eXchanger for the iDataPlex Rack

Figure 3-7 shows an IBM System x iDataPlex rack.



Figure 3-7 An IBM System x iDataPlex rack

The IBM System x iDataPlex dx360 M4 is designed to optimize density and performance within typical data center infrastructure limits. The unique half-depth form factor is designed to help you improve compute density in your space-constrained data center while also improving system cooling and energy efficiency.

Common features:

- ▶ High-performance, half-depth server that provides outstanding performance with outstanding power and cooling efficiencies

- ▶ Innovative dual-socket system with multiple compute, storage, and I/O configuration options to fit customer requirements
- ▶ Flexible design that is easy to deploy, integrate, service, and manage

Hardware summary:

- ▶ Up to two 2.7 GHz 8-core Intel Xeon E5-2600 Series processors (up to 168 per iDataPlex rack)
- ▶ Up to 512 GB of DDR3 memory per server
- ▶ Two dedicated x16 PCIe 3.0 slots per server
- ▶ Up to 6.0 TB HDD storage per 2U chassis
- ▶ Up to two servers per 2U 19" chassis mountable in a standard enterprise rack or iDataPlex rack

Figure 3-8 shows two dx360 M4 compute nodes installed in an 2U iDataPlex chassis.



Figure 3-8 Two dx360 M4 compute nodes installed in an 2U iDataPlex chassis

IBM Flex System x220

The IBM Flex System x220 Compute Node is the next generation cost-optimized compute node designed for less demanding workloads and high-density computing. The x220 is efficient and equipped with flexible configuration options and advanced management to run a broad range of workloads.

Figure 3-9 shows the IBM Flex System x220 Compute Node.



Figure 3-9 The IBM Flex System x220 Compute Node

The IBM Flex System x220 Compute Node is a high-availability, scalable compute node optimized to support the next-generation microprocessor technology. With a balance between

cost and system features, the x220 is an ideal platform for general business workloads and high-performance computing.

The IBM Flex System x220 Compute Node takes up a single half wide bay within the Flex System Enterprise Chassis. It is a dual socket, Xeon E5-2400 series based server.

Hardware summary:

- ▶ Intel Xeon processor E5-2400 product family. Up to two processors, 16 cores, and 32 threads maximize the concurrent execution of multi-threaded applications.
- ▶ Memory capacity up to 192 GB. There are 12 DIMM sockets supporting low profile (LP) RDIMMs and UDIMMs that support memory speeds of up to 1600 MHz to maximize memory performance.
- ▶ Automated power management with onboard sensors.
- ▶ Two 2.5" hot swap disk slots. Two 2.5-inch hot-swap SAS/SATA drive bays supporting SAS, SATA, and SSD drives. Optional eXFlash support for up to eight 1.8-inch SSDs. Onboard ServeRAID C105 supports SATA drives only.
- ▶ Integrated system management.
- ▶ SW RAID, HW RAID, or ServeRAID M5115 (RAID 0/1/5/6/10/50).
- ▶ Optional hypervisor.
- ▶ 2 x Mezzanine Cards (x8+x4) + x4 PCI Express 3.0.
- ▶ Dual Integrated 1 GbE.

3.1.3 IBM additional other software components for HPC solution

This section describes additional IBM software components for High Performance Computing solutions enablement.

The IBM XL C/C++ and Fortran Compiler

The IBM XL C/C++ compiler and IBM XL Fortran compiler offer advanced compiler and optimization technologies and are built on a common code base for easier porting of your applications between platforms. They comply with the latest C/C++ international standards and industry specifications and support a large array of common language features.

The IBM XL compiler is designed to optimize your infrastructure for Power Systems. From enhancements to support the latest standards, highly-tuned math libraries, to industry leading optimization technology, the IBM XL compiler allows you to get the most out of your hardware investment:

- ▶ Designed to take advantage of the latest Power Systems hardware for increased value and reduced costs
- ▶ Maximizes application performance through industry-leading optimization technology for better system utilization and cost reduction
- ▶ Improves developer productivity with access to highly-tuned libraries, utilities, options, and modern tools that simplify programming, shorten the development cycle, and reduce risk
- ▶ Eases application migration to Power Systems through conformance to the latest international programming standards protecting application investments

IBM MASS Library

IBM Mathematical Acceleration Subsystem (MASS) was originally launched by IBM in 1995, and has been continuously improved and expanded since then. There are currently versions

of MASS for all of the POWER processors that run AIX or LINUX operating systems. There are also versions for BlueGene. The libraries consists of libraries of mathematical functions tuned for optimum performance on a variety of IBM processors (Refer to Table 3-2).

- ▶ MASS consists of libraries of tuned mathematical functions that are available in versions for the AIX and Linux platforms.
- ▶ MASS libraries offer improved performance over the standard mathematical library routines, include both scalar and vector functions, are thread-safe, and support compilations in C, C++, and Fortran applications.
- ▶ The MASS libraries are shipped with the XL C, XL C/C++, and XL Fortran compiler products, and are also available for separate download.

Table 3-2 Features and benefits

Features	Details	Benefits
Scalar libraries	The MASS scalar libraries contain an accelerated set of frequently-used single- and double-precision math intrinsic functions.	Provide speedups up to 8x over the standard system library.
Vector libraries	The MASS vector libraries provide increased performance by operating on a vector of arguments. They include single-precision, double-precision, and integer functions.	Provide speedups up to over 35x compared to the standard system library.
SIMD libraries	The MASS Single Instruction Multiple Data (SIMD) libraries are provided for SIMD architectures. They operate on SIMD vector arguments, and include single-precision, double-precision, and integer functions.	Provide speedups up to 80% compared to the standard system library.
Compatible processors	POWER, POWER2, POWER3, POWER4, POWER5, IBM POWER6®, POWER7, IBM Blue Gene/L®, IBM Blue Gene/P®, Cell Broadband Engine (PPU and SPU).	
Supported operating systems	AIX, Linux	
32/64 bit mode support	32-bit mode applications are supported and 64-bit mode applications on processors supporting them.	

IBM ESSL and PESSL

The Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL are collections of state-of-the-art mathematical subroutines specifically designed to improve the performance of engineering and scientific applications on the IBM POWER processor-based servers and blades. ESSL and Parallel ESSL are commonly used in the aerospace, automotive, electronics, petroleum, utilities, and scientific research industries for applications, such as:

- ▶ Structural Analysis
- ▶ Time Series Analysis
- ▶ Computational Chemistry
- ▶ Computational Techniques
- ▶ Fluid Dynamics Analysis
- ▶ Mathematical Analysis
- ▶ Seismic Analysis
- ▶ Dynamic Systems Simulation
- ▶ Reservoir Modeling

- ▶ Nuclear Engineering
- ▶ Quantitative Analysis
- ▶ Electronic Circuit Design

ESSL and Parallel ESSL support 32-bit and 64-bit Fortran, C and C++ serial, SMP, and SPMD applications running under AIX and Linux.

ESSL

ESSL is a collection of high performance, mathematical subroutines providing a wide range of functions for many common scientific and engineering applications. The mathematical subroutines are divided into nine computational areas:

- ▶ Linear Algebra Subprograms
- ▶ Matrix Operations
- ▶ Linear Algebraic Equations
- ▶ Eigensystem Analysis
- ▶ Fourier Transforms, Convolutions, Correlations and Related Computations
- ▶ Sorting and Searching
- ▶ Interpolation
- ▶ Numerical Quadrature
- ▶ Random Number Generation

ESSL provides the following run-time libraries:

The ESSL Serial Library provides thread-safe versions of the ESSL subroutines for use on all processors. You can use this library to develop your own multi-threaded applications.

The ESSL Symmetric Multi-Processing (SMP) Library provides thread-safe versions of the ESSL subroutines for use on all SMP processors. In addition, some of these subroutines are also multi-threaded, meaning, they support the shared memory parallel processing programming model. You do not have to change your existing application programs that call ESSL to take advantage of the increased performance of using the SMP processors; instead, you can simply re-link your existing programs.

The ESSL IBM Blue Gene® Serial Library and the ESSL Blue Gene SMP Library run in a 32-bit integer, 32-bit pointer environment. Both libraries are tuned for IBM Blue Gene/P. A subset of the subroutines in the ESSL Blue Gene Serial Library use SIMD algorithms that utilize the IBM PowerPC® 450 dual FPUs. The ESSL Blue Gene SMP library provides thread-safe versions of the ESSL subroutines. A subset of these subroutines are also multi-threaded versions. Those multi-threaded versions support the shared memory parallel processing programming model. Some of these multi-threaded subroutines also use SIMD algorithms that utilize the PowerPC 450 dual FPUs.

The ESSL Serial Library and the ESSL SMP Library support the following application environments:

- ▶ 32-bit integers and 32-bit pointers
- ▶ 32-bit integers and 64-bit pointers
- ▶ 64-bit integers and 64-bit pointers

All libraries are designed to provide high levels of performance for numerically intensive computing jobs and both provide mathematically equivalent results. The ESSL subroutines can be called from application programs written in Fortran, C, and C++, and ESSL running on the AIX and Linux operating systems.

Parallel ESSL

Parallel ESSL is a scalable mathematical subroutine library for stand-alone clusters or clusters of servers connected through a switch and running AIX and Linux. Parallel ESSL supports the Single Program Multiple Data (SPMD) programming model using the Message Passing Interface (MPI) library. The Parallel ESSL SMP libraries support parallel processing applications on clusters of Power System servers and blades connected through a LAN supporting IP or with an InfiniBand switch.

Parallel ESSL provides subroutines in the following computational areas:

- ▶ Level 2 Parallel Basic Linear Algebra Subprograms (PBLAS)
- ▶ Level 3 PBLAS
- ▶ Linear Algebraic Equations
- ▶ Eigensystem Analysis and Singular Value Analysis
- ▶ Fourier Transforms
- ▶ Random Number Generation

For communication, Parallel ESSL includes the Basic Linear Algebra Communications Subprograms (BLACS), which use MPI. For computations, Parallel ESSL uses the ESSL subroutines (ESSL is a pre-requisite).

The Parallel ESSL subroutines can be called from 32-bit and 64-bit application programs written in Fortran, C, and C++ running the AIX and Linux operating systems.

The Parallel ESSL SMP Libraries are provided for use with the IBM Parallel Environment MPI library. You can run single or multi-threaded US or IP applications on all types of nodes. However, you cannot simultaneously call Parallel ESSL from multiple threads.

IBM General Parallel File System

IBM General Parallel File System (GPFS™) is a cluster file system. It provides concurrent access to a single file system or set of file systems from multiple nodes. These nodes can all be SAN attached or a mix of SAN and network attached. This enables high performance access to this common set of data to support a scale-out solution or provide a high-availability platform.

GPFS has many features beyond common data access including data replication, policy based storage management, and multi-site operation. You can create a GPFS cluster of AIX nodes, Linux nodes, Windows server nodes, or a mix of all three. GPFS can run on virtualized instances providing common data access in environments, leverage logical partitioning, or other hypervisors. Multiple GPFS clusters can share data within a location or across a wide area network (WAN) connection.

GPFS provides a global namespace, shared file system access among GPFS clusters, simultaneous file access from multiple nodes, high recoverability and data availability through replication, the ability to make changes while a file system is mounted, and simplified administration even in large environments.

Shared file system access among GPFS clusters

GPFS allows you to share data between separate clusters within location or across a WAN. Between clusters, users can share access to some or all file systems in either cluster.

Improved System performance

Using GPFS file systems can improve system performance by:

- ▶ Allowing multiple processes or applications on all nodes in the cluster simultaneous access to the same file using standard file system calls

- ▶ Increasing aggregate bandwidth of your file system by spreading reads and writes across multiple disks
- ▶ Balancing the load evenly across all disks to maximize their combined throughput, eliminating storage hotspots
- ▶ Supporting large file and file system sizes
- ▶ Allowing concurrent reads and writes from multiple nodes
- ▶ Allowing for distributed token (lock) management. Distributing token management reduces system delays associated with a lockable object waiting to obtain a token
- ▶ Allowing for the specification of multiple networks for GPFS daemon communication and for GPFS administration command usage within your cluster

File consistency

GPFS uses a sophisticated token management system to provide data consistency while allowing multiple independent paths to the same file by the same name from anywhere in the cluster.

Increased data availability

GPFS provides multiple features that improve the reliability of your file system. This includes automatic features, such as file system logging and configurable features, such as intelligently mounting file systems on startup to providing tools for flexible synchronous replication.

Enhanced system flexibility

With GPFS, your system resources are not frozen. You can add or delete disks while the file system is mounted. When the time is favorable and system demand is low, you can rebalance the file system across all currently configured disks. In addition, you can also add or delete nodes without having to stop and restart the GPFS daemon on all nodes.

Simplified storage management

GPFS provides storage management based on storage pools, policies, and filesets.

A storage pool is a collection of disks or RAID's with similar properties that are managed together as a group. Storage pools provide a method to partition storage within a file system.

Files are assigned to a storage pool based on defined policies. There are two types of policies:

- ▶ Placement policies:
 - Placing files in a specific storage pool when the files are created.
- ▶ File management policies:
 - Migrating files from one storage pool to another
 - Deleting files based on file characteristics
 - Changing the replication status of files
 - Snapshot metadata scans and file list creation

Filesets provide a method for partitioning a file system and allow administrative operations at a finer granularity than the entire file system.

Simplified administration

GPFS offers many of the standard file system interfaces, allowing most applications to execute without modification. Operating system utilities are also supported by GPFS. That is,

you can continue to use the commands you always used for ordinary file operations. The only unique commands are those for administering the GPFS file system.

GPFS administration commands are similar in name and function to UNIX and Linux file system commands with one important difference: The GPFS commands operate on multiple nodes. A single GPFS command can perform a file system function across the entire cluster.

LoadLeveler

LoadLeveler is a job management system that allows users to run more jobs in less time by matching the jobs' processing needs with the available resources. LoadLeveler schedules jobs and provides functions for building, submitting, and processing jobs quickly and efficiently in a dynamic environment.

When jobs are submitted, LoadLeveler will dispatch jobs based on their priority, resource requirements, and special instructions. Administrators can specify that long-running jobs run only on off-hours, that short-running jobs be scheduled around long-running jobs, or that certain users or groups get priority. In addition, the resources themselves can be tightly controlled: use of individual machines can be limited to specific times, users, or job classes.

LoadLeveler tracks the total resources used by each job and offers several reporting options to trace jobs and utilization by user, group, account, or type over a specified time period. To support chargeback for resource use, LoadLeveler can incorporate machine speed to adjust chargeback rates and be configured to require an account for each job.

LoadLeveler offers both a command line interface and a graphical interface in addition to an API that enables user-written applications to control it. LoadLeveler also supports high availability configurations to ensure reliable operation and automatically monitors the available compute resources to ensure that no jobs are scheduled to failed machines.

xCAT

Extreme Cloud Administration Toolkit (xCAT) is an open source, scalable, distributed computing management and provisioning tool that IBM developed. It is used for the deployment and administration of Linux or AIX-based clusters.

xCAT support:

- ▶ Provision Operating Systems on physical or virtual machines: SLES, RHEL, CentOS, Fedora, AIX, Windows, VMWare, KVM, PowerVM, zVM
- ▶ Scripted install, Stateless, Statelite, iSCSI or Cloning
- ▶ Remotely manage systems: Integrated Lights-out management, remote console, and distributed shell support
- ▶ Quickly set up and control management node services: DNS, HTTP, DHCP, TFTP

xCAT offers complete and ideal management for HPC clusters, RenderFarms, Grids, WebFarms, Online Gaming Infrastructure, Clouds, Data centers. It is agile, extendable, and based on years of system administration best practices and experience.

3.2 Developing parallel applications

To obtain all the advantages of a parallel environment, the applications that run in such an environment need to be developed using a paradigm different than the programming models used to develop applications to be run on a single computer because these applications are single or multi-threaded. Ideally, a parallel application needs to be flexible (run on different

platforms and environments) and scalable (have good performance running on environments that can have from a few to thousands of nodes).

This section provides an overview of the most commonly used parallel programming models, frameworks, and libraries that can help the development of applications for parallel environments using these models.

3.2.1 Programming models

Different mechanisms can be used by parallel processes to communicate with each other. This section describes the most common parallel programming models.

Shared memory (without threads)

In the shared memory model, a range of memory addresses can be accessed by the processes, which can be running on a single or multiple nodes. This region of memory can be used by these processes to communicate with each other by reading and writing data on this address space. When a shared memory space is used by several nodes, this memory space does not necessarily reside on a single centralized node. It can be physically spread on different nodes of the cluster.

Shared memory programming can be a relatively easy and intuitive model, but alternately the programmer needs to protect the critical sections from race conditions, by making use of concurrency mechanisms, such as locks, semaphores, and mutexes.

Threads

In the threads programming model, parallelism on a single node is obtained by creating multiple, concurrent execution paths within a single process. A thread can be described as a procedure or function that can be scheduled and run independently from its main process. Each thread can have its own private data and access a global, shared memory space. In this case, the programmer also needs to protect the critical sections, as mentioned in “Shared memory (without threads)” on page 34.

Threads can be used to improve the performance of a process, mainly on symmetric multiprocessing (SMP) and simultaneous multithreading (SMT) systems, generating less overhead on the operating system when compared to multiple process creation. This makes programming simpler and saves system resources, such as processing cycles and memory, that can be used for the real computation.

Message passing

In the message passing model, the processes have their own local memory that is used for the computation, and the communication between the processes is done by sending and receiving messages. These messages can be used to perform remote method invocation, send/receive data, or for synchronization between processes.

Hybrid

A common choice when developing applications for a parallel environment is to use more than one of the models previously described.

One of the most used hybrid models is a combination of the message passing model with the threads model. This model can take advantage of the communication between several nodes in a cluster, and simultaneously use the increasingly availability of several processing cores/threads in a node.

3.2.2 Most used frameworks and libraries

This section gives an overview of the most used frameworks and libraries that are available to help the development of parallel applications using the programming models described in the previous section.

Note: In addition to the parallel frameworks and libraries discussed in this chapter, IBM PE Developer Edition also provides tools to aid development using other parallel libraries, such as IBM LAPI, IBM PAMI, OpenACC, and OpenSHMEM. Parallel static analysis is discussed in “Parallel Static Analysis” on page 133.

POSIX threads

POSIX threads, or Pthreads, is a standard C language threads Application Programming Interface (API) used to create and manipulate threads. Implementations of this API are available for most POSIX UNIX-like operating systems and also for Microsoft Windows. The interface was specified by the IEEE POSIX 1003.1 standard (1995).

The Pthreads interface is defined as a set of C language types and functions, implemented using a `pthread.h` header file and a shared library, which sometimes can be provided as part of another library, such as the GNU C library (`glibc`).

Using Pthreads requires the programmer to explicitly program the parallelism of the threads and do all the threads management (controlling each thread creation, termination, synchronization, scheduling, and so on). This provides a fair amount of control to the programmer, but alternately it demands attention to details to make sure the application is thread-safe, meaning that the threads that are run simultaneously cannot have race conditions, possible dead-locks, or make the shared data to come to an inconsistent state. The programmer also needs to make sure that the external library's functions that are used by the threads are thread-safe too, or problems that are hard to debug can arise.

The maximum amount of threads a process can create and the amount of thread stack space available is implementation dependent.

Example 3-1 shows an example of a simple program using the Pthreads API.

Example 3-1 Pthreads example

```
1 #include <stdio.h>
2 #include <pthread.h>
3 #define NUMTHREADS 4
4
5 struct thread_args {
6     int thread_id;
7     int arg1;
8     int arg2;
9     int res;
10 };
11
12 void *sum(void *void_arg) {
13     struct thread_args *args = (struct thread_args *) void_arg;
14     printf("Thread %d will do the calculation\n", args->thread_id);
15     args->res = args->arg1 + args->arg2;
16     pthread_exit(NULL);
17 }
18
```

```

19 int main(int argc, char *argv[]) {
20     int i, rc;
21     int a1[NUMTHREADS], a2[NUMTHREADS], a1a2[NUMTHREADS];
22     struct thread_args args[NUMTHREADS];
23     pthread_t threads[NUMTHREADS];
24     void *status;
25
26     for (i = 0; i < NUMTHREADS; i++) {
27         a1[i] = i;
28         a2[i] = i;
29         args[i].thread_id = i;
30         args[i].arg1 = a1[i];
31         args[i].arg2 = a2[i];
32         printf("Main program will create thread %d\n", i);
33         pthread_create(&threads[i], NULL, sum, &args[i]);
34     }
35     for (i = 0; i < NUMTHREADS; i++) {
36         pthread_join(threads[i], &status);
37         a1a2[i] = args[i].res;
38         printf("a1a2[%d] = %d\n", i, a1a2[i]);
39     }
40
41     pthread_exit(NULL);
42 }

```

Note: The `pthread.h` header file needs to be included in the source code in order to use the POSIX threads API.

In Example 3-1 on page 35, the main program allocates three arrays of integer and initializes the first two (`a1` and `a2`). The main program creates one thread for each pair of elements of the `a1` and `a2` arrays and uses an array of `struct thread_args` elements to pass these values to the threads and the thread ID. The `pthread_create()` function passes only one argument to the function that will be threaded, so if the code needs to pass more than one parameter, the easiest way is to create a structure with all of the parameters, including one field for the return value if needed, then pass a pointer to this structure. Each of the threads will do the calculation and return the results on the same structure. The main program will then synchronize with the other threads, waiting for each of them to finish by calling `pthread_join()` on line 36, and store the result of the calculation in the `a1a2` array. In this example, the number of threads is fixed on the compilation time using a `define` directive (line 3), but it can also be determined at runtime.

This code can be compiled using the IBM XL C compiler, using the `x1c_r` invocation command:

```
$ x1c_r pthreads.c -o pthreads
```

Tip: All IBM XL invocations with a suffix of `_r` allow for thread-safe compilation and are used to create threaded applications or to link programs that use multi-threading.

Example 3-2 shows the output of the example program. Note that messages from the main program and the threads get interleaved, showing that they are running at the same time.

Example 3-2 Output from the Pthread program

```
$ ./pthreads
```

```

Main program will create thread 0
Main program will create thread 1
Thread 0 will do the calculation
Main program will create thread 2
Thread 1 will do the calculation
Main program will create thread 3
Thread 2 will do the calculation
Thread 3 will do the calculation
a1a2[0] = 0
a1a2[1] = 2
a1a2[2] = 4
a1a2[3] = 6

```

Tip: More information about POSIX Threads can be obtained at:

<https://computing.llnl.gov/tutorials/pthreads/>

OpenMP

Open Multi-Processing (OpenMP) is an Application Program Interface that provides a portable, scalable model for shared-memory parallel applications on platforms from desktops to supercomputers. The API is defined and managed by a group of major hardware and software vendors, including IBM.

OpenMP supports C/C++ and Fortran programming languages on most architectures and operating systems, including GNU/Linux, AIX, and Windows. Table 3-3 provides more information about the most used OpenMP compilers.

Table 3-3 OpenMP compilers

Vendor	Compiler	Supported operating systems (OS)	Supported languages	Notes
GNU	gcc	Linux, AIX, Solaris, Mac OS X and Windows	C, C++ and Fortran	Compile with <code>-fopenmp</code>
IBM	XL C/C++ / Fortran ^a	AIX and Linux	C, C++ and Fortran	Use the thread-safe compiler invocation commands (with <code>_r</code> suffix) with the <code>-qsmp</code> flag.
Intel	C/C++ / Fortran	Linux, Mac OS X and Windows	C, C++ and Fortran	Compile with <code>-Qopenmp</code> on Windows or <code>-openmp</code> on Linux or Mac OS X

a. For more information about IBM XL compilers, visit <http://www-01.ibm.com/software/awdtools/fortran> and <http://www-01.ibm.com/software/awdtools/xlcppp/>

Tip: More information about OpenMP compilers can be obtained at:

<http://openmp.org/wp/openmp-compilers/>

The goal of OpenMP is to provide an easy-to-use, portable, and simple standard programming model among a variety of shared memory architectures and platforms. This is done by providing a set of compiler directives that can be used parallel to a program, which is accomplished through using threads.

Unlike Pthreads, the programmer is not required to explicitly declare the functions that will be used by the threads or perform all of the thread management. Parallelization can be obtained by simply taking a serial program and inserting the appropriate compiler directives, or it can be as complex as inserting functions and locks.

The OpenMP API consists of three primary components:

- ▶ A set of compiler directives
- ▶ Runtime library routines
- ▶ Environment variables

Compiler directives appear in the source code as comments and are ignored by the compiler if those directives are not supported or if you tell the compiler to do so. In such cases, the program will be compiled and run as a serial application, with no parallelism. The OpenMP directives are used for several purposes:

- ▶ Spawning a parallel region
- ▶ Dividing blocks of code among threads
- ▶ Distributing loop interactions between threads
- ▶ Serializing sections of code
- ▶ Synchronization of work between threads

The compiler directives have the following format:

```
sentinel directive_name [clause, ...]
```

Example 3-3 and Example 3-4 show an example of directives in Fortran and C/C++.

Example 3-3 Example of Fortran directive

```
!$OMP PARALLEL DEFAULT(SHARED) PRIVATE(ALPHA,BETA)
```

Example 3-4 Example of C/C++ directive

```
#pragma omp parallel default(shared) private(alpha,beta)
```

The OpenMP Runtime Library Routines are used for several purposes:

- ▶ Setting and querying the number of threads
- ▶ Querying the thread ID
- ▶ Setting and querying the dynamic threads feature
- ▶ Querying if the code is in a parallel region, and at what level
- ▶ Setting and querying nested parallelism
- ▶ Setting, initializing and terminating locks
- ▶ Querying clock time and resolution

Note: Not all the OpenMP library routines are supported by all the implementations.

OpenMP provides several environment variables that can be used to control the execution of parallel application at run time. These variables can be used to control options, such as:

- ▶ Set the number of threads
- ▶ Specify how loop interactions are divided
- ▶ Bind threads to processors
- ▶ Enable/disable nested parallelism
- ▶ Set the maximum level of nested parallelism

- ▶ Enable/disable dynamic threads
- ▶ Set thread stack size

These environment variables are set in the same way that all the other environment variables are set, and it depends on the shell that is being used.

OpenMP does not provide any automatic mechanism to handle parallel I/O. If multiple threads can access the same file, it is the programmer's responsibility to ensure that the I/O is done correctly in a multi-threaded program.

It is possible to accomplish significant performance improvements by using only a small number of directives. Example 3-5 shows a simple example of how to parallelize the processing of an array using only one compiler directive.

Example 3-5 Example of array parallelization using OpenMP

```
1 #include <stdio.h>
2
3 int main(int argc, char *argv[]) {
4     const int n = 10;
5     int i, a[n];
6     int tid;
7
8     #pragma omp parallel for private(tid)
9     for (i = 0; i < n; i++) {
10        tid = omp_get_thread_num();
11        printf("Thread %d will calculate index #%d.\n", tid, i);
12        a[i] = 2 * i;
13    }
14
15    return 0;
16 }
```

The source code can be compiled with the IBM XL C compiler, using the `xlc_r` invocation command:

```
$ xlc_r -qsmp openmp.c -o with_openmp
```

One of the possible outputs is shown in Example 3-6, where we can see that two threads are created and the array processing is divided between them.

Example 3-6 Output of the OpenMP program

```
$ ./with_openmp
Thread 0 will calculate index #0.
Thread 0 will calculate index #1.
Thread 1 will calculate index #5.
Thread 1 will calculate index #6.
Thread 1 will calculate index #7.
Thread 1 will calculate index #8.
Thread 1 will calculate index #9.
Thread 0 will calculate index #2.
Thread 0 will calculate index #3.
Thread 0 will calculate index #4.
```

Note: IBM PE Developer Edition provides tools to aid the development of OpenMP applications. OpenMP profiling is discussed in “OpenMP profiling” on page 115 and “Use OpenMP profiling to identify if the workload for threads” on page 147. OpenMP static analysis is discussed in “Parallel Static Analysis” on page 133.

Unified Parallel C

Unified Parallel C (UPC) is an extension of the C programming language designed for high-performance computing on large-scale parallel systems. The extension provides a shared memory model in which the processes can access a single, partitioned global address space (PGAS) where variables can be directly read and written by any process, but each variable is physically associated with a single processor. This model can be used either with a common global address space (SMP or NUMA (Non-Uniform Memory Access)) or in a cluster with distributed memory. Since all the processes access a single address space, UPC provides a Single Program, Multiple Data (SPMD) programming model, in which the amount of parallelism is determined at the process startup time, typically with a single thread per processor/core.

To implement parallelism, UPC extends the ISO C 99 standard with the following constructs:

- ▶ An explicitly parallel execution model
- ▶ A shared address space
- ▶ Synchronization primitives and a memory consistency model
- ▶ Memory management primitives.

Table 3-4 shows the list of mostly used UPC compilers, vendors, and supported operating systems.

Table 3-4 UPC compilers

Vendor	Compiler	Supported operating systems (OS)	Supported platforms
GNU	gupc	Linux and Mac OS X	x86, x86_64 and ia64
Berkeley	Berkeley UPC compiler	Linux, AIX, Mac OS X and various UNIX-like systems	x86, x86_64, ia64 PowerPC, IBM BlueGene/P, IBM BlueGene/Q, IBM BlueGene/L (experimental), IBM Power 775, among others.
HP	HP UPC	Linux	x86_64 and ia64

In contrast to OpenMP, the parallelization of code and variables needs to be explicitly programmed. Example 3-7 shows a simple UPC program where three arrays are declared as shared (line 4), meaning that they can be accessed by any thread running on any node. The first thread will initialize two of the arrays (lines 9-15), and all the threads will be synchronized at line 17 using the `upc_barrier` statement. The `upc_forall` statement at line 19 is a simple way to define work sharing among the threads. The difference between a normal C for loop and the UPC `upc_forall` loop is the fourth field, called the affinity field. This field determines which thread will execute the current iteration of the loop body inside the `upc_forall` block. In this example, we have as many elements on the array as the number of threads, and each of the threads will execute one iteration of the loop. The threads will be synchronized again at line 24, to make sure all the elements of the `a1a2` array are computed. Lastly, the first thread will print the contents of the combined array.

Example 3-7 Example of array parallelization using UPC

```
1 #include <stdio.h>
```

```

2 #include <upc_relaxed.h>
3
4 shared int a1[THREADS], a2[THREADS], a1a2[THREADS];
5
6 int main() {
7     int i;
8
9     if (MYTHREAD == 0) {
10        printf("Thread %d will initialize the arrays.\n", MYTHREAD);
11        for (i = 0; i < THREADS; i++) {
12            a1[i] = i;
13            a2[i] = i;
14        }
15    }
16
17    upc_barrier;
18
19    upc_forall(i = 0; i < THREADS; i++; i) {
20        printf("Thread %d will computer array index %d.\n", MYTHREAD, i);
21        a1a2[i] = a1[i] + a2[i];
22    }
23
24    upc_barrier;
25
26    if (MYTHREAD == 0) {
27        printf("Thread %d will print the combined array.\n", MYTHREAD);
28        for (i = 0; i < THREADS; i++)
29            printf("a1a2[%d] = %d\n", i, a1a2[i]);
30    }
31
32    return 0;
33 }

```

Note: To use the UPC API, at least one of the UPC header files needs to be included on the source file. In Example 3-7, the `upc_relaxed.h` header is being included to indicate that a relaxed memory consistency model is to be used.

For this example, the Berkeley UPC compiler is used to compile the source code. The following command compiles the code, indicating that the UDP protocol is used for communication between the processes and that two threads are created per node:

```
$ upcc -network=udp -pthreads=2 example.upc -o example
```

When using UDP for communication, a list of the cluster nodes to be used for the job needs to be provided. The list can be provided using a text file containing a node name on each line and exporting an environment variable called `UPC_NODEFILE` that contains the file name:

```
$ export UPC_NODEFILE=<path to nodes file>
```

Example 3-8 on page 42 shows the output of the example program. The `-n 4` parameter indicates that a total of four threads will be run. For this example, a file containing two nodes (`node01` and `node02`) was provided as the `UPC_NODEFILE`. The UPCR (UPC Runtime) engine shows that two threads will be run on each of the nodes. Following, the messages printed by the threads are displayed, showing that each of the threads ran one iteration of the loop and the first thread was able to directly access the region of memory written by the other threads, which could have been run on other nodes.

Example 3-8 Output of the UPC program

```
$ upcrun -n 4 ./example
UPCR: UPC threads 0..1 of 4 on node01 (pshm node 0 of 2, process 0 of 2,
pid=10360)
UPCR: UPC threads 2..3 of 4 on node02 (pshm node 1 of 2, process 1 of 2,
pid=49089)
Thread 0 will initialize the arrays.
Thread 0 will computer array index 0.
Thread 1 will computer array index 1.
Thread 3 will computer array index 3.
Thread 2 will computer array index 2.
Thread 0 will print the combined array.
a1a2[0] = 0
a1a2[1] = 2
a1a2[2] = 4
a1a2[3] = 6
```

Tip: More information about the UPC language can be obtained at:

<https://upc.gwu.edu/>

Note: IBM PE Developer Edition provides tools to aid the development of UPC applications. UPC static analysis is discussed in “Parallel Static Analysis” on page 133.

MPI

The Message Passing Interface (MPI) is a standard that specifies a portable, scalable, efficient, flexible and language-independent interface for writing message-passing programs. MPI addresses primarily the message-passing programming model, in which data is moved from the address space of one process to another process through cooperative operations on each process. MPI is a specification, not an implementation or a language. This specification is for a library interface and all MPI operations are expressed as functions, subroutines, or methods, according to the appropriate language bindings, which for C/C++ and Fortran are part of the MPI standard. Bindings are available for other languages as well, including Perl, Python, R, Ruby, Java, and CL.

The designers of MPI sought to make use of the most attractive feature of several existing message-passing systems, rather than selecting one of them to be adopted as a standard. MPI has been strongly influenced by the work of several organizations, including the IBM Thomas J. Watson Research Center. MPI has become widely used for communication among processes that follow parallel programming models running on distributed memory systems.

The design goal of MPI is quoted from MPI: A Message-Passing Interface Standard (Version 1.1), as follows:

- ▶ Design an application programming interface (not necessarily for compilers or a system implementation library).
- ▶ Allow efficient communication: Avoid memory-to-memory copying and allow overlap of computation and communication and offload to communication co-processor, where available.
- ▶ Allow for implementations that can be used in a heterogeneous environment.
- ▶ Allow convenient C and Fortran 77 bindings for the interface.

- ▶ Assume a reliable communication interface: the user need not cope with communication failures. Such failures are dealt with by the underlying communication subsystem.
- ▶ Define an interface that is not too different from current practice, such as PVM, NX, Express, p4, and so on, and provide extensions that allow greater flexibility.
- ▶ Define an interface that can be implemented on many vendor's platforms, with no significant changes in the underlying communication and system software.
- ▶ Semantics of the interface must be language independent.
- ▶ The interface must be designed to allow for thread-safety.

The MPI standard includes:

- ▶ Point-to-point communication
- ▶ Collective operations
- ▶ Process groups
- ▶ Communication contexts
- ▶ Process topologies
- ▶ Bindings for Fortran 77 and C
- ▶ Environmental management and inquiry
- ▶ Profiling interface

The MPI standard has several implementations available. Table 3-5 shows a list of the most common implementations and the supported operating systems.

Table 3-5 MPI implementations

Vendor	Name	Supported operating systems
IBM	IBM MPI ^a	AIX and Linux
Several (including IBM)	Open MPI	Linux, Solaris and Windows
Several (including IBM)	MPICH	Most flavors of UNIX, Linux, Mac OS X and Windows.
HP	HP-MPI	Linux and Windows
Intel	Intel MPI	Linux and Windows

a. Included in the IBM Parallel Environment (PE)

The standard has two different major versions, MPI-1 and MPI-2. MPI-1 emphasizes message passing and has a static runtime environment. MPI-2 is basically a superset of MPI-1, although some functions are deprecated. It includes new features, such as support for parallel I/O, dynamic process management, and remote memory management. MPI-1 has no shared memory concept, and MPI-2 has a limited distributed shared memory concept.

Example 3-9 shows the source code of a simple example program that uses the most common MPI functions.

Example 3-9 MPI example

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <mpi.h>
4
5 int main(int argc, char *argv[]) {
6     int my_rank, numprocs;
7     int *a1, *a2, *ala2;

```

```

 8   int temp[2];
 9   char hostname[256];
10   int i;
11   MPI_Status stat;
12
13   MPI_Init(&argc, &argv);
14   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
15   MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
16
17   gethostname(hostname, sizeof(hostname));
18   printf("Process rank #%d is running on %s\n", my_rank, hostname);
19
20   if (my_rank == 0) {
21       a1 = (int *) malloc(numprocs * sizeof(int));
22       a2 = (int *) malloc(numprocs * sizeof(int));
23       a1a2 = (int *) malloc(numprocs * sizeof(int));
24       for (i = 0; i < numprocs; i++) {
25           a1[i] = i;
26           a2[i] = i;
27       }
28
29       a1a2[0] = a1[0] + a2[0];
30
31       for (i = 1; i < numprocs; i++) {
32           temp[0] = a1[i];
33           temp[1] = a2[i];
34           MPI_Send(temp, 2, MPI_INT, i, 0, MPI_COMM_WORLD);
35           MPI_Recv(temp, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &stat);
36           a1a2[i] = temp[0];
37       }
38
39       for (i = 0; i < numprocs; i++)
40           printf("a1a2[%d] = %d\n", i, a1a2[i]);
41
42   } else {
43       MPI_Recv(temp, 2, MPI_INT, 0, 0, MPI_COMM_WORLD, &stat);
44       temp[0] = temp[0] + temp[1];
45       MPI_Send(temp, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
46   }
47
48   MPI_Finalize();
49   return 0;
50 }

```

This program allocates three arrays, in which the third one will contain the sum of the elements of the other two. The `MPI_Init` function (called on line 13) will create MPI processes. `MPI_Comm_size` (line 14) is used to discover the number of created processes of this program, and `MPI_Comm_rank` (line 15) is used to discover the identification of the current MPI process, or *rank*. The first MPI process (rank 0) will initialize the `a1` and `a2` arrays, calculate the sum for the index 0, and send a message to all the other ranks with the contents of an element of each of the first two arrays. Each of the other ranks will receive the message, calculate the sum of the two elements, and send the result back to rank 0, which will store the value on the third array. `MPI_Send` and `MPI_Recv` are the two main MPI functions to perform blocking communication.

The code can be compiled using different compilers and libraries. For this example, we use the `mpicc` front-end compiler included in the IBM Parallel Environment (PE), using the IBM MPI library:

```
$ mpicc -I/opt/ibmhpc/pe1207/pempi/include \  
> -L/opt/ibmhpc/pe1207/pempi/gnu/lib/libmpi64/ -lmpi_ibm vectors.c -o vectors
```

For this example, POE is used to launch four instances of the MPI program, which run on two nodes:

```
$ poe ./vectors -hfile ~/hosts_file -procs 4
```

Example 3-10 displays the output of the MPI program showing that four processes ranging from rank 0 to rank 3 run on two different nodes.

Example 3-10 MPI example output

```
Process rank #0 is running on node01  
Process rank #2 is running on node01  
Process rank #1 is running on node02  
Process rank #3 is running on node02  
a1a2[0] = 0  
a1a2[1] = 2  
a1a2[2] = 4  
a1a2[3] = 6
```

Note: IBM PE Developer Edition provides tools to aid the development of MPI applications. MPI profiling and tracing is discussed in “MPI profiling and trace” on page 111, “Using binary instrumentation for MPI profiling” on page 145, and “MPI profiling” on page 173, and MPI static analysis is discussed in “Parallel Static Analysis” on page 133.



Server installation

This chapter provides instructions to install and configure IBM Parallel Environment Developers Edition (PEDE) server component on supported operating systems. We also mention tuning tips and customizations for High Performance Computing (HPC) clusters.

The following details are covered in this chapter:

- ▶ Software requirements
- ▶ PEDE packaging considerations
- ▶ Addition software for integration with PEDE:
 - Job schedulers (IBM TWS LoadLeveler)
 - Distributed file systems (IBM GPFS)
 - Environment control (environment modules)
 - Software revision control tools (GIT or CVS)
- ▶ PEDE Install instructions (AIX and Linux)
- ▶ Post-Installation tuning:
 - Quick Parallel Environment Runtime tuning
 - GPFS tunable parameters affecting HPC performance
 - HPC Cluster verification
 - Environment customization (environment modules, shell)

4.1 Software requirements

This section describes the software requirements for IBM PE Developer Edition server component in the three supported operating systems. “Supported operating systems (software)” on page 3 shows the supported operating systems that are available for IBM PE Developer Edition. Table 4-1 shows the software packages that are required to install IBM PE Developer Edition server component.

Table 4-1 Software requirements for PEDE server component

AIX 7.1	RHEL 6.2 (on Power)	RHEL 6.2 (on x86_64)	SLES 11 SP2 (on x86_64)
	compat-libstdc++ (ppc and ppc64)	compat-libstdc++ (32 bit and 64 bit)	libstdc++-devel libstdc++43-devel
	libgcc (ppc and ppc64)	libgcc (32 bit and 64 bit)	libgcc (32 and 64 bit)
	libstdc++ (ppc and ppc64)	libstdc++ (32 bit and 64 bit)	libstdc++33-32 bit libstdc++33 (64 bit)
	libstdc++-devel (ppc and ppc64)	libstdc++-devel (32 bit and 64 bit)	libstdc++43-32 bit libstdc++43 (64 bit)
	libXp	libXp	
	openmotif	openmotif	
IBM XLC/C++ Compilers (12.1 or later) IBM XLF Compilers (14.1 or later)	IBM XLC/C++ Compilers (12.1 or later) IBM XLF Compilers (14.1 or later)	(compiler option 1) GNU Compilers	(compiler option 1) GNU Compilers
		(compiler option 2) Intel Compilers (11.1 or later)	(compiler option 2) Intel Compilers (11.1 or later)
RSCT and SRC	RSCT and SRC	SRC	SRC
Parallel Environment Runtime	Parallel Environment Runtime	Parallel Environment Runtime	Parallel Environment Runtime

Install targets: The software requirements in Table 4-1 are focused in compilation nodes. Compute nodes might not require the same full software packages installed, only the runtime packages, depending on what purpose they carry.

4.2 PEDE packaging considerations

This section details IBM PE Developer Edition server component packaging and presents additional software possible to integrate with supported environments.

4.2.1 Package contents

The IBM PE Developer Edition server component is distributed as a single package that is available using DVD media with the following contents:

- ▶ IBM International Program License Agreement in multi-language booklet (LC23-5123), and its License Information (L-RHAN-8KEP76) in multiple languages (ppe.loc.license)
- ▶ Product *readme* file that describes the program's specified operating environment and program specifications
- ▶ Documentation for the HPC Toolkit (hpct_guide.pdf)
- ▶ Installation document for Eclipse PTP (ptp_inst_guide.pdf)
- ▶ Program packages:
 - PTP Eclipse (ppedev.ptp, ppedev.ptp.rte)
 - HPC Toolkit (ppedev.hpct, ppedev.rte)

The previous contents are valid for all supported distributions, as detailed in Table 4-2.

Table 4-2 IBM PEDE server program packages for respective supported operating systems

Operating system	Name	Description
AIX 7.1	ppedev.loc.license	IBM PEDE License
	ppedev.ptp.rte	PTP Runtime
	ppedev.ptp	PTP Framework
	ppedev.rte	IBM HPC Toolkit Runtime
	ppedev.hpct	IBM HPC Toolkit
RHEL 6.2 (on Power)	ppedev_license-1.2.0-0.ppc64.rpm	IBM PEDE License
	ppedev_ptp_rte_rh6p-1.2.0-0.ppc64.rpm	PTP Runtime
	ppedev_ptp_rh6p-1.2.0-0.ppc64.rpm	PTP Framework
	ppedev_runtime_rh6p-1.2.0-0.ppc64.rpm	IBM HPC Toolkit Runtime
	ppedev_hpct_rh6p-1.2.0-0.ppc64.rpm	IBM HPC Toolkit
RHEL 6.2 (on x86_64 systems)	ppedev_license-1.2.0-0.x86_64.rpm	IBM PEDE License
	ppedev_ptp_rte_rh6x-1.2.0-0.x86_64.rpm	PTP Runtime
	ppedev_ptp_rh6x-1.2.0-0.x86_64.rpm	PTP Framework
	ppedev_runtime_rh6x-1.2.0-0.x86_64.rpm	IBM HPC Toolkit Runtime
	ppedev_hpct_rh6x-1.2.0-0.x86_64.rpm	IBM HPC Toolkit
SLES 11 SP2 (on x86_64 systems)	ppedev_license-1.2.0-0.x86_64.rpm	IBM PEDE License
	ppedev_ptp_rte_sles11x-1.2.0-0.x86_64.rpm	PTP Runtime
	ppedev_ptp_sles11x-1.2.0-0.x86_64.rpm	PTP Framework
	ppedev_runtime_sles11x-1.2.0-0.x86_64.rpm	IBM HPC Toolkit Runtime
	ppedev_hpct_sles11x-1.2.0-0.x86_64.rpm	IBM HPC Toolkit

IBM PEDE clients: They are in the `/opt/ibmhpc/ppdev.ptp/eclipse` directory. Further details for the supported operating systems and the available program packages are in “Supported operating systems (software)” on page 3.

4.2.2 Additional software

In this section, we list software tools that enrich the IBM PE Developer Edition experience. This software is not included in the IBM PE Developer Edition server package and can be either an IBM product or Open Source software. The Open Source software can be obtained by compiling the application code or in binary format along with the operating system that distributes it. Table 4-3 details the names of these tools and their corresponding program packages for the following areas:

- ▶ Job schedulers
- ▶ Distributed file systems
- ▶ Environment control tools
- ▶ Software revision control tools

Package versions: The software package versions presented in Table 4-3 are examples of supported versions. For complete support details, consult corresponding online product support.

Table 4-3 Program package names for respective operating systems

Operating system	Tool	Name
AIX 7.1	IBM TWS LoadLeveler	LoadL.resmgr.full LoadL.resmgr.loc.license LoadL.resmgr.msg.en_US LoadL.scheduler.full LoadL.scheduler.loc.license LoadL.scheduler.msg.en_US LoadL.scheduler.webui
	IBM GPFS	gpfs.base gpfs.msg.en_US gpfs.docs.data
	Environment Modules	(need compilation from source)
	Git	git-4.3.20-4
	CVS	cvs-1.11.17-3
RHEL 6.2 (on Power)	IBM TWS LoadLeveler	LoadL-scheduler-full-RH6-PPC64-5.1.0.10-0.ppc64 LoadL-utils-RH6-PPC64-5.1.0.10-0.ppc64 LoadL-resmgr-full-RH6-PPC64-5.1.0.10-0.ppc64 LoadL-full-license-RH6-PPC64-5.1.0.0-0.ppc64
	IBM GPFS	gpfs.base-3.5.0-3.ppc64 gpfs.gpl-3.5.0-3.noarch gpfs.docs-3.5.0-3.noarch gpfs.msg.en_US-3.5.0-3.noarch
	Environment Modules	environment-modules-3.2.7b-6.el6.ppc64
	Git	git-1.7.1-2.el6_0.1.ppc64
	CVS	cvs-1.11.23-11.el6_0.1.ppc64

Operating system	Tool	Name
SLES 11 SP2 (64 bit)	IBM TWS LoadLeveler	LoadL-full-license-SLES11-X86_64-5.1.0.4-0 LoadL-scheduler-full-SLES11-X86_64-5.1.0.11-0 LoadL-resmgr-full-SLES11-X86_64-5.1.0.11-0
	IBM GPFS	gpfs.base-3.4.0-11 gpfs.gplbin-2.6.32.12-0.7-default-3.4.0-11 gpfs.msg.en_US-3.4.0-11
	Environment Modules	(need compilation from source)
	Git	git-core-1.6.0.2-7.26
	CVS	cvs-1.12.12-144.21

Eclipse (synchronized projects): A software control system is required to use these projects. Git or CVS is available from the Linux distributions respective repositories and from the AIX Toolbox for the AIX operating systems:

<http://www-03.ibm.com/systems/power/software/aix/linux/toolbox/alpha.html>

Job schedulers

Job schedulers improve the use of cluster resources and enables queued job execution.

IBM TWS LoadLeveler

IBM Tivoli Workload Scheduler LoadLeveler enables HPC clusters to integrate job scheduling with Parallel Operating Environment (POE) runtime. It is a licensed product and can be obtained from IBM to AIX, RHEL, and SUSE operating systems.

Distributed file systems

Distributed file systems enable distributed jobs to run and debug within a shared file system. They are also used to simplify user data management and increase cluster IO performance.

IBM GPFS

The IBM General Parallel File System is a high performance and scalable distributed file system. It is a licensed product and can be obtained from IBM for AIX, RHEL, and SUSE operating systems.

Environment control tools

Environment control tools enable customized environments to be selected on demand when building Eclipse projects.

Environment modules

The Open Source software that is useful to create different compilation environments are:

- ▶ Availability: UNIX and Linux
- ▶ License: GNU GPL v2
- ▶ Download options:
 - <http://sourceforge.net/projects/modules/files/> (requires compilation)
 - Operating system distributed (compiled)

Software revision control tools

Software revision control tools add control for synchronized and remote Eclipse projects.

Git

- ▶ Availability: POSIX compatible operating systems (UNIX, Linux and Windows)
- ▶ License: GNU GPL v2
- ▶ Download options:
 - <http://git-scm.com/>
 - Operating system distributed

Concurrent Versions System

- ▶ Availability: UNIX, Linux and Windows
- ▶ License: GNU GPL v2
- ▶ Download options:
 - <http://savannah.nongnu.org/projects/cvs/>
 - Operating system distributed

4.3 Installation

This section describes IBM PE Developer Edition server installation for supported operating systems. We list installation instructions for the following operating systems, for the *Login/Front End* node:

- ▶ AIX 7.1
- ▶ RHEL 6 (on Power)
- ▶ SLES 11 SP2 or RHEL 6.2 (x86_64)

When using a cluster for HPC purposes, the packages do not need to be installed on every single node. Table 4-4 describes where you need to install each package, depending on the node type.

Table 4-4 IBM PE Developer Edition server installation layout

Components	Compute nodes (w/disk)	Compute nodes (diskless)	Login/Front End nodes
HPC Toolkit Runtime	Install	Install	Install
HPC Toolkit (~75MB)	No need to install	No need to install	Install
PTP Framework Runtime	Install (if using PTP debugger)	Install (if using PTP debugger)	Install
PTP Framework (~1.5 GB)	No need to install	No need to install	Install (here or somewhere else)

PTP Framework: This package only needs to be installed once and in only one server. This package contains all of the supported PTP client packages.

4.3.1 AIX 7.1

After all requirements are met from Table 4-1 on page 48, follow the next steps to install PEDE over AIX 7.1:

1. Copy all files to a directory named `<images_directory>`
2. Install the IBM HPC Toolkit runtime: `installp -a -X -Y -d <images_directory> ppedev.rte`
3. Install the PTP runtime: `installp -a -X -Y -d <images_directory> ppedev.ptp.rte`
4. Install the PTP framework: `installp -a -X -Y -d <images_directory> ppedev.ptp`
5. Install the IBM HPC Toolkit: `installp -a -X -Y -d <images_directory> ppedev.hpct`

4.3.2 RHEL 6 (on IBM POWER)

After all requirements are met from Table 4-1 on page 48, follow the next steps to install PEDE on RHEL 6:

1. Install the license: `rpm -hiv ppedev_license-1.2.0-0.ppc64.rpm`

License installation: After installing the license package, the `rpm` command displays informative text, as shown in Example 4-1.

Example 4-1 License acceptance procedures

IBM PE Developer Edition License RPM is installed. To accept the LICENSE please run:

```
/opt/ibmhpc/ppedev.hpct/lap/accept_ppedev_license.sh
```

Before calling `accept_ppedev_license.sh`, you must set the `IBM_PPEDEV_LICENSE_ACCEPT`

environment variable to one of the following values:

```
yes = Automatic license acceptance.  
no  = Manual license acceptance.
```

-
2. Export the license agreement: `export IBM_PPEDEV_LICENSE_ACCEPT=yes`
 3. Accept the license: `/opt/ibmhpc/ppedev.hpct/lap/accept_ppedev_license.sh`
 4. Install the PTP runtime: `rpm -hiv ppedev_ptp_rte_rh6p-1.2.0-0.ppc64.rpm`
 5. Install the PTP framework: `rpm -hiv ppedev_ptp_rh6p-1.2.0-0.ppc64.rpm`
 6. Install the IBM HPC Toolkit runtime: `rpm -hiv ppedev_runtime_rh6p-1.2.0-0.ppc64.rpm`
 7. Install the IBM HPC Toolkit: `rpm -hiv ppedev_hpct_rh6p-1.2.0-0.ppc64.rpm`

HPC Toolkit: Step 7 fails with dependencies requirements if `libXp` and `openmotif` rpms are not installed, as detailed in Table 4-1 on page 48.

4.3.3 SLES 11 SP2 or RHEL 6.2 (x86_64)

After all requirements are met from Table 4-1 on page 48, follow the next steps to install PEDE over SLES 11 SP3 or RHEL 6.2:

1. Install the license: `rpm -hiv ppedev_license-1.2.0-0.x86_64.rpm`

License installation: After installing the license package, the `rpm` command displays the text shown in Example 4-2.

Example 4-2 License acceptance procedures for SLES 11 SP2

IBM PE Developer Edition License RPM is installed. To accept the LICENSE please run:

```
/opt/ibmhpc/ppedev.hpct/lap/accept_ppedev_license.sh
```

Before calling `accept_ppedev_license.sh`, you must set the `IBM_PPEDEV_LICENSE_ACCEPT`

environment variable to one of the following values:

`yes` = Automatic license acceptance.

`no` = Manual license acceptance.

2. Export the license agreement: `export IBM_PPEDEV_LICENSE_ACCEPT=yes`

3. Accept the license: `/opt/ibmhpc/ppedev.hpct/lap/accept_ppedev_license.sh`

□ For SLES 11 SP2:

a. Install the PTP runtime: `rpm -hiv ppedev_ptp_rte_sles11x-1.2.0-0.x86_64.rpm`

b. Install the PTP framework: `rpm -hiv ppedev_ptp_sles11x-1.2.0-0.x86_64.rpm`

c. Install the IBM HPC Toolkit runtime:

```
rpm -hivppedev_runtime_sles11x-1.2.0-0.x86_64.rpm
```

d. Install the IBM HPC Toolkit: `rpm -hiv ppedev_hpct_sles11x-1.2.0-0.x86_64.rpm`

□ For RHEL 6.2:

a. Install the PTP runtime: `rpm -hiv ppedev_ptp_rte_rh6x-1.2.0-0.x86_64.rpm`

b. Install the PTP framework: `rpm -hiv ppedev_ptp_rh6x-1.2.0-0.x86_64.rpm`

c. Install the IBM HPC Toolkit runtime:

```
rpm -hiv ppedev_runtime_rh6x-1.2.0-0.x86_64.rpm
```

d. Install the IBM HPC Toolkit: `rpm -hiv ppedev_hpct_rh6x-1.2.0-0.x86_64.rpm`

4.4 Post-installation set up

This section describes which actions to take after installing IBM PE Developer Edition. All of the following recommendations are based on user experience, and are therefore subject to change at any time and dependent on the code-developing scenarios. The post-installation described in this section is related to cluster products tuning, system environment configurations, and components customization that work along side with IBM PE Developer Edition:

- ▶ Quick Parallel Environment Runtime tuning
- ▶ GPFS tunable parameters affecting HPC performance
- ▶ HPC Cluster verifications
- ▶ Customizing the environment

4.4.1 Quick Parallel Environment Runtime tuning

Because the Parallel Environment Runtime can be installed in different cluster architectures and sizes, a set of tuning parameters must be verified and, if required, changed for each

particular case. These actions tend to be time consuming and might also need investigation for advanced tuning. The Parallel Environment Operating (POE) environment delivers a script tool that can quickly evaluate which detected parameters must be changed, as shown in Example 4-3 and Example 4-4. The script path that is valid for AIX, RHEL, and SUSE operating systems respectively are:

- ▶ `/opt/ibmhpc/pecurrent/ppe.poe/bin/pe_node_diag` (AIX)
- ▶ `/opt/ibmhpc/pecurrent/base/bin/pe_node_diag` (RHEL and SUSE)

Example 4-3 Output from /opt/ibmhpc/pecurrent/base/bin/pe_node_diag (RHEL 6.2 on Power)

```
# /opt/ibmhpc/pecurrent/base/bin/pe_node_diag
/proc/sys/net/ipv4/ipfrag_low_thresh has 196608 but 1048576 is recommended.
/proc/sys/net/ipv4/ipfrag_high_thresh has 262144 but 8388608 is recommended.
limit for nofiles is [1024],recommended is [4096]
limit for locked address space is [64],recommended is [unlimited]
```

For Example 4-3, the `/etc/security/limits.conf` and `/etc/sysctl.conf` files must be changed to accommodate the recommended parameters values.

Example 4-4 Output from /opt/ibmhpc/pecurrent/ppe.poe/bin/pe_node_diag (AIX)

```
# /opt/ibmhpc/pecurrent/ppe.poe/bin/pe_node_diag
sb_max has 1114112 but 8388608 is recommended.
limit for data is [131072],recommended is [unlimited]
limit for nofiles is [2000],recommended is [4096]
maxuproc has 256 but 1024 is recommended.
```

In Example 4-4, change the `/etc/security/limits` file and execute the following commands to accommodate the recommended values:

- ▶ `chdev -l sys0 -a maxuproc=1024`
- ▶ `no -p -o sb_max=8388608`

Verification: After all modifications, always start a new shell and run the script again (`pe_node_diag`) to ensure that all parameters are changed persistently. Sometimes a reboot is required to activate the changes.

Parallel Environment Runtime for AIX (1.1.0): By default uses RSH communication between nodes. To switch to SSH, edit the `/etc/ppe.cfg` file, and change the following line from:

```
PE_SECURITY_METHOD: COMPAT
```

to

```
PE_SECURITY_METHOD: SSH poesec
/opt/ibmhpc/pecurrent/base/gnu/lib64/poesec_ossh.so m[t=1,v=1.1.0]
```

4.4.2 GPFS tunable parameters affecting HPC performance

If using GPFS within an HPC cluster, performance will be an important factor to improve. Consider changing a set of tunable parameters that depend on the file system size, number of disks, nodes, users, and workload pattern. Along with the current GPFS online documentation, there is additional documentation in the following web site:

<http://www.ibm.com/developerworks/wikis/display/hpccentral/GPFS+Tuning+Parameters#GPFSTuningParameters-nsdMaxWorkerThreads>

Performance impact parameters for HPC workloads (importance):

- ▶ `maxFilesToCache` (high workloads)
- ▶ `maxMBpS` (InifiniBand networks)
- ▶ `maxReceiverThreads` (large number of nodes cluster)
- ▶ `nsdMaxWorkerThreads` (large number of NSDs per node)
- ▶ `numaMemoryInterleave` (Linux)
- ▶ `pagepool` (available memory dependent, random IO and GPFS clients)
- ▶ `prefetchPct` (sequential access)
- ▶ `prefetchThreads` (high number of NSDs/node)
- ▶ `worker1Threads` (high asynchronous or direct IO)

4.4.3 HPC Cluster verifications

This section combines a set of configuration verifications, such as environment variables, startup scripts, and security aspects, to enhance cluster interoperability and reduce the need for troubleshooting in case of a software problem.

- ▶ SSH:
 - Check if file `~/.ssh/known_hosts` is populated with all nodes. If not, use `ssh-keyscan`.
- ▶ Parallel Environment Runtime:
 - Create the `hosts.list` file in your home directory.
 - Check if the `/etc/hosts.equiv` or home directory `.rhosts` has all the node names and if they all are resolvable to IPs (If using “PE_SECURITY_METHOD: COMPAT”).
- ▶ LoadLeveler
 - Create `mpd.hosts` file in your home directory.
- ▶ GPFS
 - Use `mmchconfig` to tune GPFS, using default GPFS configurations will not be ideal for HPC clusters.

4.4.4 Customizing the environment

Developing with Eclipse permits the user to directly customize the building environment from the GUI. Although it can be simple to use, for bigger or higher complexity projects, we can customize the operating system environment to increase interoperability between users. There are examples on how to:

- ▶ Create new modules using the environment modules tool.
- ▶ Do the shell customization.

Using the environment modules tool (RHEL 6.2)

This tool features creation and management of modules to differentiate compilation environments, such as different versions of the compilers.

Note: The tool source code is also available for compilation in UNIX systems. The install directory can be different from the one illustrated here (only for RHEL 6.2).

► Creating new modules

It is possible to create modules based on a specific format, as detailed in Example 4-5.

Example 4-5 How to module from environment-modules

```
# cat /usr/share/Modules/modulefiles/use.own
#%Module1.0#####
##
## use.own modulefile
##
## modulefiles/use.own.  Generated from use.own.in by configure.
##
proc ModulesHelp { } {
    global rkversion

    puts stderr "\tThis module file will add \${HOME}/privatemodules to the"
    puts stderr "\tlist of directories that the module command will search"
    puts stderr "\tfor modules.  Place your own module files here."
    puts stderr "\tThis module, when loaded, will create this directory"
    puts stderr "\tif necessary."
    puts stderr "\n\tVersion $rkversion\n"
}

module-whatis  "adds your own modulefiles directory to MODULEPATH"

# for Tcl script use only
set    rkversion    3.2.7

eval set [ array get env HOME ]
set    ownmoddir    ${HOME}/privatemodules

# create directory if necessary
if [ module-info mode load ] {
    if { ![ file exists $ownmoddir ] } {
        file mkdir $ownmoddir
        set null [open $ownmoddir/null w]
        puts $null
    }
}
"%Module#####"
puts $null "##"
puts $null "## null modulefile"
puts $null "##"
puts $null "proc ModulesHelp { } {"
puts $null "    global version"
puts $null ""
puts $null "    puts stderr \"\tThis module does absolutely
nothing.\""
```

```

puts $null "          puts stderr "\tIt's meant simply as a place holder in
your\""
puts $null "          puts stderr "\tdot file initialization.\""
puts $null "          puts stderr "\n\tVersion \$version\n\"
puts $null "}"
puts $null ""
puts $null "module-what is          \"does absolutely nothing\"
puts $null ""
puts $null "# for Tcl script use only"
puts $null "set      version      3.2.7"
}
}

```

```
module use --append $ownmoddir
```

- ▶ Configuring the environment modules tool:
 - Default directory for the configuration modules: **/usr/share/Modules/modulefiles/**
 - Modules directory can be changed in the file: **/usr/share/Modules/init/.modulespath**
 - In the modules directories, several other directories can be separately created
- ▶ Modules shell initialization:
 - Uses **/etc/profile.d/modules.sh** (or **.csh**) script to initialize modules
 - Compatible shells: **bash, csh, ksh, perl, python, sh, tcsh, and zsh**
- ▶ Modules examples:
 - Null (Example 4-6)
 - Intel MPI compilers (Example 4-7)
 - Intel C/C++ and Fortran compilers (Example 4-8 on page 59)

Example 4-6 Delivered null module (does nothing)

```

# cat /usr/share/Modules/modulefiles/null
#%Module1.0#####
##
## null modulefile
##
## modulefiles/null. Generated from null.in by configure.
##
proc ModulesHelp { } {
    global version
    puts stderr "\tThis module does absolutely nothing."
    puts stderr "\tIt's meant simply as a place holder in your"
    puts stderr "\tdot file initialization."
    puts stderr "\n\tVersion $version\n"
}
module-what is "does absolutely nothing"
# for Tcl script use only
set version "3.2.8"

```

Example 4-7 shows the Intel MPI compilers module.

Example 4-7 Intel MPI compilers module

```

# cat intel/impi-4.0.2.003
#%Module *- tcl *-
##

```



```

## dot modulefile
##
proc ModulesHelp { } {
    global intelversion
    puts stderr "\tAdds 64-bit Intel MPI to your environment"
}
module-whatIs "Adds 64-bit Intel MPI to your environment"
# for Tcl script use only
set intelversion 4.0.2.003
prepend-path PATH /opt/intel/impi/$intelversion/intel64/bin
prepend-path LD_LIBRARY_PATH /opt/intel/impi/$intelversion/intel64/lib

```

Example 4-8 shows the Intel C/C++ and Fortran compilers module.

Example 4-8 Intel C/C++ and Fortran compilers module

```

# cat intel/compilers-11.1.073
#%Module *- tcl *-
##
## dot modulefile
##
proc ModulesHelp { } {
    global intelversion
    puts stderr "\tAdds 64-bit Intel C/C++ and Fortran compilers to your
environment"
}
module-whatIs "Adds 64-bit Intel C/C++ and Fortran compilers to your
environment"

prepend-path PATH /opt/intel/Compiler/11.1/073/bin/intel64
prepend-path MANPATH /opt/intel/Compiler/11.1/073/man/en_US
prepend-path LD_LIBRARY_PATH /opt/intel/Compiler/11.1/073/lib/intel64

```

Shell environment customization (RHEL/SUSE)

Under Linux, there are a couple of ways to provide a better base environment to all users that develop on a specific node (Example 4-9):

1. Add custom profile scripts under */etc/profile.d/*. After */etc/profile* is called, all files inside the */etc/profile.d/* are called. The shell must be restarted or the file loaded manually with the **source** command for the script contents to be read.

Example 4-9 How to add GPFS path to all users

```

# echo "export PATH=\$PATH:/usr/lpp/mmfs/bin" >> /etc/profile.d/gpfs.sh
# cat /etc/profile.d/gpfs.sh
export PATH=\$PATH:/usr/lpp/mmfs/bin

```



Managing projects using the Eclipse and PTP framework

This chapter describes how to set up Eclipse and PTP Framework either by importing existing code or starting new projects. We also explain how to manage these projects from a local or remote site perspective. Through the chapter, we present different customizations involving building, running, and debugging parallel applications.

The following topics are covered:

- ▶ Available project scenarios:
 - Synchronized
 - Remote
 - Local
- ▶ Creating a new parallel application
- ▶ Importing an existing parallel application
- ▶ Edit features of Eclipse
- ▶ Customize Eclipse to build and run parallel applications
- ▶ Building and running an application
- ▶ Debugging using Eclipse
- ▶ Integrating external applications

Note: The Eclipse and PTP Framework client side is available for Linux (32/64 bit), Mac OSX (64 bit), and Windows (32/64 bit) distributions. For further details, check “Supported operating systems (software)” on page 3.

5.1 Available project scenarios

Depending on what resources we have or which target the code must run against, it is possible to use different approaches for each case. In this chapter, we present three specific code development layouts supported by Eclipse. From the complex one to the simple one, we have:

- ▶ Synchronized projects
- ▶ Remote projects
- ▶ Local projects

In Table 5-1, we can evaluate some differences between the three methods.

Table 5-1 Properties for the three project scenarios (client view)

Property	Synchronized	Remote	Local
Ability to build, run and debug	Remote/Local	Remote	Local
Development for x86 architecture	Remote/Local	Remote	Local
Development for Power architecture	Remote	Remote	NA
Code indexing	Fast (local)	Slow (remote)	Fast (local)
Symmetric dual-site code development (same toolchain)	Yes x86 (32/64 bit)	No	No

5.1.1 Synchronized

Synchronized projects mirror project files on the local and remote machines. Build, run, and debug can take place on either or both. They use SSH connections to connect to a remote server and, currently, the Git software revision control subsystem to maintain the coherency between local files and remote ones. Figure 5-1 on page 63 demonstrates the workflow for the synchronized scenario.

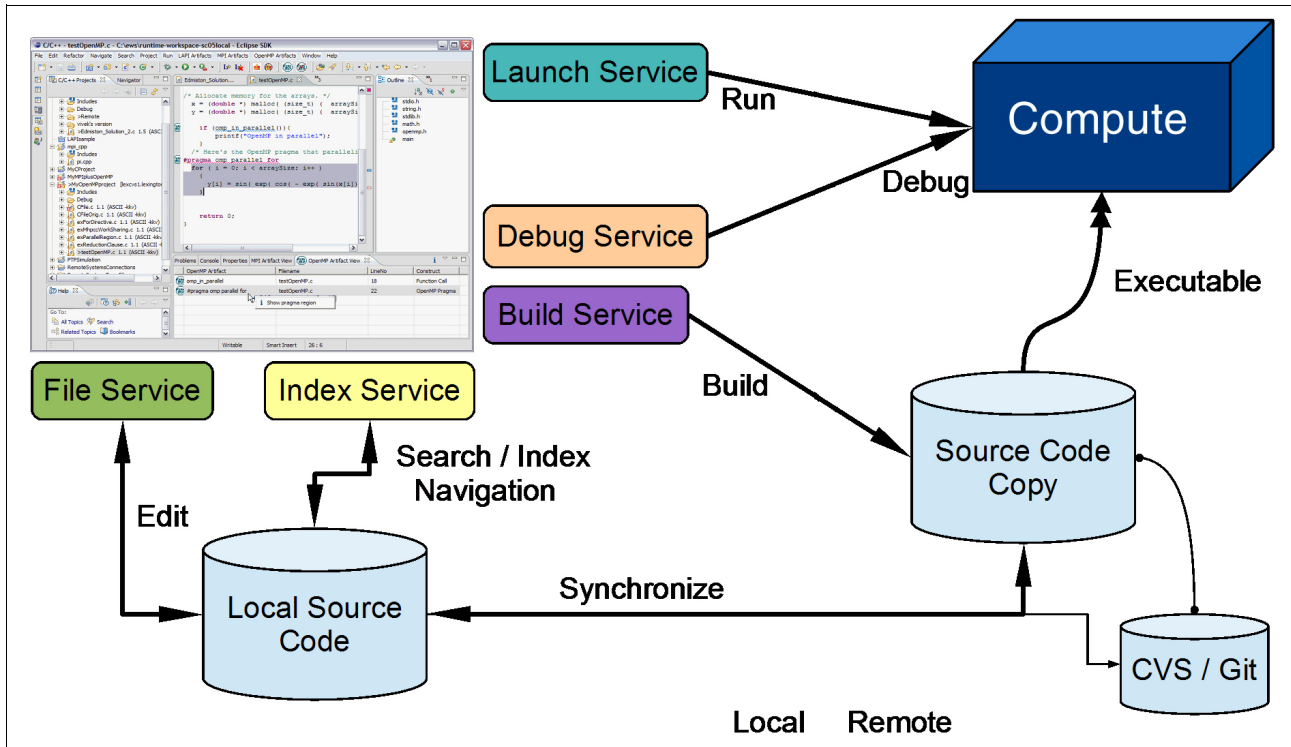


Figure 5-1 Communication workflow for synchronized Eclipse projects

The advantages of synchronized projects:

- ▶ Ability to use local machines to continue code development while the remote machine is not available.
- ▶ Maintain fast indexing of local code for speedy editing tasks like code navigation, code assist, outline, and search.
- ▶ Ability to develop code on different architectures without forcing eclipse to move to the same respective architectures.
- ▶ Ability to create and index Fortran projects. Remote projects do not provide indexing on Fortran code.

To create a synchronized project:

1. Select **File** → **New** → **Other**. A new dialog appears (see Figure 5-2 on page 64), and in the Remote folder you can create either a:
 - Synchronized C/C++ project
 - Synchronized Fortran project¹
2. Under the same folder, you can also convert an existing C/C++ or Fortran local project to a synchronized one with the option: Convert a C/C++ or Fortran project to a synchronized project

¹ Synchronized Fortran project includes all the features of a C/C++ project too

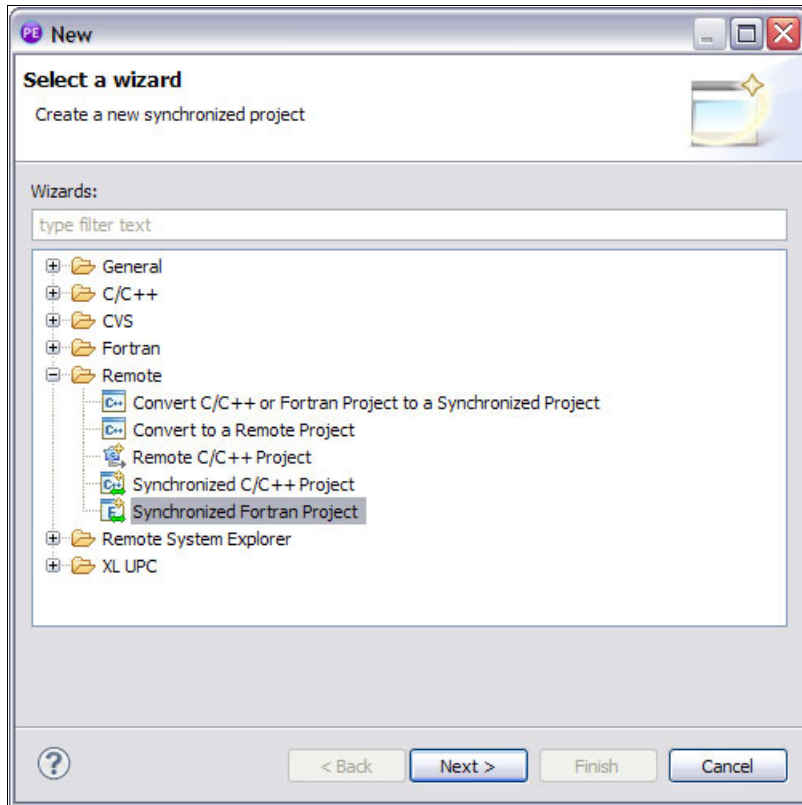


Figure 5-2 Remote folder in new project wizard

5.1.2 Remote

Remote projects are based on a remote toolchain to build, run, and debug the code. They rely on SSH connections to connect to the targeted host for edit, index, build, run, and debug commands. Figure 5-3 on page 65 demonstrates the workflow for the remote scenario.

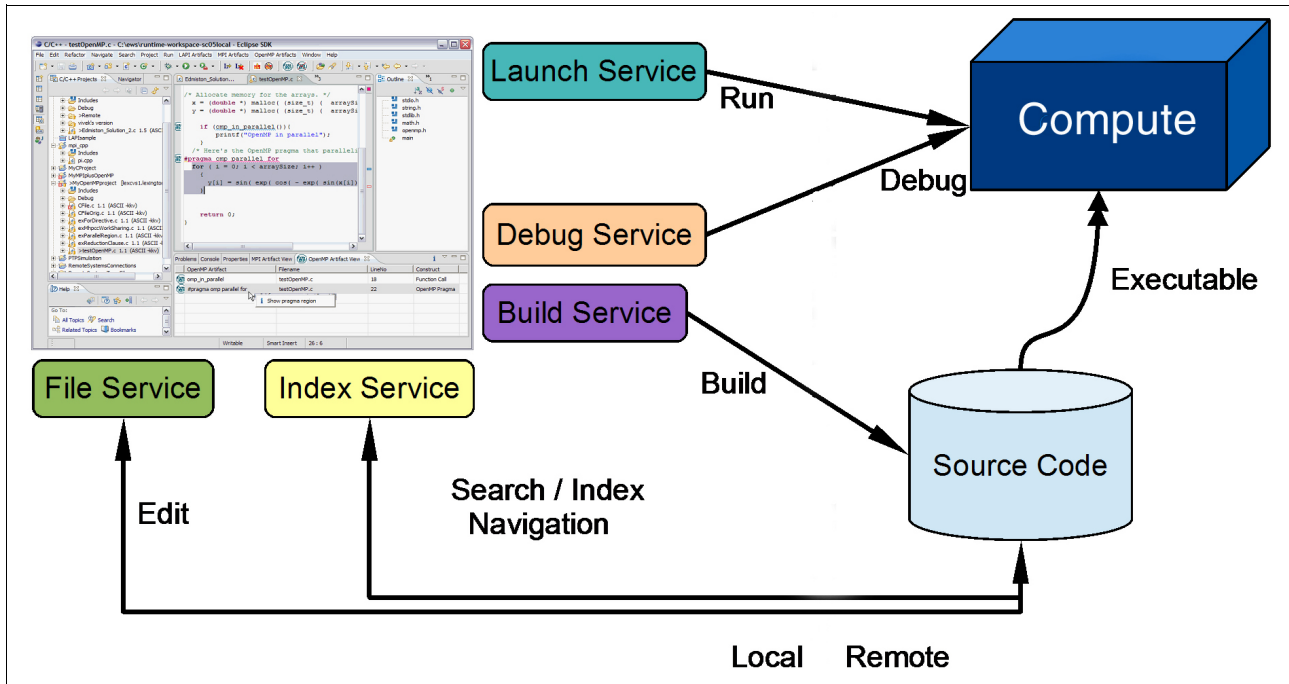


Figure 5-3 Communication workflow for remote Eclipse projects

Advantages of remote projects:

- ▶ No space requirements for code development in the local machine due to absence of code locally (the only space consumed will be for Eclipse internal management files). Also for security reasons, some developers might prefer not to store source code on the local machine.
- ▶ Decreased requirements for CPU performance on local machine because indexing, build, run, and so on, are all on the remote machine, and the local machine only needs to run Eclipse.
- ▶ Ability to develop code on different architectures without forcing eclipse to move to the same respective architectures.

To create remote projects, select **File** → **New** → **Other**. A new dialog (see Figure 5-2 on page 64) appears. In the Remote folder you can create a Remote C/C++ project. Under the same folder, you can also convert an existing C/C++ project to a remote one with the option, Convert to a remote project.

5.1.3 Local

Local projects are based on a local toolchain to build, run, and debug the code. They have no dependencies on other machines and therefore can be more reliable and fast (assuming the same code sizes compared with the other two scenarios).

Advantages of local projects:

- ▶ Easy to start with and has no external hardware requirements
- ▶ Local host has full control of development availability

Disadvantages of local projects:

- ▶ Most local machines (for example, mobile computers and workstations) do not provide the best environment for developing and running large parallel codes

There are two ways to create local projects:

- ▶ **File** → **New** → **<PROJECT>**
- ▶ **File** → **New** → **Other**

For the last option, a new window appears, and in the C/C++ and Fortran folders you can choose a local project.

5.2 Creating a new parallel application

In this section, we demonstrate how to create a simple Eclipse managed *Hello World* C and MPI code project. The Eclipse project example is used on an x86 architecture platform using a synchronized scenario. For the example in this section, we used the following resources and respective requirements.

Resources:

- ▶ Local machine: x86_64 (Intel i5 560 M) with RHEL 6.2
- ▶ Remote machine: x86_64 (Intel Xeon X5570) with RHEL 6.2

Local machine requirements:

- ▶ Java 1.6, or later installed
- ▶ IBM PEDE client installed

Remote machine requirements (for a synchronized project):

- ▶ Reachable through SSH (with password or ssh key)
- ▶ Have Git installed for synchronization
- ▶ IBM PEDE server installed (check Table 4-1 on page 48)

To create a new parallel application:

1. After you completed those prerequisites, click **File** → **New** → **Other (or Ctrl+N)**. A new dialog appears as shown in Figure 5-2 on page 64.
2. Under the folder Remote, select **Synchronized C/C++ Project**, and click **Next**. The New Synchronized Project window is displayed, as shown in Figure 5-4 on page 67.

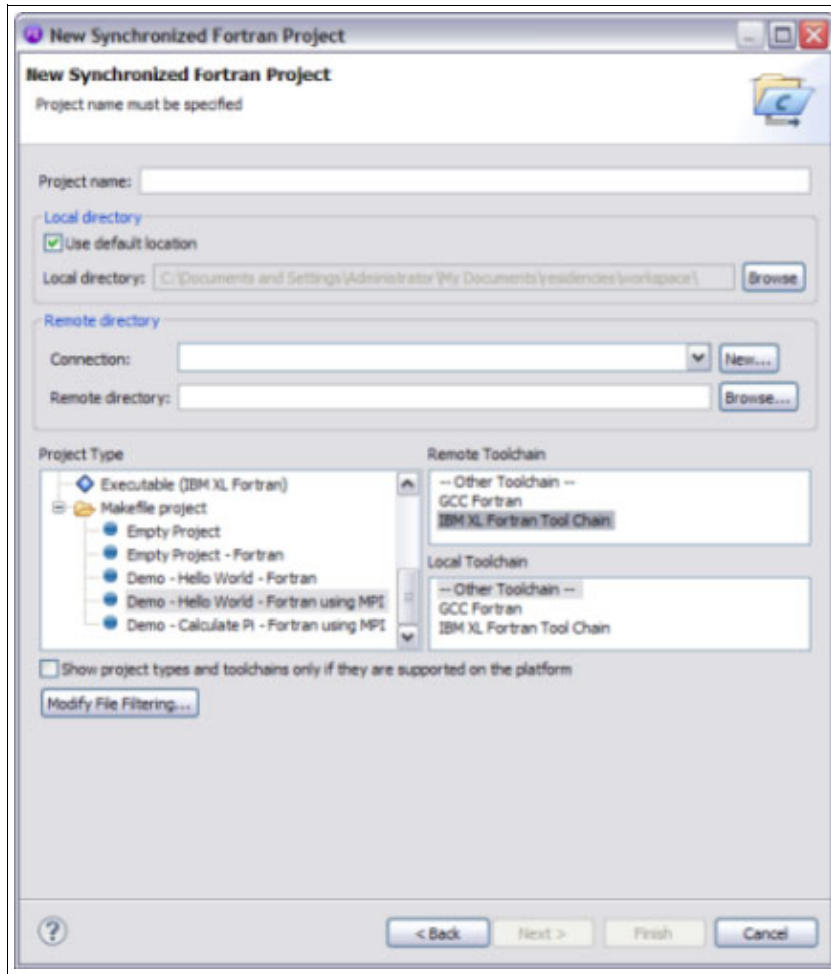


Figure 5-4 New synchronized C/C++ project wizard

In Figure 5-4:

1. Type a name for the project.
2. At the Local directory group, type a path to the project local directory, or use the default location (based on your workspace directory and project name).
3. At the Remote directory group, select the connection to the remote host (if already created) or create a new one:
 - a. To create a new connection, click **New**, and a new window named Generic Remote Host is displayed, as shown in Figure 5-5 on page 68.
 - b. Give a name to the target connection (target name).
 - c. At the Host Information group, select the **Remote host** option, specify the fully qualified host name or IP address of the remote host (Host), and the user that will login (User). For the ssh authentication method, there are two possible ways: either through passphrase or through public key authentication:
 - i. For password authentication, type the selected user password (password).
 - ii. For public key authentication, specify the path location for the private ssh key file of the selected user. If the private key file has a passphrase, enter it on the respective field (passphrase).

- d. Click the **Finish** to complete the Target Environment Configuration. The window returns to the new synchronized project dialog.

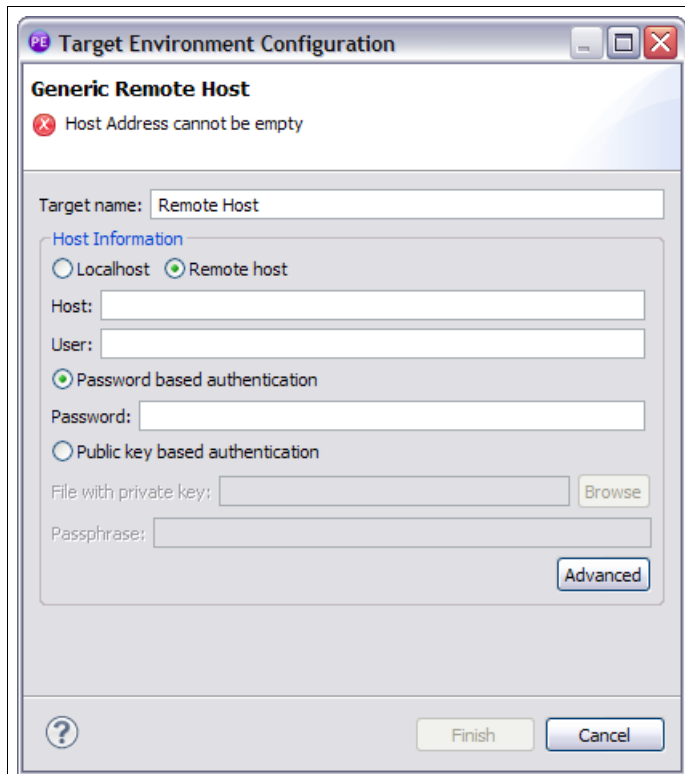


Figure 5-5 Target environment configuration wizard

4. On the Remote directory group, click **Browse**, and select the project remote directory path (If the directory does not exist, it will be created).
5. Inside Project type, under the Executable folder, select **MPI Hello World C Project**.
6. Inside Remote Toolchain, select **Remote Linux GCC Tool Chain**.
7. Inside Local Toolchain, select **Linux GCC**. If you do not require a build on the local machine, a local toolchain is not required at all.
8. Click **Next**.
9. In the Basic Settings window, Figure 5-6 on page 69, use the defaults, and click **Next** or edit the following items:
 - a. Code author
 - b. Copyright notice
 - c. Hello world greeting
 - d. Source directory name

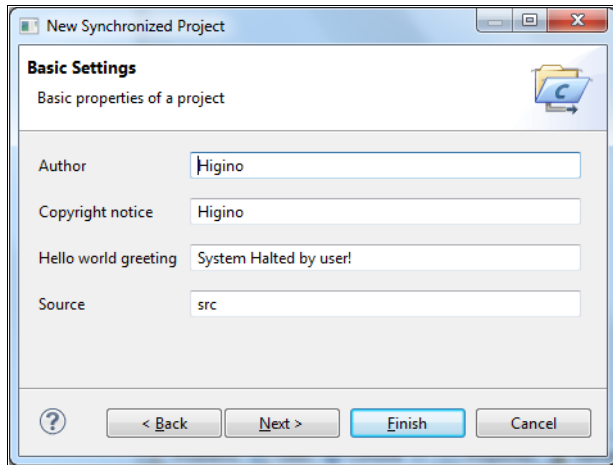


Figure 5-6 Basic settings dialog for MPI project

10. This project is specified before as an MPI project; therefore, the next window presents you with MPI Project Settings (see Figure 5-7) where it can be used to add paths to the build. For this example, accept the default settings by clicking **Next**.

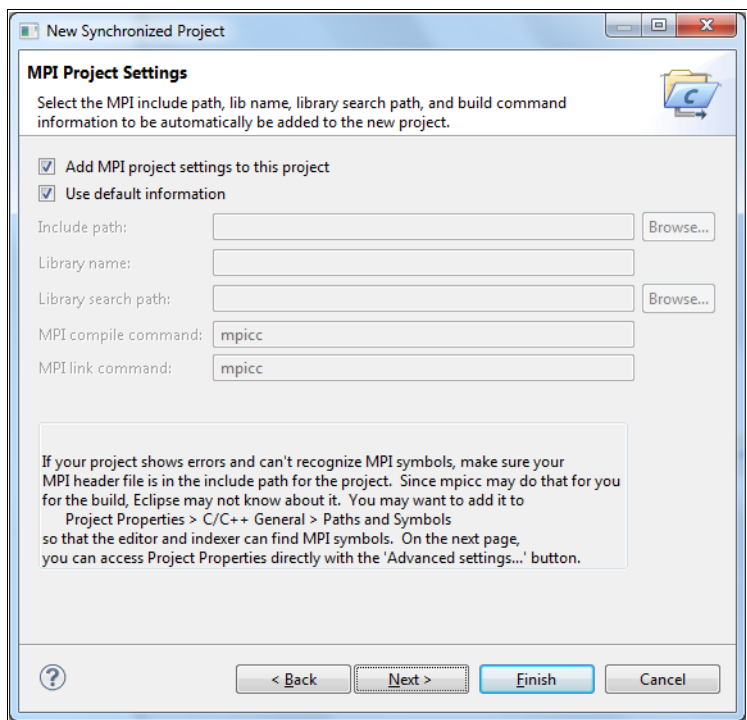


Figure 5-7 MPI project settings dialog

11. The last window (Figure 5-8 on page 70) shows two pre-built build configurations. Accept the two default selected configurations to include them in your project, and finally click **Finish**.

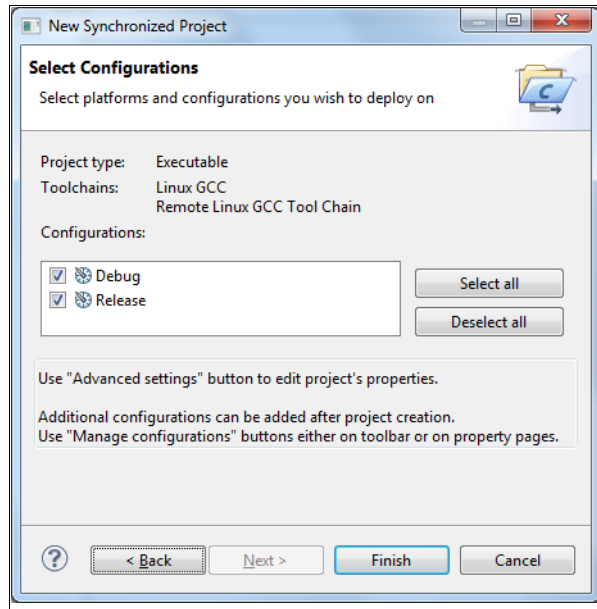


Figure 5-8 Available pre-built configurations for the MPI based hello world project

Eclipse now creates the Hello World project and synchronizes all data with the remote host.

5.3 Importing an existing parallel application

In this section, we demonstrate how to import an existing C based code (makefile project) to Eclipse using a synchronized scenario. When creating a project with existing code, it can initially be located either locally or remotely. For the presented example, we are using a code that already exists on the local machine and the synchronized project setup copies automatically the code to the remote machine.

Resources:

- ▶ Local machine: x86_64 (Intel i5 560M) with RHEL 6.2
- ▶ Remote machine: x86_64 (Intel Xeon X5570) with RHEL 6.2

Local machine requirements:

- ▶ Java 1.6, or later installed
- ▶ IBM PEDE client installed

Remote machine requirements (for a synchronized project):

- ▶ Reachable through SSH (with password or ssh key)
- ▶ Have Git installed for synchronization
- ▶ IBM PEDE server installed (check Table 4-1 on page 48)

To import an existing parallel application:

1. After you have all the prerequisites, click **File** → **New** → **Other (or Ctrl+N)**. A new window is displayed, as shown in Figure 5-2 on page 64.
2. Under the Remote folder, select **Synchronized C/C++ project**, and click **Next**. The New Synchronized Project window is displayed, as shown in Figure 5-4 on page 67.

In this window (Figure 5-4 on page 67):

1. Type a name for the project.
2. At the Local directory group, type a path to the directory where the code resides. This directory will be used as your local project directory.
3. At the Remote directory group, select the connection to the remote host (if already created) or create a new one:
 - a. To create a new connection, click **New**, and a new window named Generic Remote Host is displayed, as shown in Figure 5-5 on page 68.
 - b. Give a name to the target connection (target name).
 - c. At the Host Information group, select the Remote host option, specify the fully qualified host name or IP of the remote host (host), and specify the user that will login (user). For the authentication method, there are two possible ways, either using passphrase or through public key authentication:
 - i. For the first method, type the selected user password (password)
 - ii. For the second method, specify the path location for the private key file of the selected user. If the private key file has a passphrase, enter it on the respective field (passphrase)
 - d. Click **Finish** to complete the Target Environment Configuration.
4. On the Remote directory group, click **Browse**, and select the **Project remote directory path** (path must be already created).
5. Inside the Project type, under the folder Makefile project, select Empty Project.
6. Inside the Remote Toolchain, select **Remote Linux GCC Tool Chain**.
7. Inside the Local Toolchain, select **Linux GCC**.
8. Click **Next**.
9. The last window presents you with two pre-built build configurations. Accept the two default selected configurations to include them in your project, and click **Finish**.

Eclipse now synchronizes all of the files between the two sites. After the synchronization process is completed, the project can be built and run.

Assumptions: This section example is based on a makefile that is built with the `make a11` command.

5.4 Building and running an application

In this section, we demonstrate how to create Eclipse builds (targets) and profile runs using an existing code (makefile based) already imported to Eclipse, and inside a synchronized scenario.

When you create a synchronized Eclipse Project, there are two locations implicit: local and remote. Creating additional build configurations depend only on the toolchain you choose. So we can create new configurations for different compilers or event environment variables depending on the objectives, and when required inspect these build properties by right-clicking the project directory and then clicking **Properties**. This procedure applies to the

active build configurations. So whenever you need to edit the properties of a certain build, you need either to edit that configuration directly or activate it first and then go through the same process again. The next steps demonstrates how to go through both ways:

To create a new toolchain build configuration in Eclipse:

1. Go to **Project** → **Build Configurations** → **Manage**.
2. When Figure 5-9 is displayed, click **New**.

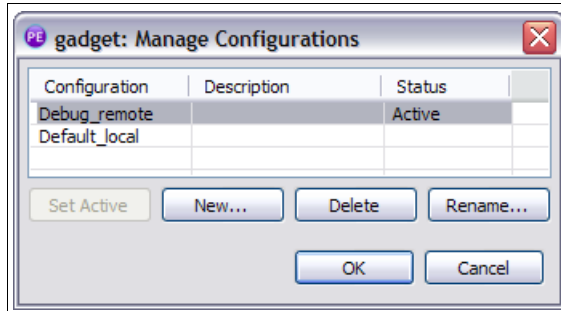


Figure 5-9 Manage build configurations

3. When Figure 5-10 is displayed:
 - a. Type the new build configuration name.
 - b. Type a description for that build.
 - c. Select the creation method of this new build, which might be one of the following items:
 - i. Existing configuration (can be one of already created build configurations)
 - ii. Default configuration (if you have set some one already)
 - iii. Import from projects (other projects that you have already configured in Eclipse)
 - iv. Import predefined configurations (delivered with Eclipse)
 - d. Click **OK** when you are satisfied with the options.

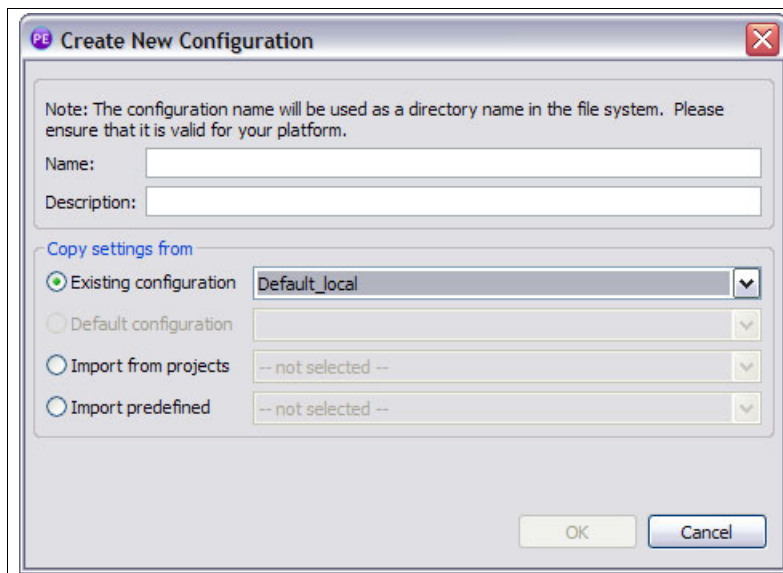


Figure 5-10 Create new build configuration wizard

The window in Figure 5-9 is displayed again, now with the newly created configuration.

4. If you want, select it and make it active. The options, **Delete** and **Rename** are also available.
5. To exit this management area, click **OK** to accept all changes or **Cancel** to discard them.

5.4.1 Building (using targets)

Eclipse is flexible enough to allow you to configure the build environment in many details, so the build process can be entirely controlled through its GUI. For either a C/C++ or Fortran project, it allows you to change several build settings, for example:

- ▶ Build directory (root folder by default), command (*make* by default), and targets.
- ▶ Customize environment variables that are exported at build time.
- ▶ Enable and disable build logging.
- ▶ Choose binary and error parsers that best fits.
- ▶ Switch between toolchain.
- ▶ Change toolchain flags (only for Eclipse managed build). For makefile-based projects where the makefile content is not managed by Eclipse, change the toolchain settings (compiler, flags, steps,) manually.

All build properties are accessible by right-clicking the project folder and then selecting **Properties** → **C/C++ Build (or Fortran Build)**. The project properties window along with sections for build configuration are shown in Figure 5-11.

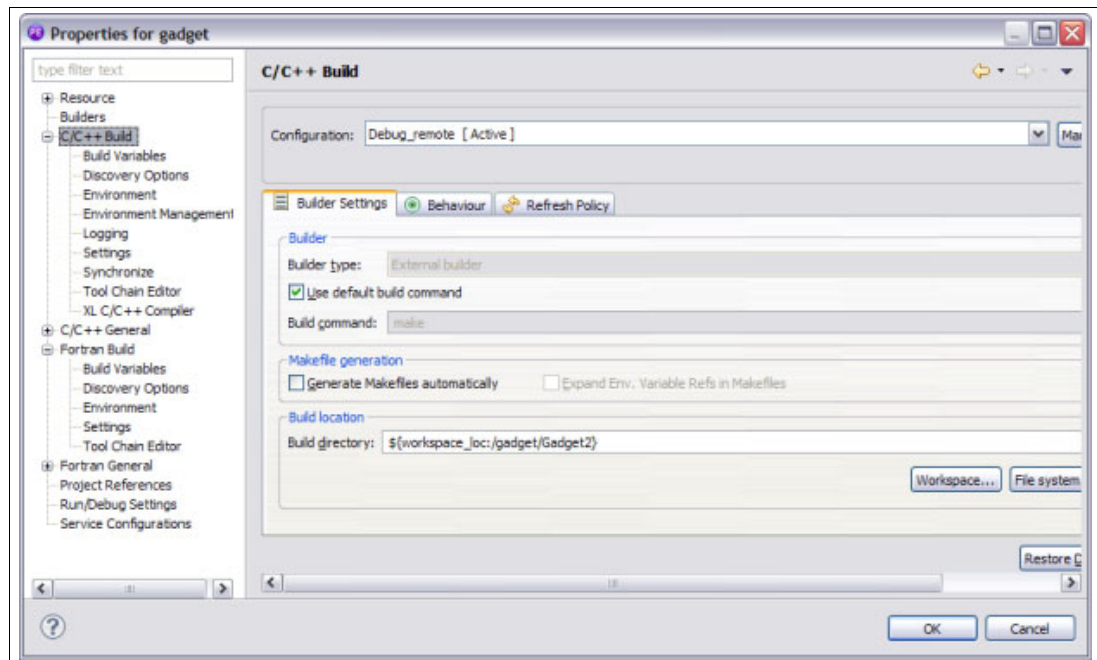


Figure 5-11 Project properties window

The simplest way to build a synchronized project is to right-click the project folder and then click **Build project**, as shown in Figure 5-12 on page 74. Eclipse uses connection configuration specified when the project was created to log into the remote target and then run **make all** to build the application. A similar procedure can be followed to clean up build artifacts (right-click the project folder and then select **Clean Project**) where it executes **make clean**.

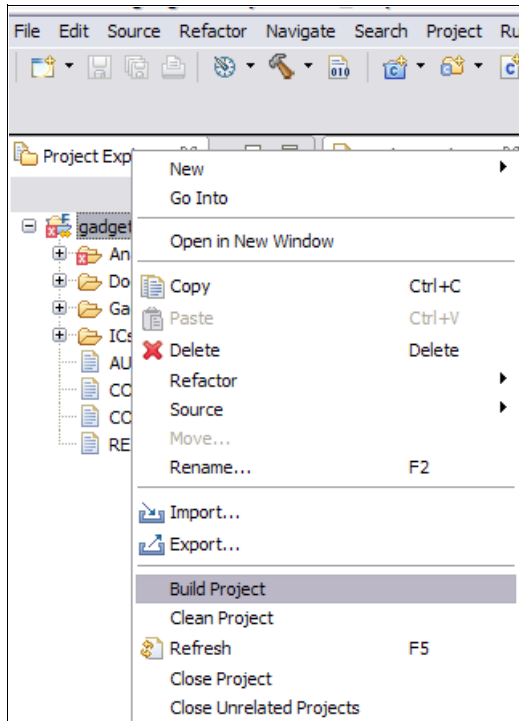


Figure 5-12 Build project

Custom build targets

It is relatively common for projects requiring makefile targets to build and clean up other than all and clean defaults. Thus, we can create specific build targets for the project without changing the toolchain or creating an entirely new configuration (as the one presented in 5.4, “Building and running an application” on page 71).

Custom build targets are often used when a makefile project has several ways of building a program or when there are different targets inside the makefile. Therefore, you must execute different targets.

When creating targets, you can either use the Project Explorer view or use the Make Target view. To do this, make sure you are already in the C/C++ or Fortran perspective. Go to **Window** → **Show View** → **Make Target**.

The Make Target view appears at the right-top corner of the Eclipse window (see Figure 5-16 on page 77).

Depending on your current view, the Make Target view might be shown or not in the Show View submenu of the Menu Window. To open it, in these cases, you can either switch back to the C/C++ or Fortran views first or go to **Window** → **Show View** → **Other (Alt+Shift+Q)** → **Make** → **Make Target**.

In this case, the Make Target view appears at the bottom of Eclipse window, as shown in Figure 5-13 on page 75.

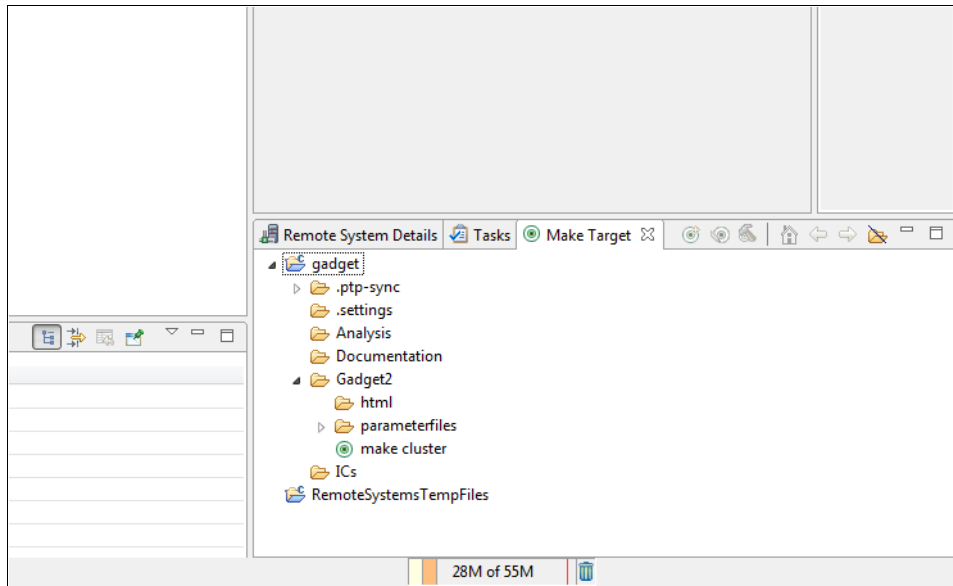


Figure 5-13 Alternative location for the make target view

To actually create a target example, we use the Project Explorer:

1. Select a directory inside the Project Explorer where you have a makefile, and right-click it.
2. Select **Make Targets** → **Create**, as shown in Figure 5-14 on page 76.

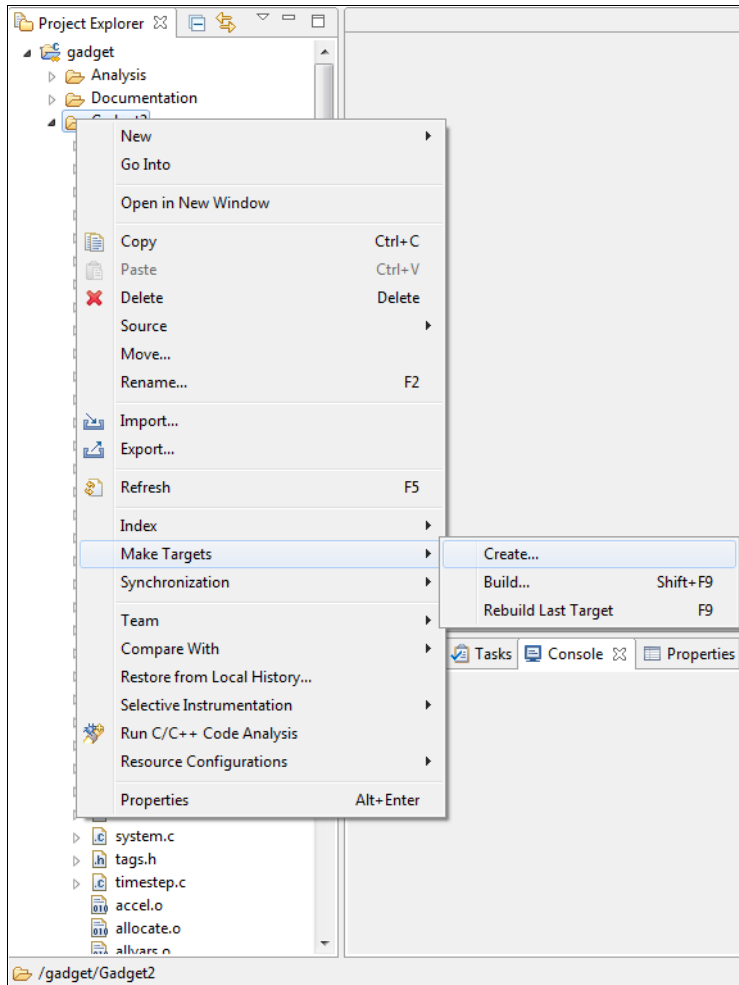


Figure 5-14 Create targets from the project explorer view

3. A new window is displayed named Create Make Target (see Figure 5-15 on page 77):
 - a. Type a new name for this target.
 - b. Customize the Make Target if needed.
 - c. Customize the Build Command.
 - d. Leave all the other options default. After you finish, click **OK**.

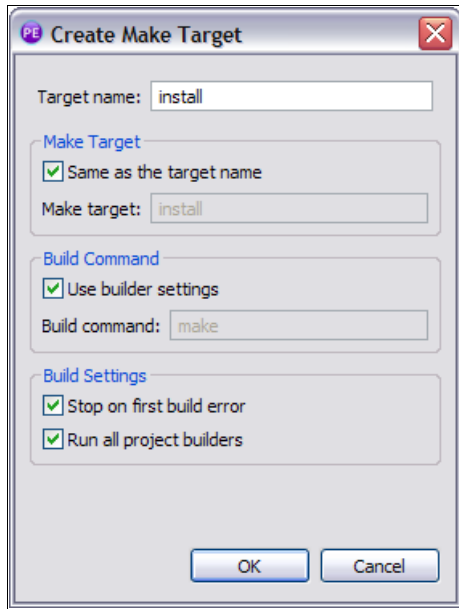


Figure 5-15 Create make target dialog

The Target will now appear on the Make Target view under the same directory you issued the creation. For example, Figure 5-16 shows two custom build targets called *install* and *cluster*.

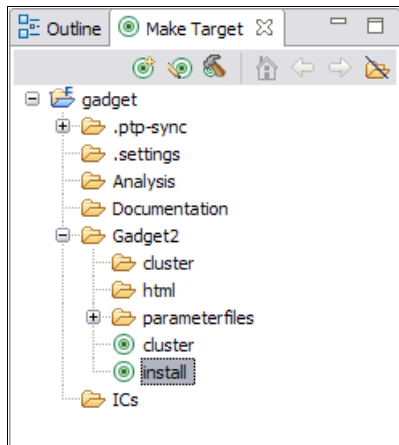


Figure 5-16 Make target view

To execute a specific build target, we can do it through the:

1. Eclipse menu:
 - a. Select **Project** → **Make Target** → **Build (Shift+F9)**.
 - b. Select a target build configuration and click **Build**.
2. Project Explorer:
 - a. Right-click over the directory containing a Make Target configuration.
 - b. Select **Make Targets** → **Build (Shift+F9)**.
 - c. Select a target build configuration, and click **Build**.
3. Make Target view:
 - a. Double-click the target configuration or right-click it, and select Build Target.

5.4.2 Running

When your code is built and becomes ready to be tested, you can either execute it and watch the whole process from beginning to the end, debug it, or profile it. There is also a fourth option that is available for external applications, but we talk about it at the end of this chapter. In Eclipse, we can create separate configurations for each of these three types:

- ▶ Run configuration
- ▶ Debug configuration
- ▶ Profile configuration

All three are separate configurations (although they can be created from each other) and therefore run independently. When a specific configuration is created, it is automatically made available to the other two options because some of the configurations are shared between them and others are specific, for example:

- ▶ After creating an Eclipse project, and building it, usually a run configuration has to be configured for the program to run (unless it runs in a command line or is used by a third party program). After creating this *run configuration*, you can also debug it and profile it. If necessary, it is possible to edit these configurations in debug or profile mode to access the specific options for each mode.
- ▶ For example, if a run configuration is created, both the debug and profile configurations are available for the same run configuration options but with specific configurations (debug options and profile options respectively) that are not configured by the time that the run configuration is created (assumes default values).
- ▶ External applications mode does not share these configurations, as detailed in the “Integrating external applications” on page 91.
- ▶ You can access these modes through the Eclipse Run menu or using the Eclipse quick buttons, as shown in Figure 5-17.



Figure 5-17 Eclipse quick access buttons for debug, run and profile options

Run configuration

To create, edit, and delete run configurations:

1. Select **Run** → **Run Configurations**.
2. The Run Configurations window is displayed, shown in Figure 5-18 on page 79. In this window, you have two separate areas, the left side is where you manage your configurations, and the right side is where you actually edit the configurations for the selected run configuration.

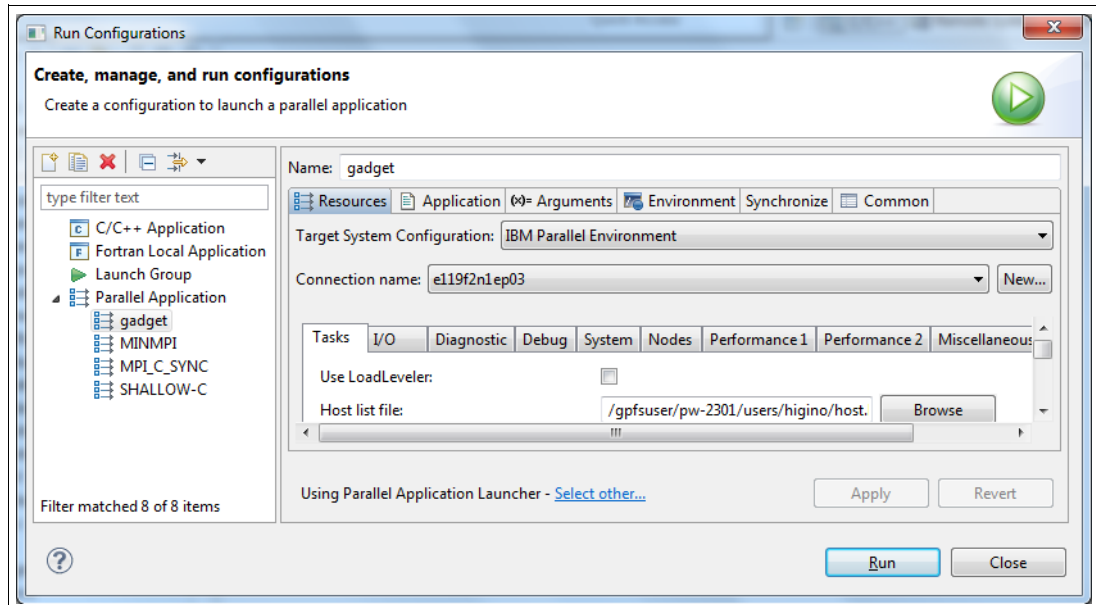


Figure 5-18 Run configurations dialog

3. Figure 5-24 on page 83 illustrates the buttons you can use to create, duplicate (for already created configurations), and delete configurations, respectively from left to right.
4. In Run Configurations (Figure 5-18), you have the following predefined configurations from where you can create customized configurations:
 - C/C++ application
 - Fortran local application
 - Launch group
 - Parallel application
5. To create an example of a *Parallel Application*, select it, and click **New Launch configuration** (first button as shown in Figure 5-24 on page 83). A new subentry label of the Parallel Application label is displayed. You can name it and customize it with the following options, which are available on the right side of the Run Configurations window:
 - Resources (Figure 5-19 on page 80).
 - Application (Figure 5-21 on page 81).
 - Arguments (Figure 5-22 on page 81).
 - Environment (Figure 5-23 on page 82).
 - Synchronize
 - Common
6. This mode represents the baseline configuration of a run, and it shares all the configurations with the other two modes (except synchronize for the profile mode). The most important tabs are the Resources, Application, Arguments, and Environment tabs. Inside these tabs, we can edit:
 - The engine that distributes the processes and their respective options. Currently it is the available interface to several target system configurations, including IBM Parallel Environment (PE) and LoadLeveler, as shown in Figure 5-20 on page 80.
 - The nodes that are used to run the application.
 - The application path (available in all nodes).
 - The application arguments and alternative working paths.
 - The definition of additional environment variables.

7. To save the changes, click **Apply** or just click **Run** to save them and run the configuration.

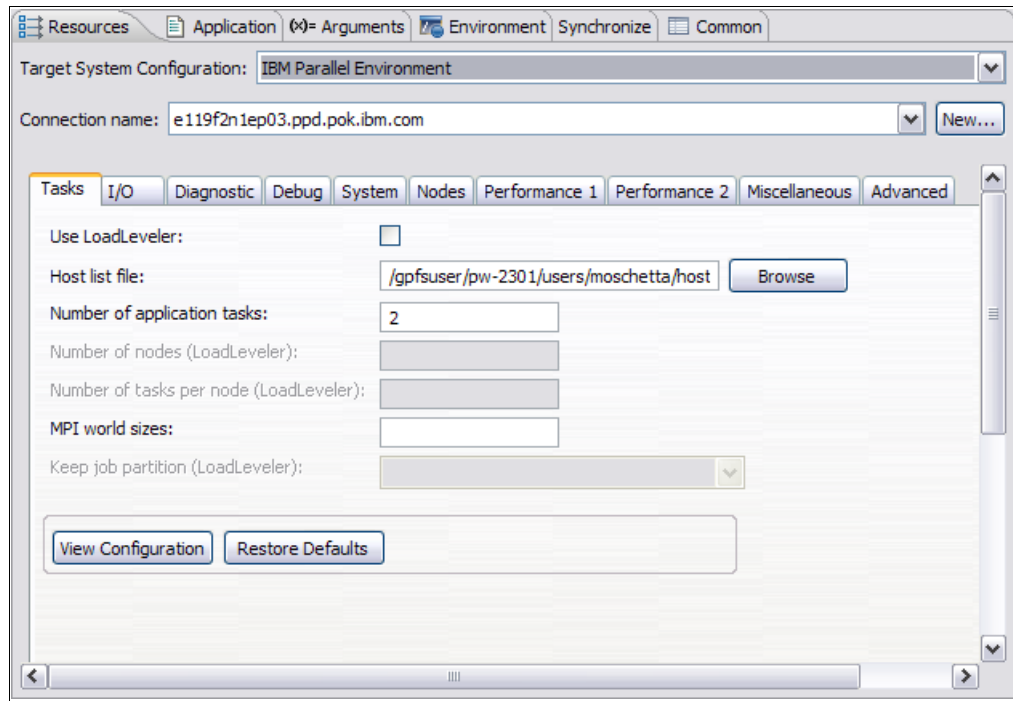


Figure 5-19 Run configurations (resources tab)

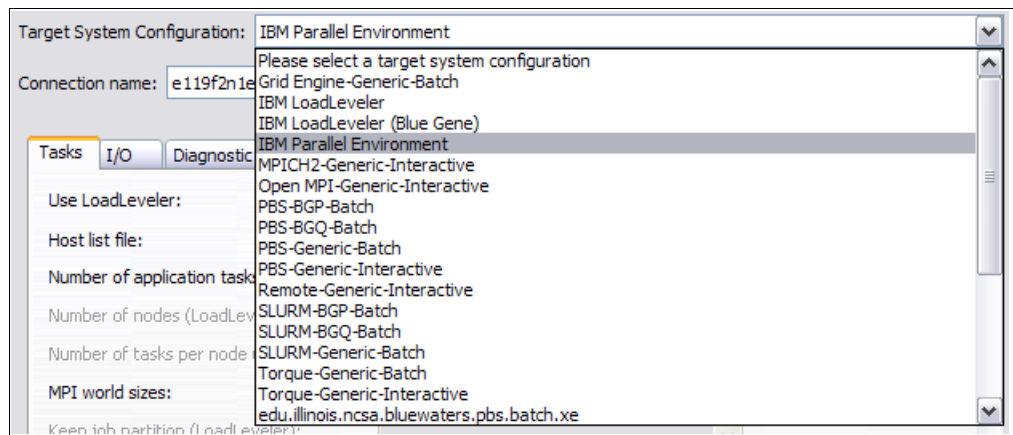


Figure 5-20 List of target system configurations in the resources tab

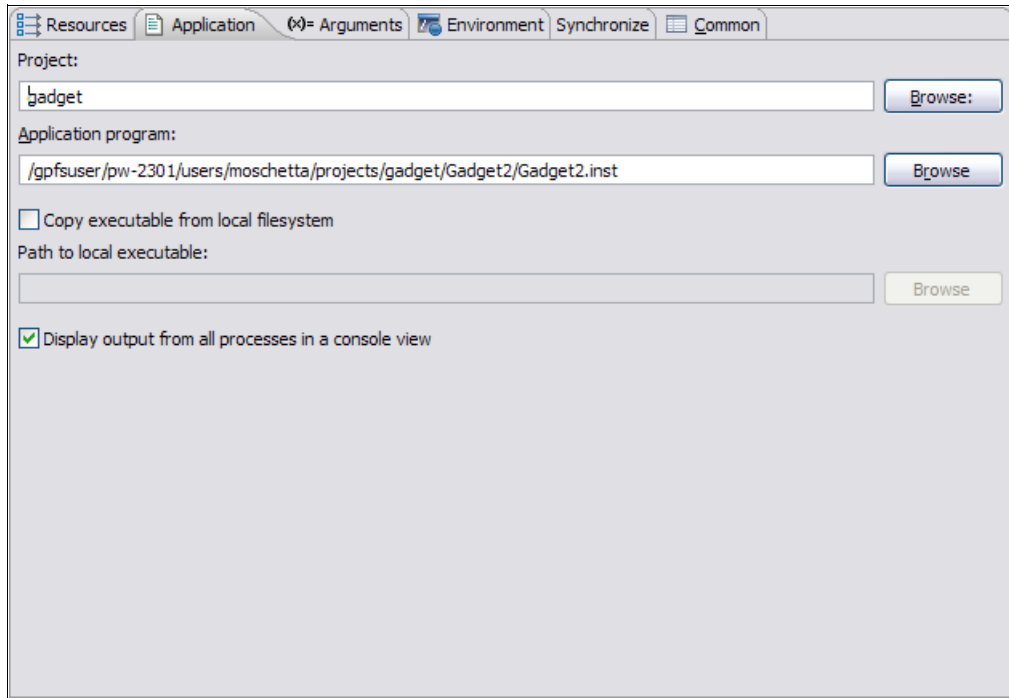


Figure 5-21 Run configurations (application tab)

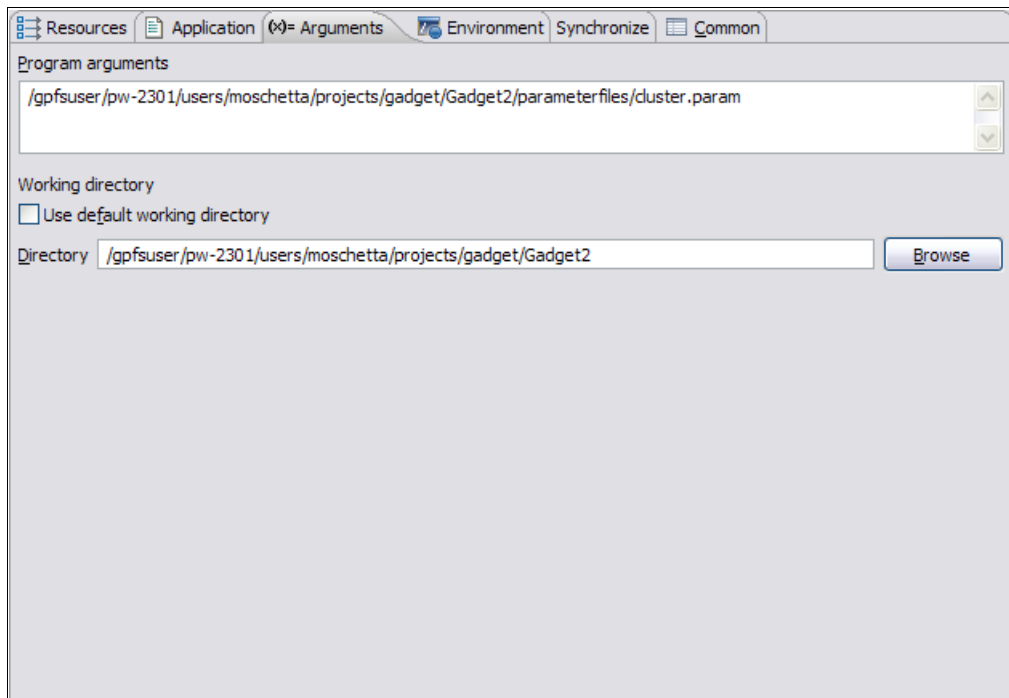


Figure 5-22 Run configurations (arguments tab)

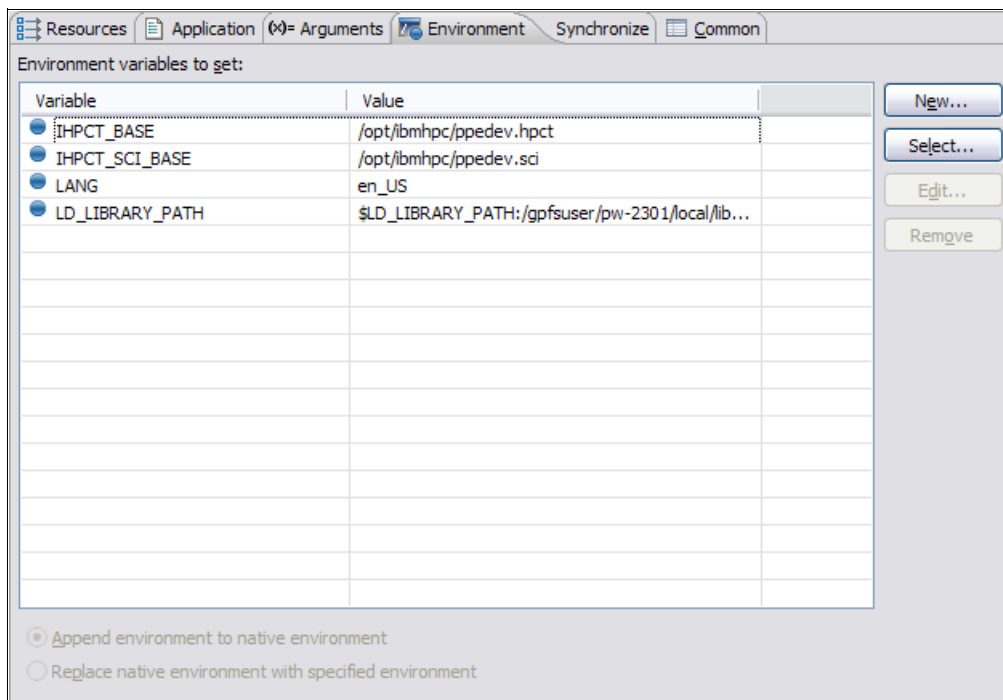


Figure 5-23 Run configurations (environment tab)

Detailed information about how to create a run configuration for a parallel application and submit it to any job scheduler supported, as for instance IBM PE or LoadLeveler, can be performed with the following menu path **Help** → **Help Contents** → **Parallel Tools Platform (PTP) User Guide** → **Running Parallel Programs**.

Debug configuration

To create, edit, and delete debug configurations:

1. Select **Run** → **Debug Configurations**.
2. The Debug Configurations window is displayed, shown in Figure 5-25 on page 84. In this window, you have two separate areas, the left side where you manage your configurations and the right side where you actually edit the configurations for the selected debug configuration.

Figure 5-24 on page 83 illustrates the buttons you can use to create, duplicate (for already created configurations) and delete configurations, respectively from left to right.

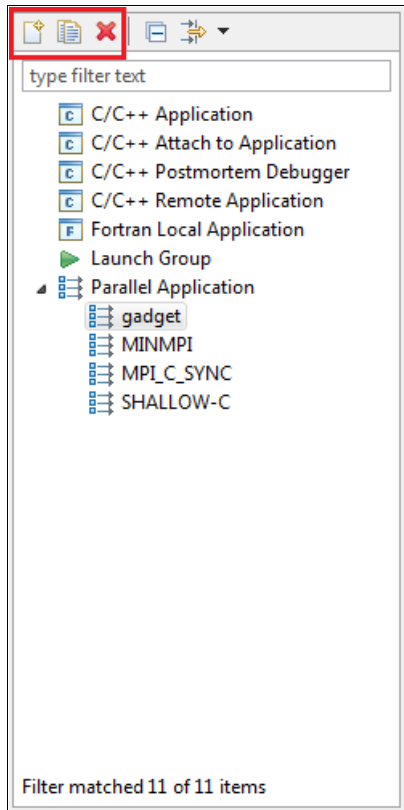


Figure 5-24 Eclipse configuration management buttons

3. In the Debug Configurations window (Figure 5-24), you have the following predefined configurations from where you can create customized configurations:
 - C/C++ application
 - C/C++ attach to application
 - C/C++ postmortem debugger
 - C/C++ remote application
 - Fortran local application
 - Launch group
 - Parallel application
4. To create an example of a Parallel Application, select it, and click **New launch configuration** (first button from Figure 5-24). A new subentry label of the Parallel Application label is displayed. You can name it and customize it with the following options, which are available on the right side of the Debug Configurations window:
 - Resources
 - Application
 - Arguments
 - Debugger
 - Environment
 - Synchronize
 - Source
 - Common
5. As the name suggests, the specific configuration in this mode is the Debugger tab. Inside that tab, choose which debugger to use and what options it uses, as shown in Figure 5-25 on page 84.

- To save the changes, click **Apply**, or click **Debug** to save them and run the debug configuration.

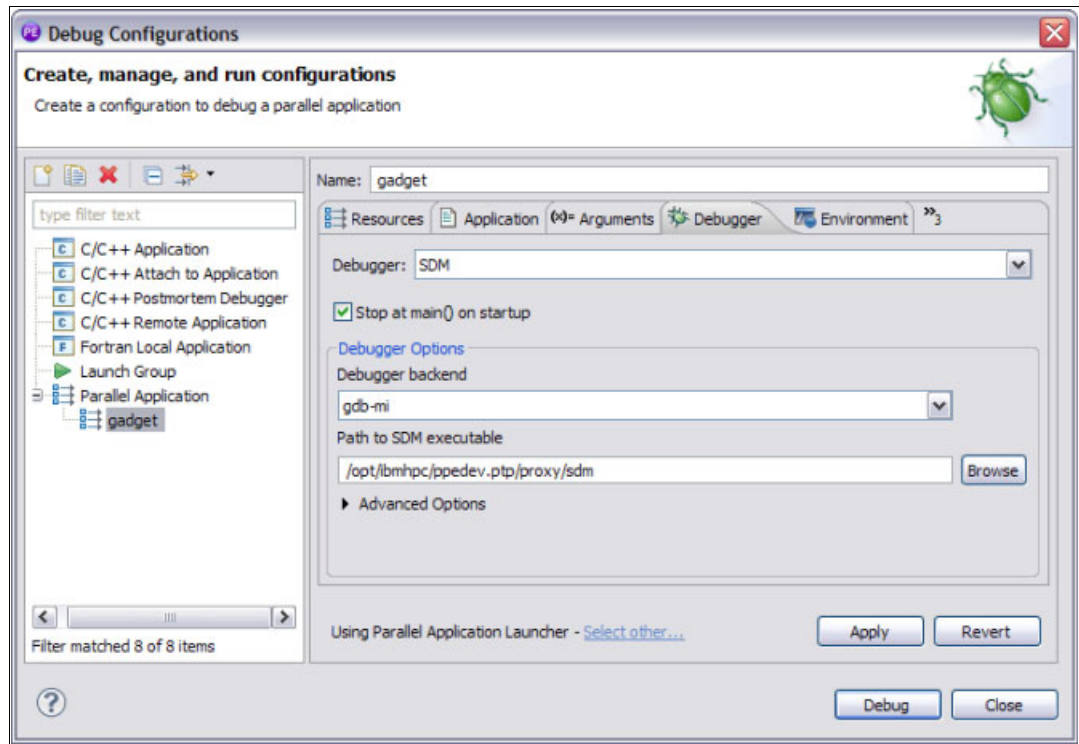


Figure 5-25 Debug configurations dialog

The Eclipse PTP parallel debugger capabilities and more details about how to set up a debug configuration properly are described in “Eclipse PTP Parallel debugger” on page 138.

Profile configuration

To create, edit, and delete profile configurations:

- Select **Run** → **Profile Configurations**. The Profile Configurations window is displayed, shown in Figure 5-26 on page 85. In this window, you have two separate areas. The left side is where you manage your configurations, and the right side is where you actually edit the configurations for the selected profile configuration.

Figure 5-24 on page 83 illustrates the buttons you can use to create, duplicate (for already created configurations) and delete configurations, respectively from left to right.

In the Profile Configurations window (Figure 5-26 on page 85), you have the following predefined configurations from where you can create customized configurations:

- C/C++ application
 - Launch group
 - Parallel application
- To create an example of a Parallel Application, select it, and click **New launch configuration** (first button from Figure 5-24 on page 83). A new subentry label of the Parallel Application label appears. You can name it and customize it with the following options that are available on the right side of the Profile Configurations window:
 - Resources
 - Application
 - Arguments
 - Environment

- Performance analysis
 - Parametric study
 - Common
 - HPC Toolkit
3. The specific configurations for this mode are the Performance Analysis, Parametric Study and the HPC Toolkit tabs, shown in Figure 5-26. Inside these tabs we can edit the tool to use for performance analysis (currently only HPC Toolkit is available), parametric tests options, and the HPC Toolkit specific options for instrumentation:
 - Data collection
 - Performance analysis using HPM (Hardware Performance Monitoring)
 - Profiling and trace MPI calls
 - MIO (I/O analysis and optimization)
 4. To save the changes, click **Apply**, or just press **Profile** to save them and run the profile configuration.

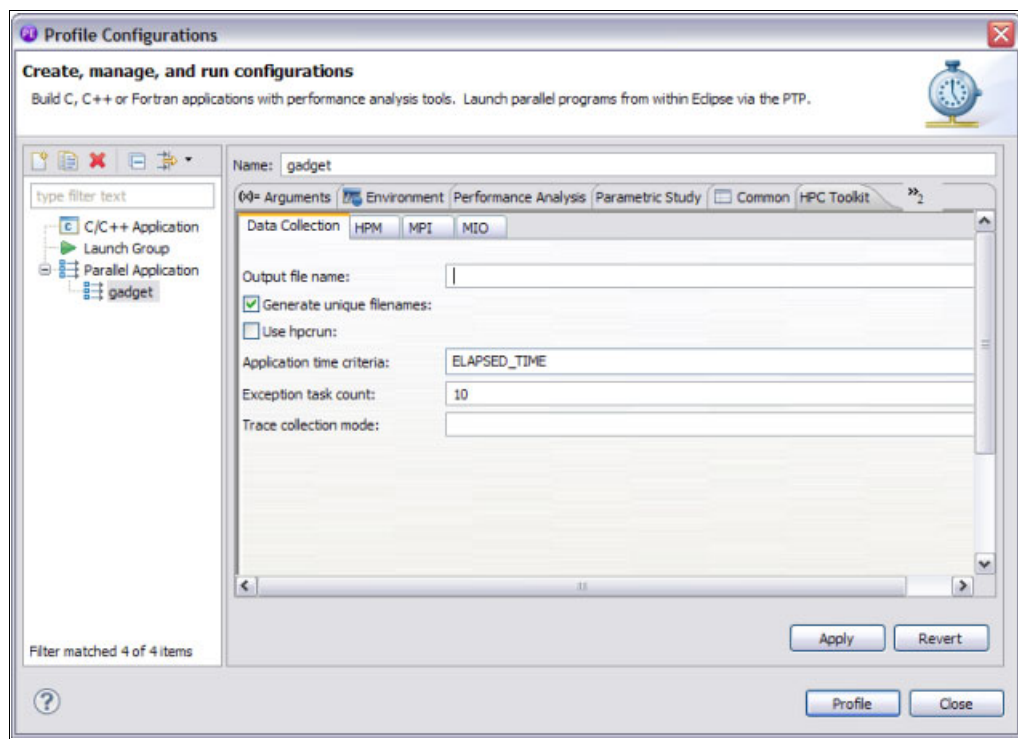


Figure 5-26 Profile configurations (HPC Toolkit tab)

5.5 Edit features of Eclipse

For the current section, we gathered a group of features, which Eclipse can help enhance your code development of C, C++, or Fortran parallel applications.

The C/C++ editor

Eclipse CDT (C/C++ Development Toolkit) provides a full-featured and rich editor for C and C++ languages. In the C/C++ perspective, you cannot perform common tasks, such as the following ones but also the more advanced tasks, which are described in Table 5-2 on page 86:

- Code with syntax highlights for C, C++ and makefile

- Code navigation
- Outline C/C++ syntax elements
- View user-defined make targets
- Outline makefile syntax elements
- Source code refactoring and transformation

Table 5-2 Eclipse edit features for C/C++

Feature	Description
Jump into function declaration	Can right-click and jump into the function code.
Code template	Provides hundreds of code templates to make coding faster and easier. Moreover, there are templates for Fortran language in its diverse variants and code statements for OpenACC, OpenMP, and MPI frameworks. Code templates can be inserted into application code by pressing Ctrl+Space and then selecting a desired template, as shown in Figure 5-30 on page 89.
Content assist	As you are typing, it displays a list of statements, such as variables, subroutines, intrinsics, and code templates, based in the scope and alphabetical in what you typed. The content assist is also triggered by pressing Ctrl+Space. See Figure 5-31 on page 89 as an example of content assist.
Show type hierarchy	Shows a tree display of a type or method. The operation is executed when you select a type or method, right-click it, and select Open Type Hierarchy . As a result, it shows a type hierarchy view, as shown in Figure 5-27 on page 87.
Show call hierarchy	Shows a tree display of function call hierarchy. The operation is executed when you select function call or declaration, right-click it, and select Open Call Hierarchy . As a result, it shows a call hierarchy view, as shown in Figure 5-28 on page 87.
Include browser	Shows a tree display with a relationship between the source and header files. The view is opened by right-clicking an open file in the editor and then selecting Show In → Include Browser . As a result, it opens a view, as shown in Figure 5-29 on page 88.

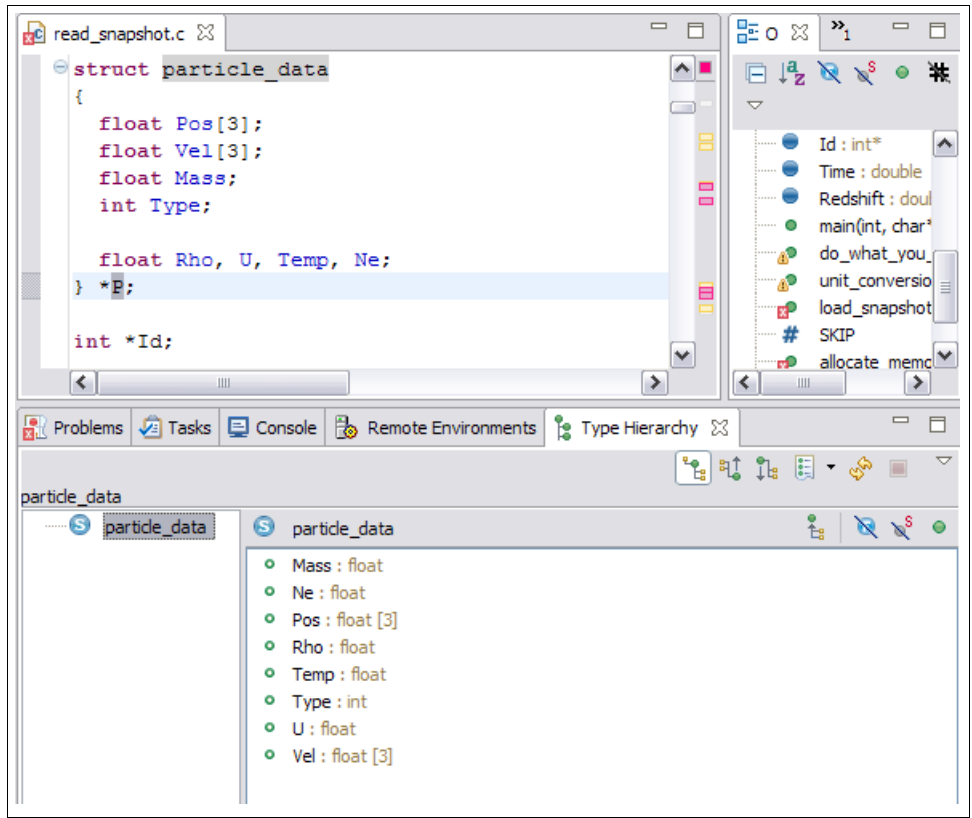


Figure 5-27 C/C++ editor type hierarchy

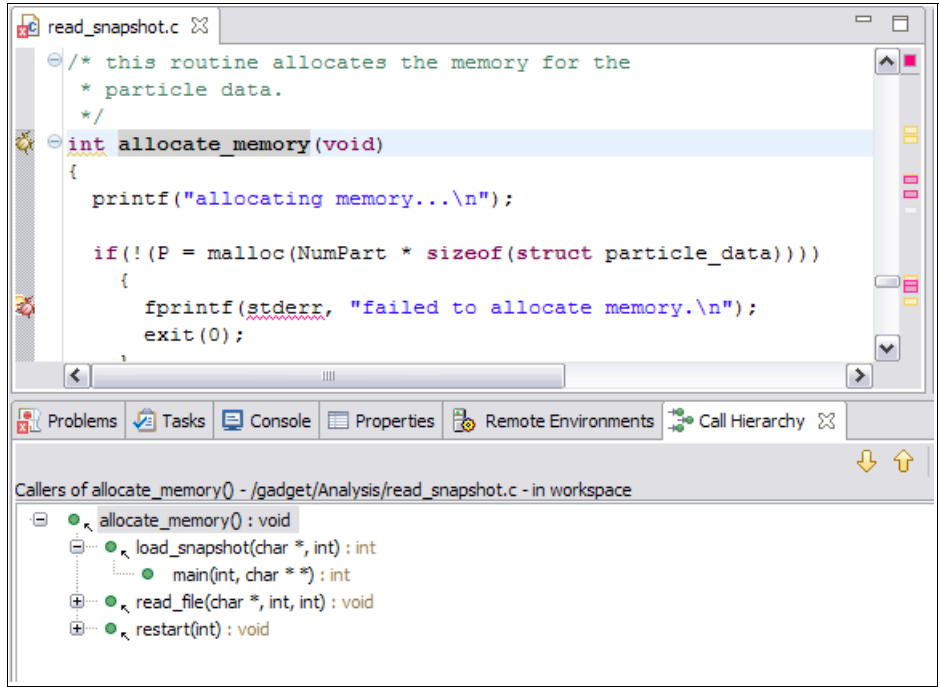


Figure 5-28 C/C++ editor call hierarchy view

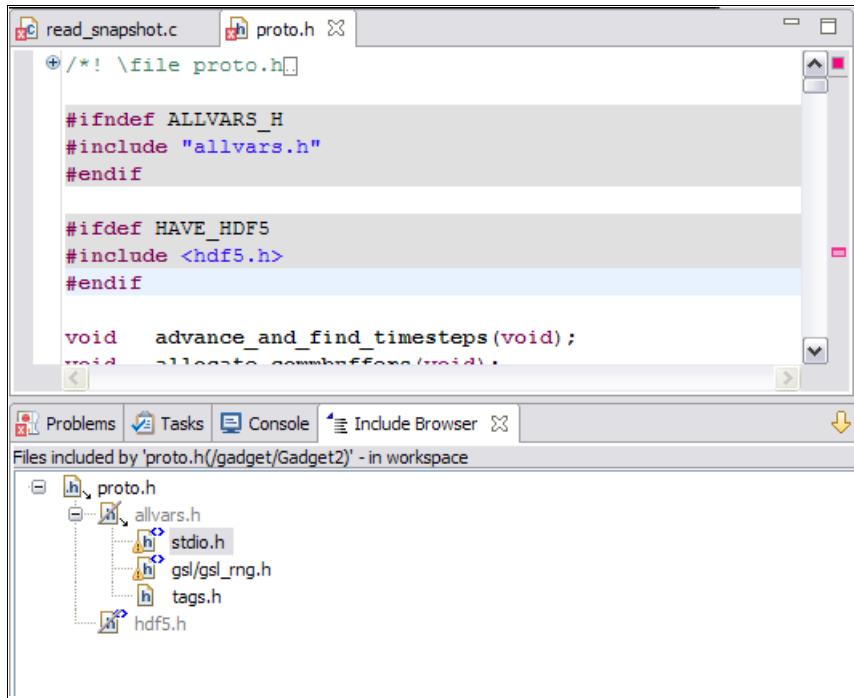


Figure 5-29 C/C++ include browser

The Fortran editor

Eclipse Photran implements a full-featured and rich editor but rather specific for the Fortran language, which are discussed in Table 5-3. Likewise the CDT editor provides the following features:

- ▶ Code navigation
- ▶ Syntax-highlighting editor
- ▶ Outline view
- ▶ Makefile-based compilation along with make target view

Table 5-3 Eclipse edit features for Fortran

Feature	Description
Code template	It provides hundreds of code templates to make coding faster and easier. Moreover, there are templates for Fortran language in its diverse variants and code statements for OpenACC, OpenMP, and MPI frameworks. Code templates can be inserted into application code by pressing Ctrl+Space and then selecting a desired template, as shown in Figure 5-30 on page 89.
Content assist	As you type, it displays a list of statements, such as variables, subroutines, intrinsics, and code templates based on the scope and alphabets that you typed. The content assist is also triggered by pressing Ctrl+Space. See Figure 5-31 on page 89 as an example of content assist.
Declaration view	Provides a view tab that displays detailed information about any selected (positioning cursor in) program, variable, subprogram, or subroutine statement. See the bottom view tab in Figure 5-32 on page 90 as an example.

Feature	Description
Hover Tip	Fortran hover tips display the same information as the declaration view but in an embedded window just below the selected statement. See the hover tip in Figure 5-32 on page 90 as an example.
Fortran language-based search	Provides search mechanics focused in Fortran syntax, allowing specific queries in statements (variable, function, subroutine, module, and so on.). The fortran search window (see Figure 5-33 on page 90) can be opened in the menu Search → Fortran .

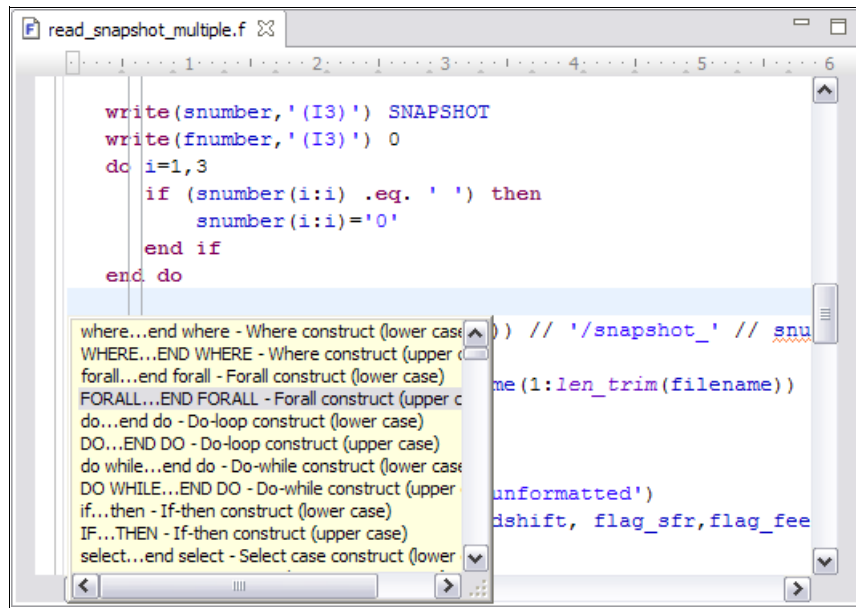


Figure 5-30 Fortran editor code templates

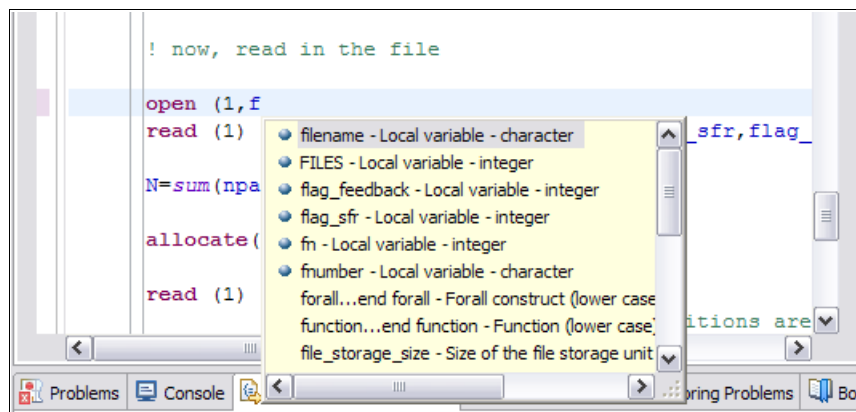


Figure 5-31 Fortran editor content assist

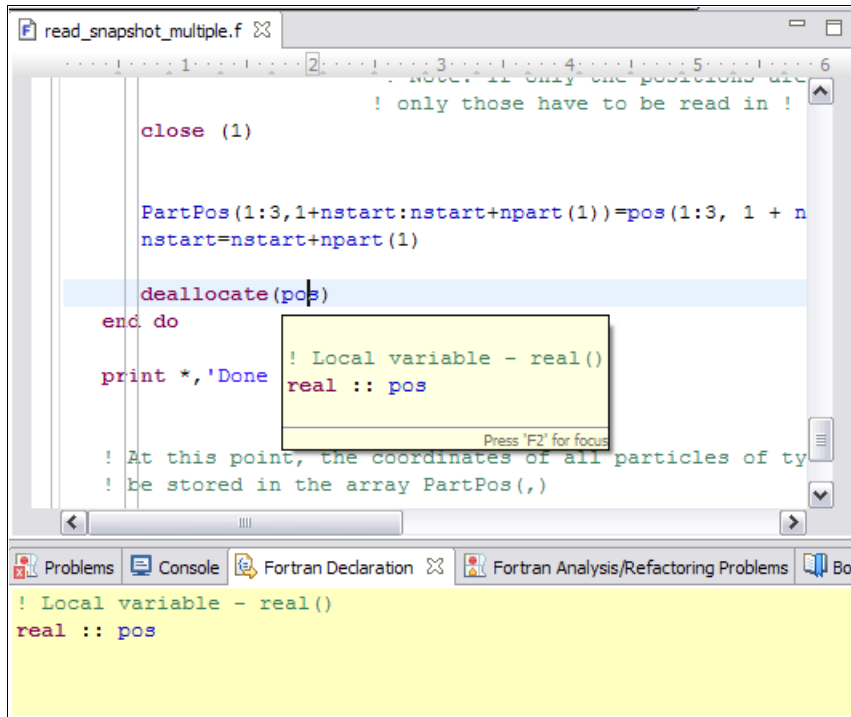


Figure 5-32 Fortran declaration view and hover tip

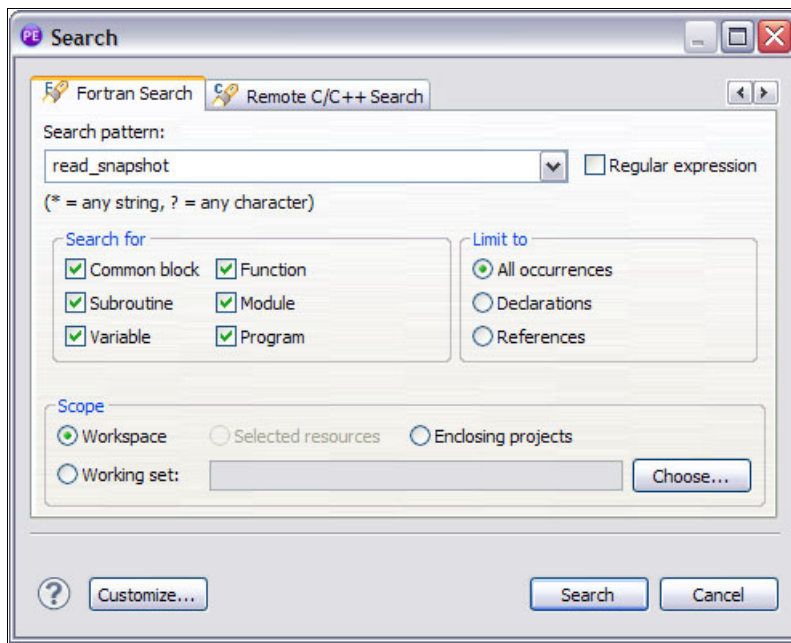


Figure 5-33 Fortran search window

Notice that most of the Fortran editor features described in Table 5-3 on page 88 rely on source code analysis that must be enabled using the menu **Project** → **properties** → **Fortran general** → **Analysis/refactoring** and then setting the options for each feature, as shown in Figure 5-34 on page 91.

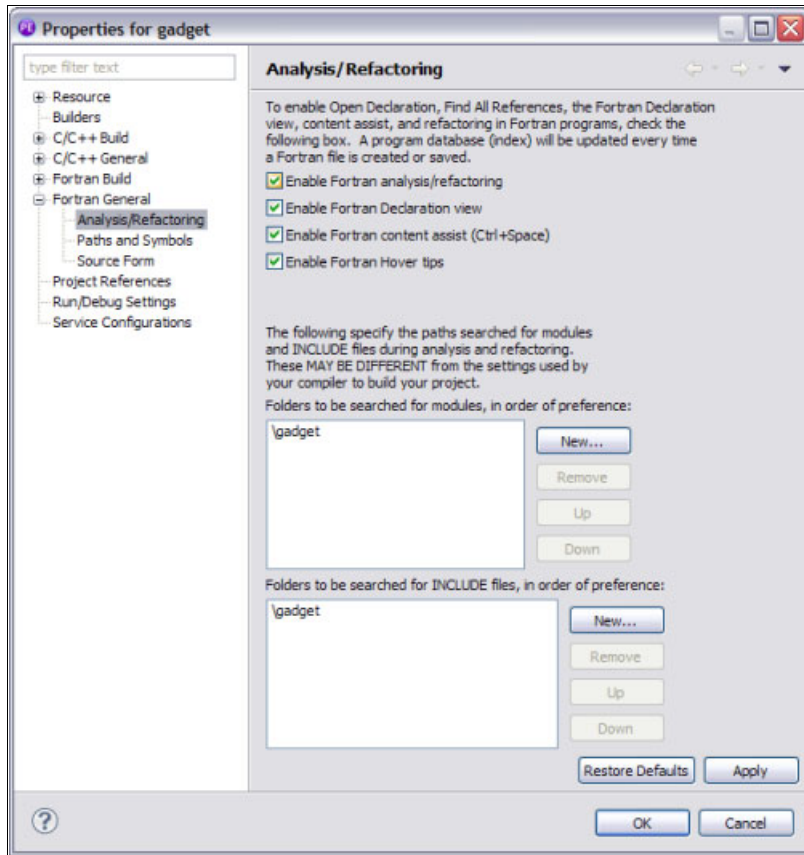


Figure 5-34 Enable Fortran analysis

The Eclipse PTP also provides tools for static analysis of parallel applications source code that is discussed in “Parallel Static Analysis” on page 133.

5.6 Debugging using Eclipse

The Eclipse PTP comes with a parallel debugger that provides parallel applications some specific debugging features that distinguish it from the Eclipse debugger for serial applications. In particular, it was designed to treat parallel application as a set of processes; therefore, providing different visualization modes, a special type of breakpoint called Parallel Breakpoint, multiple current instruction pointers, and other features that ease the debugging of parallel applications.

Refer to “Eclipse PTP Parallel debugger” on page 138 for further details about PTP parallel debugger features along with information about how it integrates within IBM Parallel Environment (PE) Developer Edition.

5.7 Integrating external applications

In this section, we present the option to incorporate an external tool or call program into Eclipse. These programs can either be configured to run after building a project or for all projects of the same selected workspace.

To create, edit, and delete external program configurations:

1. Select **Run** → **External Tools** → **External Tools Configurations**. The External Tools Configurations window is displayed, as shown in Figure 5-35. In this window, you can create, duplicate (for already created configurations), and delete configurations, respectively from left to right, as shown also in Figure 5-24 on page 83.

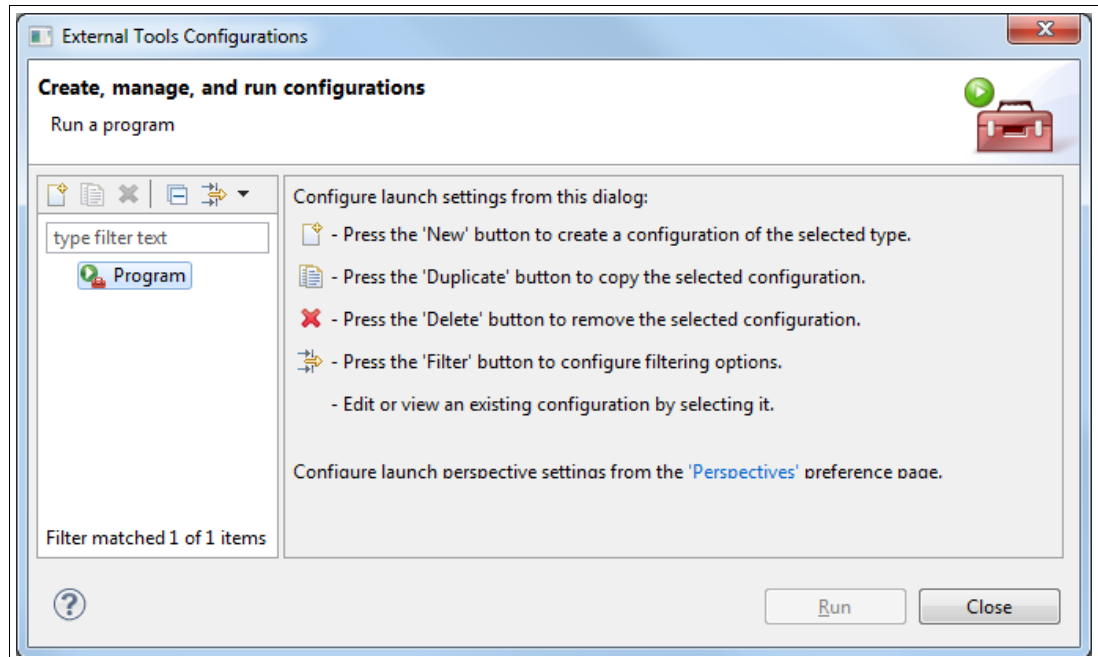


Figure 5-35 External tools configurations dialog

2. In the External Tools Configurations window (Figure 5-35), you have a predefined configuration called Program from where you can create customized configurations.

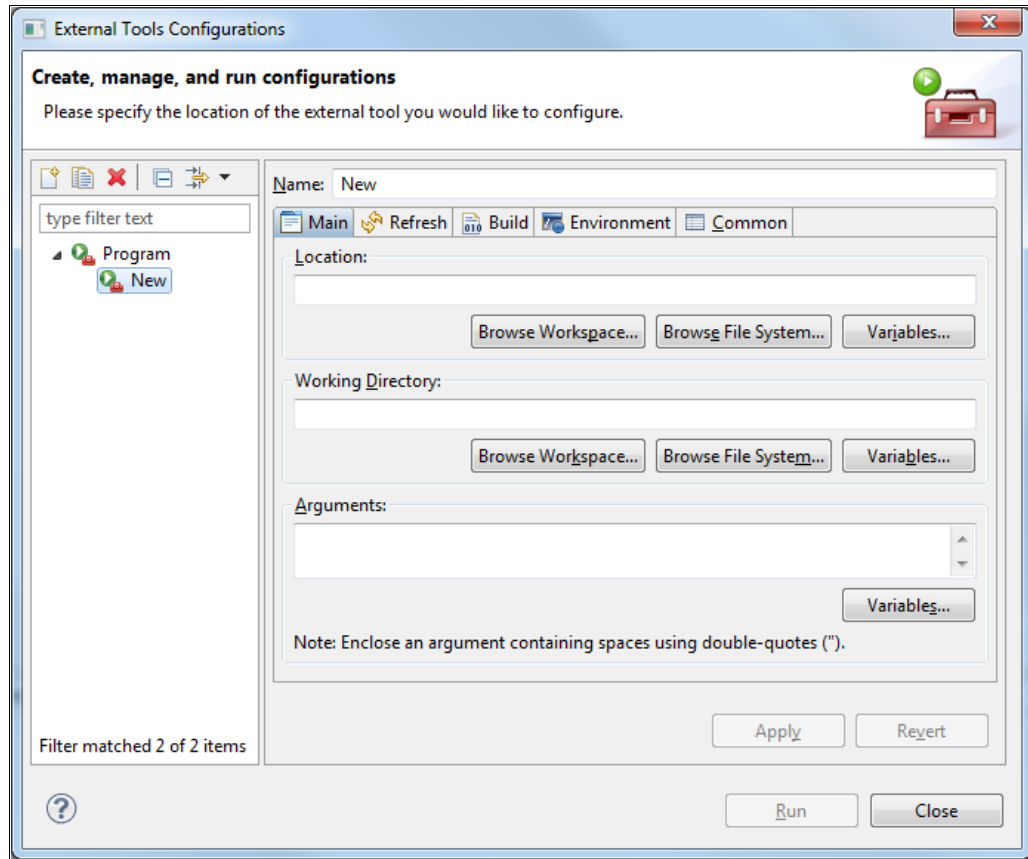


Figure 5-36 Creating a program configuration inside the external tools configurations

3. To create an example of a Program configuration, select it, and click **New Launch Configuration** (first button from Figure 5-24 on page 83). A new subentry label of the Program label appears. After creating it, a second area is distinguished to the right side of the External Tools Configuration window, where you actually give it a name and customize it within the following tabs:
 - Main
 - Refresh
 - Build
 - Environment
 - Common
4. The specific configurations for this mode are the Main, Refresh and the Build tabs (Figure 5-36). Inside these tabs, we can edit which program to use where and for what projects to run it:
 - At the Main tab, you specify the program to run, where it runs, and its arguments (Figure 5-36).
 - Inside the Refresh tab, you can specify what you want to refresh in the Eclipse workspace upon completion of the program you specified (see Figure 5-37 on page 94).
 - Under the Build tab, you can choose when to run the program. The available options are the possibility to run it after or before building a specific project or the entire workspace (Figure 5-38 on page 94).

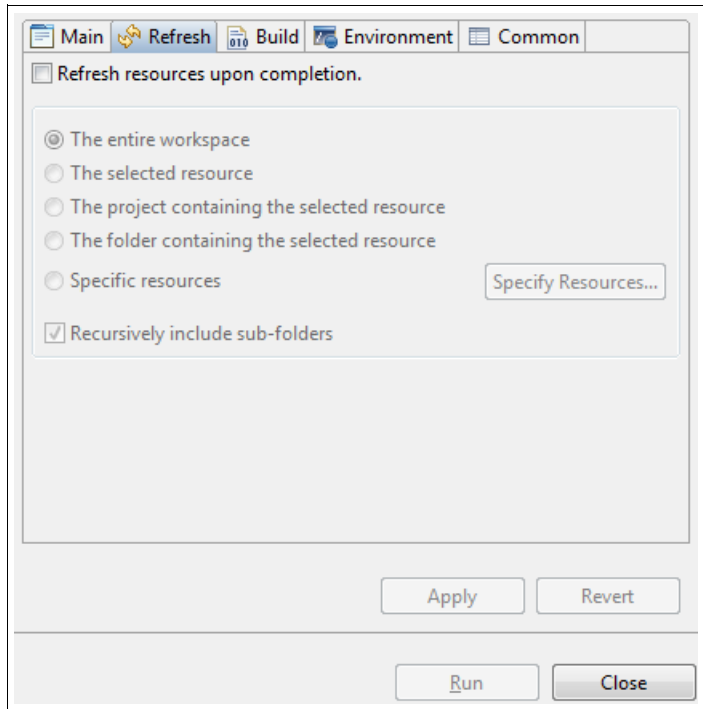


Figure 5-37 Refreshing the configuration tab

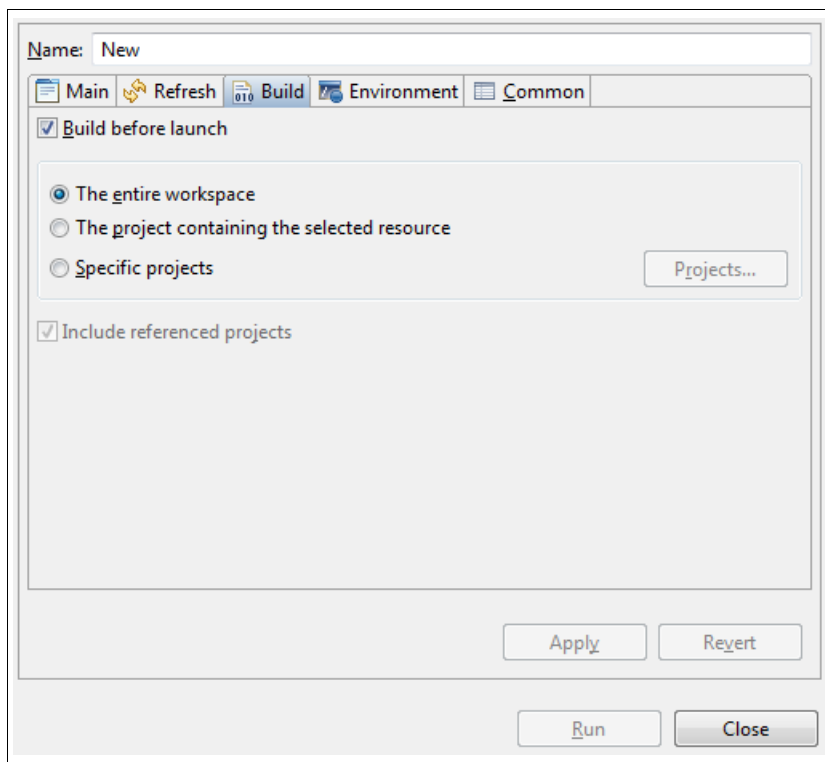


Figure 5-38 Build configuration tab

5. To save the changes, click **Apply**, or just press **Run** to save them and run the external tool configuration.



Parallel Environment Developer Edition tools

This chapter describes the use of the IBM Parallel Environment (PE) Developer Edition tools that are available to assist you with several kinds of performance analysis, tuning, debugging, and for solving issues in parallel applications.

These tools are mostly integrated within the Eclipse IDE, designed to be easily executed and still grant flexibility. They provide assistance in finding hot spots in source code, performance bottlenecks, and also are helpful in finding malfunctions and defects in parallel applications.

This chapter provides information about:

- ▶ Tuning tools
- ▶ Debugging

6.1 Tuning tools

The IBM HPC Toolkit provides profiling and trace tools integrated within the Eclipse UI and that are designed to gather rich data regarding the parallel application behavior in execution time. Therefore, it is recommended to use these tools to obtain initial performance measurements for your application, find hotspots, and also bottlenecks. The tools are:

- ▶ HPM tool for performance analysis based on hardware event counters
- ▶ MPI profiling and trace tool for performance analysis based on MPI events
- ▶ OpenMP profiling and trace tool for performance analysis of openMP primitives
- ▶ I/O profiling tool for performance analysis of parallel application I/O activities

There is the non eclipse-integrated Xprof tool, which is distributed within PEDE and relies on call graph base analysis for performance analysis. Its use is actually recommended first in the tuning process because it gets an overview of performance problems and is also suitable for identifying functions wasting most of the application execution time.

Our suggestion is to start identifying hotspots with Xprof and then narrow down the problem using other tools.

The basic workflow to use the Eclipse-integrated tools is illustrated in Figure 6-1 on page 97, where you start by preparing the application for profiling. After that, you must make a profile launch configuration according to your needs and then you must run the application to obtain data. Finally you get to visualize results in many possible ways with information presented at several levels of detail allowing operations, such as zooming in/out and easy browsing through the data collected.

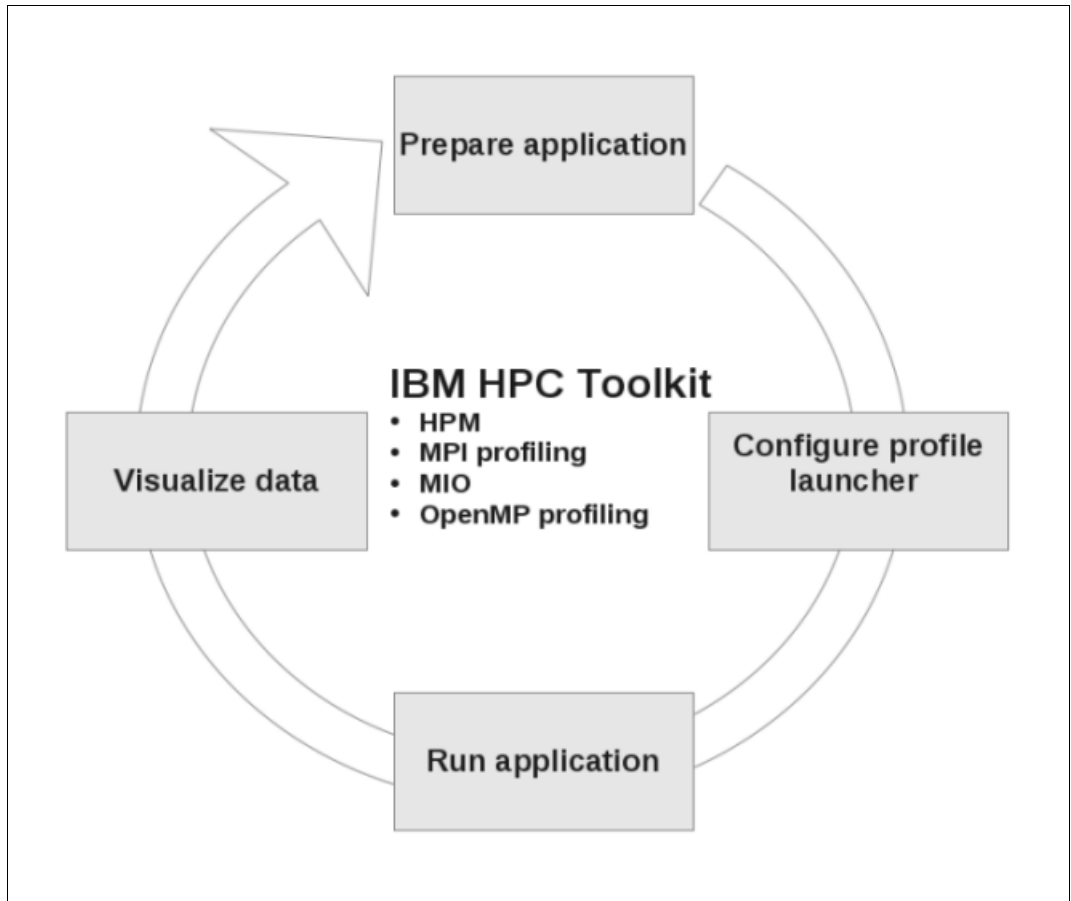


Figure 6-1 IBM HPC Toolkit basic workflow

All data is produced and collected in application runtime through the use of some technologies that require your application’s executable be instrumented. Indeed, the IBM HPC Toolkit instrumentation mechanism enables you to focus on just a small portion of code to avoid common problems on application analysis, for instance, increased overhead and production of uninteresting data. The instrumentation mechanisms are explained in 6.1.1, “Preparing the application for profiling” on page 100.

An overview about how to create a profile launch configuration is shown in 6.1.2, “Creating a profile launch configuration” on page 104. This is an essential step in the workflow where you configure a tool for execution and the IBM HPC Toolkit allows you to determine the detail level and amount of performance data to be collected.

The tools support varies based on the target environment (operating system and hardware architecture) where an application is built to run in. Table 6-1 presents the tools support by environments.

Table 6-1 Tools supported by platform

Tool	Linux on x86	Linux on Power ^a	AIX on IBM Power ^a
Xprof	No	Yes	Yes
Hardware Performance Monitor	Yes ^b	Yes	Yes
I/O Profiler	Yes	Yes	Yes

Tool	Linux on x86	Linux on Power ^a	AIX on IBM Power ^a
MPI Profiler	Yes	Yes	Yes
OpenMP Profiler	No	Yes	Yes

- a. Currently supports IBM POWER6 and POWER7 processors and application built with the IBM XL Compiler only.
- b. Currently supports Intel x86 Nehalem, Sandy Bridge, and Westmere microarchitectures.

The IBM HPC Toolkit Eclipse perspective

The IBM HPC Toolkit plug-in comes with an eclipse perspective (“Eclipse terms and concepts” on page 5) that consolidates tooling operations conveniently into a single view to support the following tasks:

- ▶ Instrument binaries in preparation to run analysis
- ▶ Manage, browse and one-click visualize performance data
- ▶ Configure visualization modes

Figure 6-2 shows illustrated views common to all the tools in the perspective:

- ▶ The bottom-left Instrumentation view allows binary instrumentation for ease. It contains tabs to instrument your binaries by selecting options specific for each tools. It also holds a button to trigger instrumentation. More information about binary instrumentation is provided in “Preparing the application for profiling” on page 100.
- ▶ The bottom-right Performance Data tab view lists all generated performance data files of your project. It also allows you to browse data generated by different tools and a double-click in any file name is going to open its associated visualization into the Performance Data Summary tab view.

The bottom-right Performance Data Summary tab view is where the gathered data actually showed up in table report format.

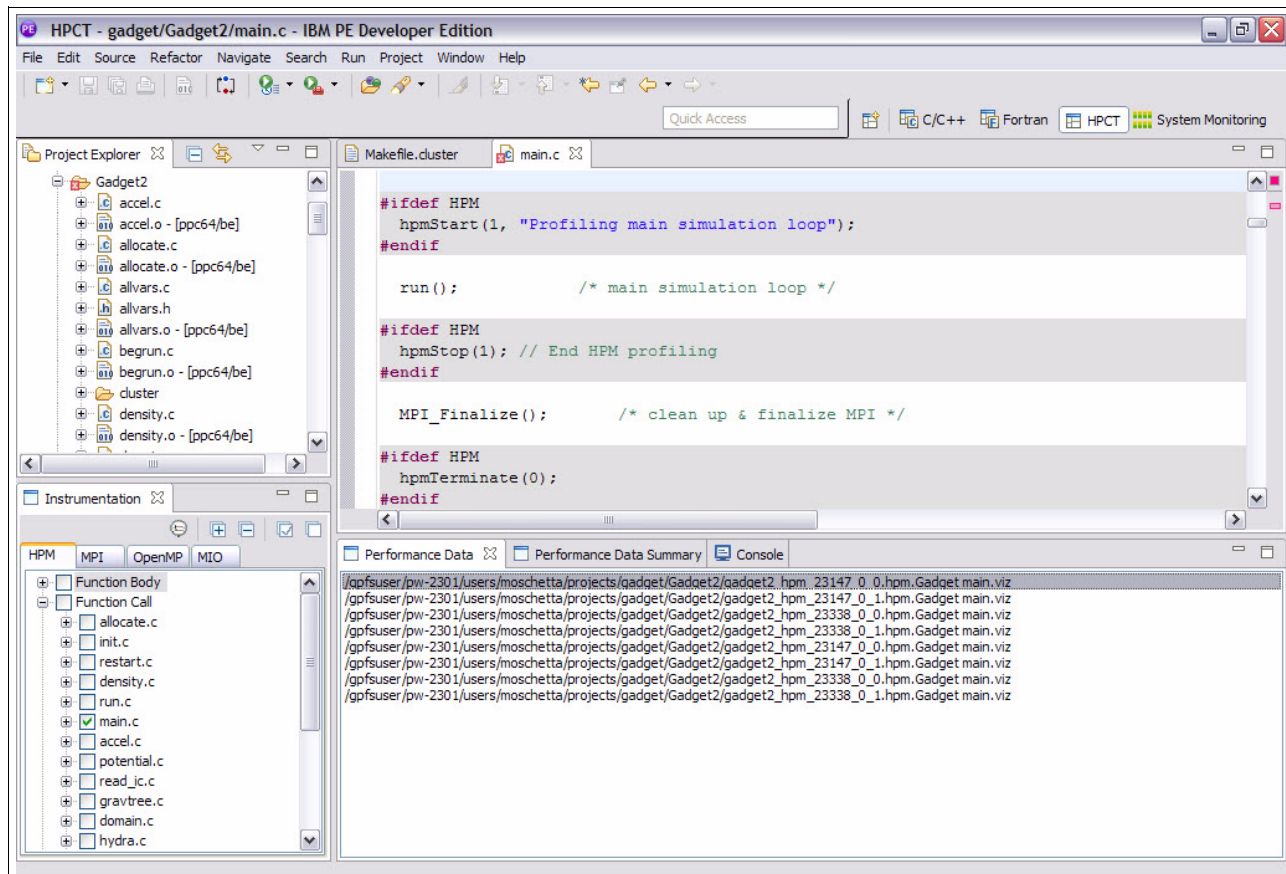


Figure 6-2 IBM HPC Toolkit Eclipse perspective

There are other views specific to some tools because of the different visualization required and so will appear eventually. They will be detailed later in this chapter, thus following are just their names for reference:

- ▶ Metric browser
- ▶ MIO detail
- ▶ MIO summary
- ▶ MIO trace
- ▶ MPI trace
- ▶ Performance data detail

The perspective can be opened in several ways, for instance:

- ▶ On top menu bar, click **Window** → **Open perspective** → **Other**. Select **HPCT** from the Open perspective window and then click **OK**.
- ▶ Click Open Perspective (Figure 6-3) on the toolbar and then select **HPCT** (Figure 6-4 on page 100).

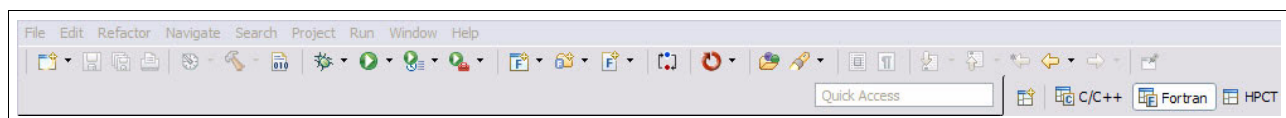


Figure 6-3 Open perspective button

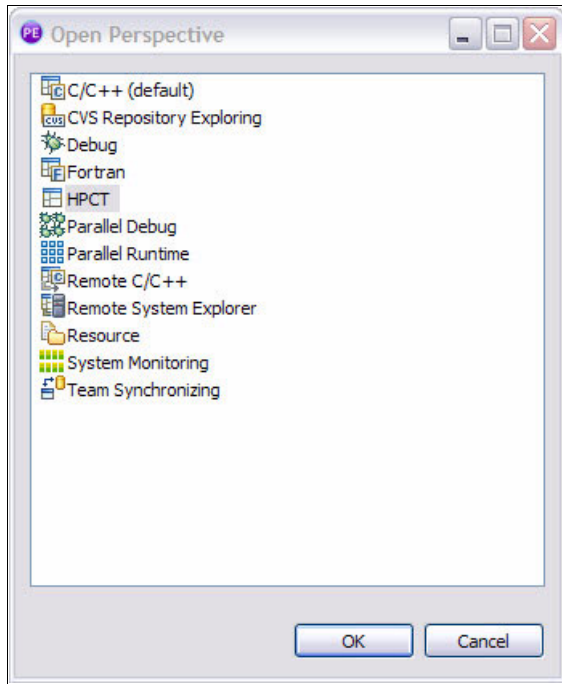


Figure 6-4 Open perspective window

6.1.1 Preparing the application for profiling

There are two requirements a parallel application must match to be analyzed by IBM HPC Toolkit tools:

- ▶ The parallel application executable must be instrumented by changing either its source code or binary file so that the IBM HPC Toolkit can get performance data.
- ▶ The parallel application must be built with the `-g` flag. Also, if chose binary instrumentation in a Linux on IBM Power system then it requires also `-Wl,--hash-style=sysv -emit-stub-syms` flags, as shown in the excerpt of Makefile in Example 6-1.

Example 6-1 Using the IBM HPC Toolkit required build flags in the Makefile

```
# HPC Toolkit required flags
HPCT_OPTS = -g -Wl,--hash-style=sysv -emit-stub-syms

LIBS = $(HDF5LIB) -g $(MPICHLIB) $(GSL_LIBS) -lgs1 -lgs1cblas -lm $(FFTW_LIB)

$(EXEC): $(OBJS)
        $(CC) $(OBJS) $(LIBS) $(HPCT_OPTS) -o $(EXEC)

$(OBJS): $(INCL)

clean:
        rm -f $(OBJS) $(EXEC)
```

Instrumenting the application

The toolkit is flexible enough to allow you to instrument a whole program, but also just the smallest areas of it where you might be interested in analyzing performance with the

advantages of giving you control over the areas of your application that you want to analyze (zoom in/out) and, consequently, the amount of data gathered.

The toolkit provides you with two modes¹ of instrumentation:

- ▶ Code instrumentation: Application code must be rewritten and recompiled with calls to the toolkit instrumentation library.
- ▶ Binary instrumentation^{2 3}: Application executable is rewritten by the toolkit with the instrumentation specified by you.

Code instrumentation

In the code instrumentation model, you must insert certain API calls into your application code so that you specify start/stop of profiling and regions performance data that must be collected. The IBM HPC Toolkit provides different runtime libraries, API, and linkage procedures for each of the performance tools (Table 6-2) so more details about the usage of code instrumentation is given in the tools section.

Table 6-2 IBM HPC Toolkit runtime libraries and headers

Library	Description	C header ^a	Fortran header ^a
hpc pmapi ^b	Provides instrumentation and analysis for Hardware Performance Monitoring tool	libhpc.h	f_hpc.h f_hpc_i8.h
mpitrace	Provides analysis for the MPI Profile and Trace tool	mpt.h	mpt_f.h
hpctkio	Provides analysis for the IO Profile tool	hpcMIO_user.h	Not supported

a. Header files are located in /opt/ibmhpc/ppdev.hpct/include

b. Only for AIX Systems

Binary instrumentation

In the binary instrumentation model, you use an GUI tool to select regions of your application that will be instrumented. The instrumentation tool is in charge of providing you the options as well as rewrite the application executable within all needed instructions to gather data for an specific HPCT. Notice that such as modality of instrumentation is straightforward in most of the cases but as stated before it isn't supported in x86 Linux systems that in turn will require source code change.

Binary instrumentation can be accomplished in three steps:

1. Open the executable for instrumentation. Within the project opened in the Project Explorer view, right-click the binary and then select **HPCT** → **Open executable** (Figure 6-5 on page 102). The HPC Toolkit (HPCT) perspective is automatically opened (refer to “The IBM HPC Toolkit Eclipse perspective” on page 98).
2. Select one or more regions that you are interested in investigating performance and so must be instrumented. Figure 6-6 on page 103 shows an example of binary instrumentation in preparation to run the HPM tool.
3. Click **Instrument** to generate an instrumented version of the binary with filename <executable>.inst, as shown in Figure 6-7 on page 103.

¹ Do not mix different modes because they will affect each other. Any calls to instrumentation functions that you code in your application (code instrumentation) might interfere with the instrumentation calls that are inserted by the toolkit (binary instrumentation).

² Not supported in x86 Linux systems.

³ IBM HPC Toolkit binary instrumentation will operate reliably on executables with a text segment size of no more than 32 MB.

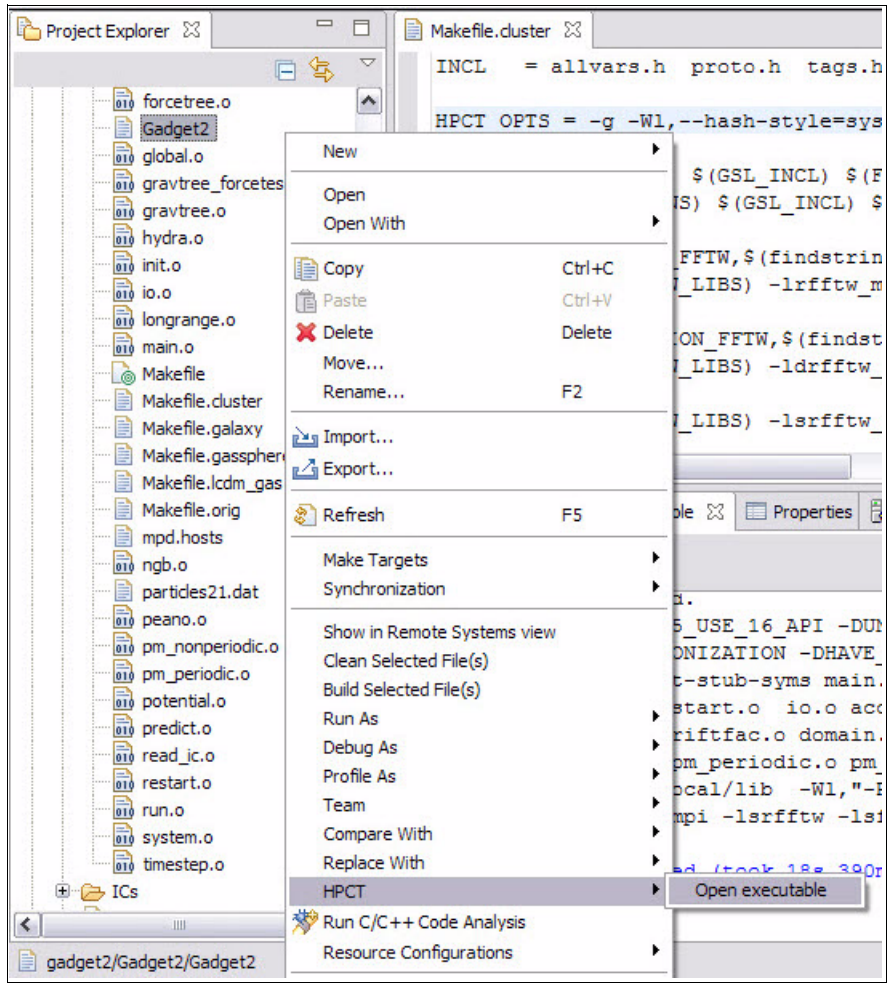


Figure 6-5 Opening executable for binary instrumentation

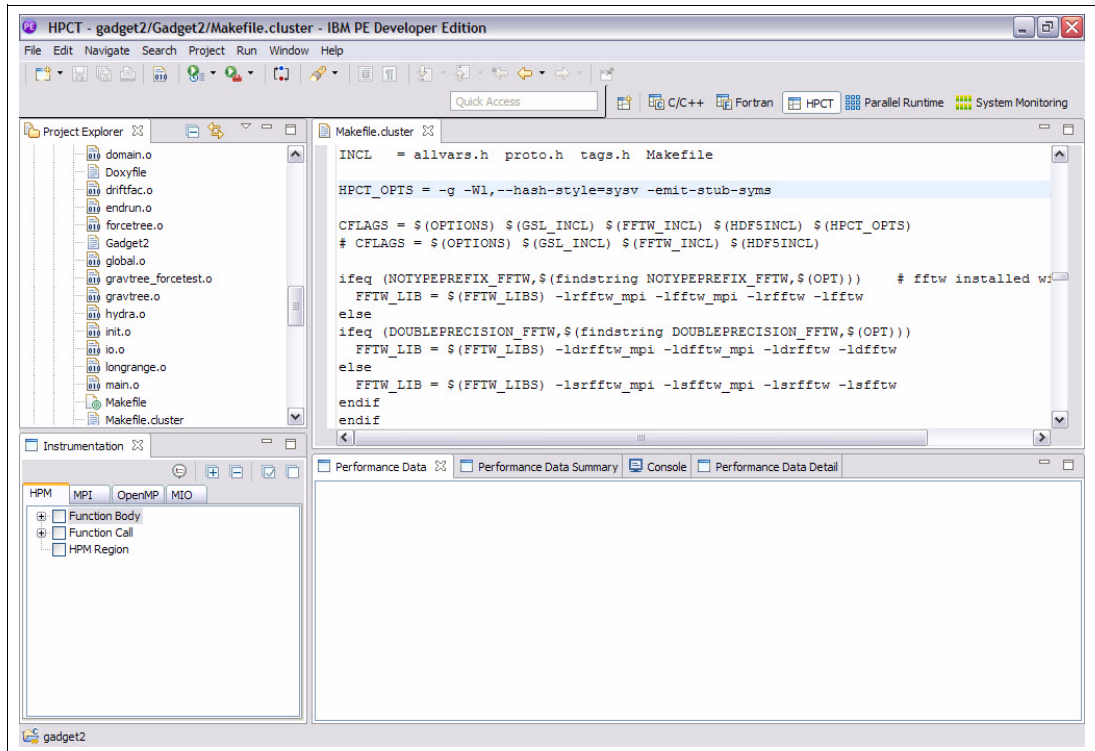


Figure 6-6 IBM HPC Toolkit perspective with Instrumentation view

In Figure 6-6, the bottom-left Instrument view has tabs for the tuning tools because each one allows different portions of the binary to be instrumented. So you must change the tab and choose options based on the tool that you want to generate an instrumented executable for analysis. In general, for the tools to work correctly, you must instrument at least one function or an entire source file of the parallel application to obtain performance measurements.

Important: Do not mix different tools in a single instrumentation because they might interfere with each other's analysis in an unpredictable way.

After you select the set of instrumentation that you want, you trigger the instrumentation tool by either pressing the instrumentation button in the view or right-clicking the selected node. Click **Instrument Executable**, as shown in Figure 6-7.

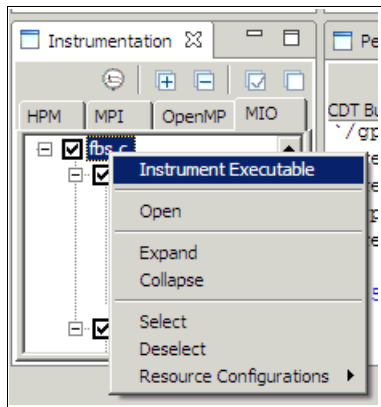


Figure 6-7 Running binary instrumentation tool

The message in Figure 6-8 shows instrumentation complete without errors.

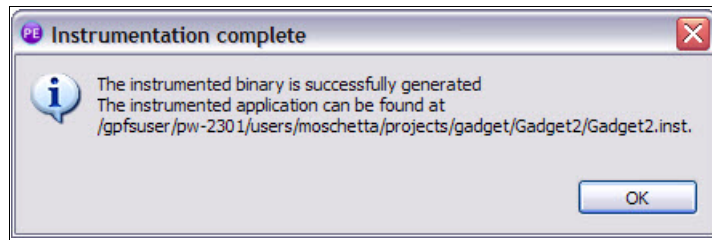


Figure 6-8 Message displayed when instrumentation complete

6.1.2 Creating a profile launch configuration

The IBM HPC Toolkit provides profiling and tracing tools that are useful for performance analysis as long as you properly create a profile launch configuration according to the tool that you want to use and the information you want to observe.

The IBM HPC Toolkit tools are executed by creating and invoking a profile configuration, where that profile configuration is created as a *parallel application* profile configuration, accessible by right-clicking the project folder and then selecting **Profile As** → **Profile Configurations**. It is going to open the Profile Configurations window where new profile launcher configurations are created under the parallel application section in the left box (Figure 6-9).

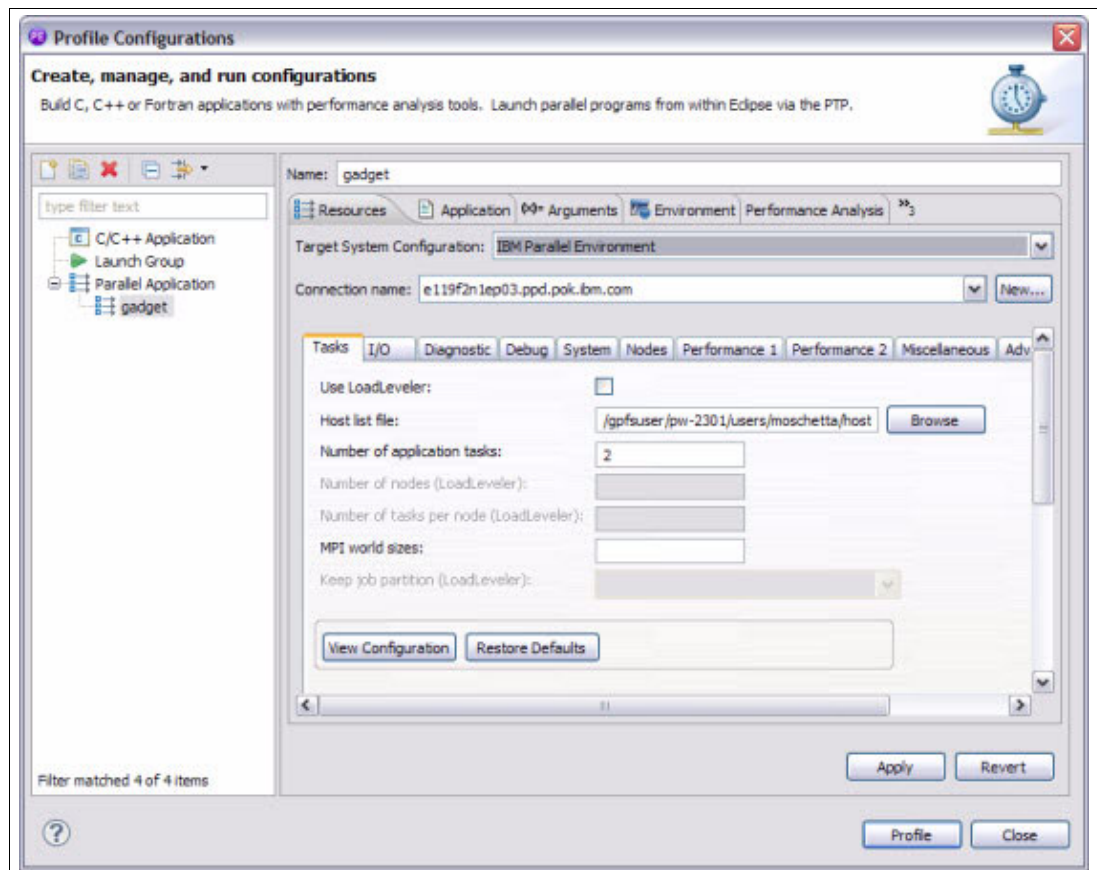


Figure 6-9 Profile configurations window

The parallel application configuration has Resources, Application, Arguments, and Environments tabs that must be fulfilled with information about how to run the parallel application. In particular, in the Application tab, it must be selected to run the instrumented executable (file named <executable>.inst by default) as illustrated in Figure 6-10.

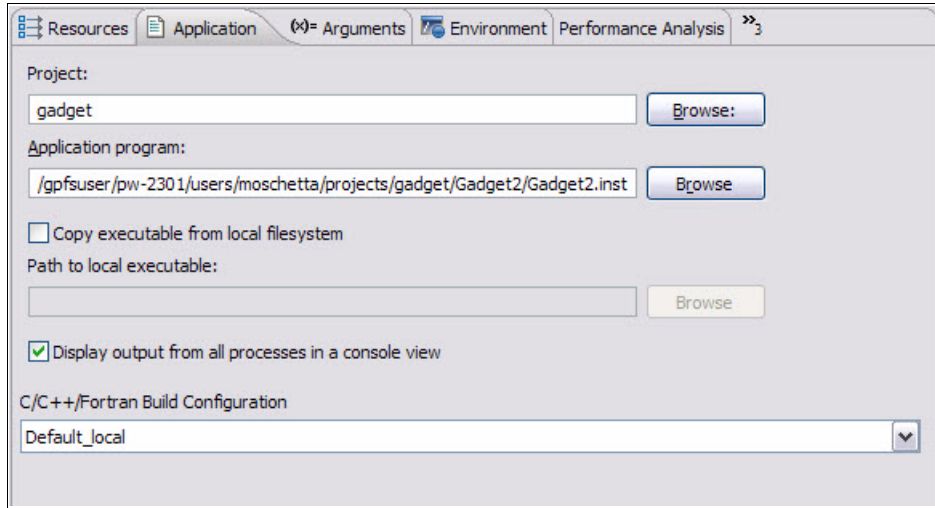


Figure 6-10 Profile configuration: set instrumented binary

There is also the HPC Toolkit tab that is omitted in Figure 6-10, but it is important to be properly filled because it is actually where you choose what performance tool is executed as well as placing information about how to control data gathering. Figure 6-11 illustrates how to open the HPC Toolkit tab.

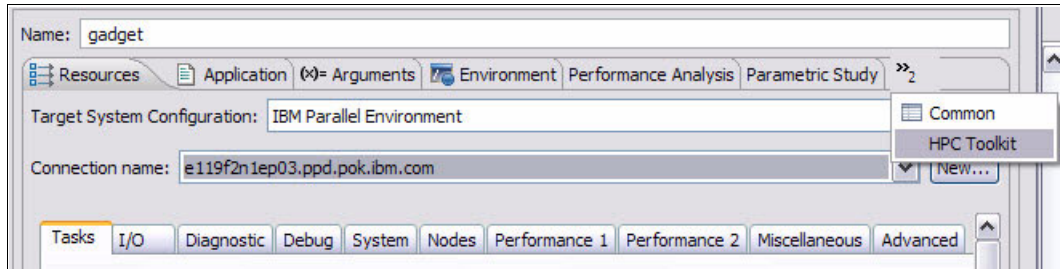


Figure 6-11 Opening HPC Toolkit tab

Figure 6-12 on page 106 shows the HPC Toolkit tab, which is composed of sub-tabs:

- Data collection** Contains fields which information is common for the tools.
- HPM** Contains specific fields to control the HPM tool (Refer to the “Hardware Performance Monitoring” on page 107).
- MPI** Contains specific fields to control the MPI tracing tool (Refer to the “MPI profiling and trace” on page 111).
- MIO** Contains specific fields to control the MPI tracing tool (Refer to Figure 6-22 on page 114).

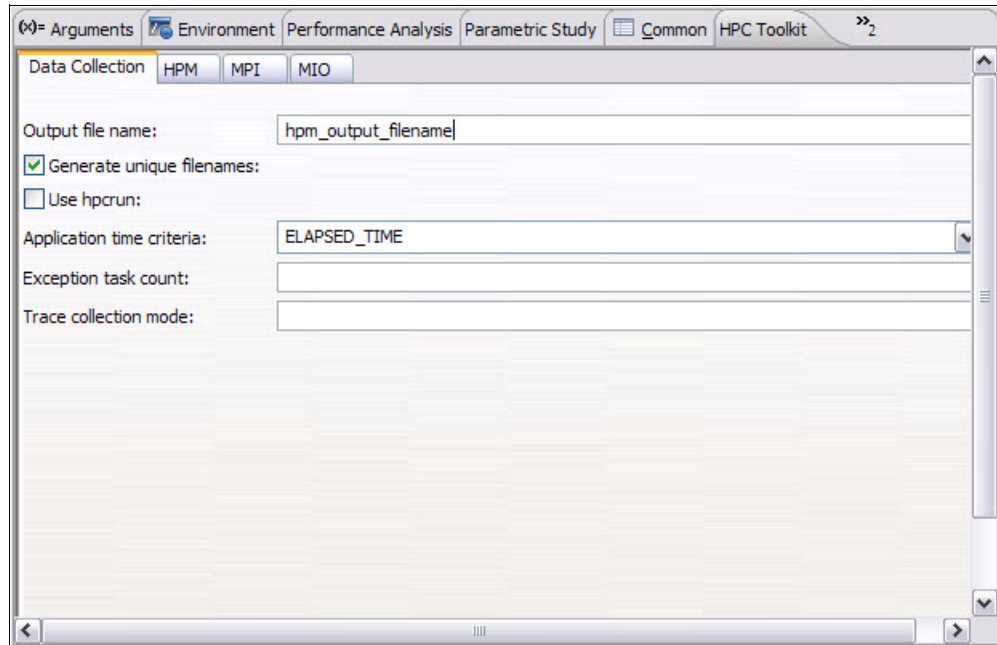


Figure 6-12 HPC Toolkit tab

The data collection tab (see Figure 6-12) is where you control the amount of data gathered in the process. Their fields must be carefully fulfilled, especially on large task applications where you really must limit the number of tasks that the tool generates data from, both to avoid file system performance impacts of generating thousands of files worth of data and from the impracticality of you managing and analyzing all of that data. The following list contains an explanation of each field:

- ▶ The Output File Name field value defines the base name for performance files generated by the tool, and the name will be <basename>_<world_id>_<world_rank>. Set it with a meaningful value for the particular tool you intend to run.
- ▶ The Generate Unique File names check box ensures that performance data files are separately generated by each MPI task. You must enable it if running an MPI application.
- ▶ The hpcrun check box allows you to change data collection behavior. If not enabled, the tool gets data for all tasks, except as limited by environment variables described for MPI profiling and the trace tool (refer to “MPI profiling and trace” on page 111). If enabled, you must set its nested fields:
 - Application time criteria specifies the metric the tool uses to decide what tasks to collect data from, either wall clock (ELAPSED_TIME) or CPU time (CPU_TIME).
 - Exception task count field limits the number of tasks the tool generates data from. You must specify the minimum and maximum number of data tasks that will be collected along with the average task and task 0.
 - Trace collection mode specifies how the tool uses system memory to collect data. There are two values accepted:
 - Memory is appropriate for applications generating small trace files which do not steal memory from the application's data space
 - Intermediate is appropriated for applications generating larger trace files

6.1.3 Hardware Performance Monitoring

The Hardware Performance Monitoring (HPM) tool leverages the hardware performance counters for performing low level analysis, which are quite helpful to identify and eliminate performance bottlenecks.

HPM allows you to obtain measurements on:

- ▶ Single hardware counter group of events
- ▶ Multiple hardware counter group of events
- ▶ Pre-defined metrics based on hardware counter group of events. Examples of derived metrics are:
 - Instructions per cycles
 - Branches mispredicted percentage
 - Percentage of load operations from L2 per cycle

Profiling your application

To profile your application:

1. Build the application using the required flags, as described in “Preparing the application for profiling” on page 100.
2. Instrument the parallel application in one of the following modes (see “Instrumenting the application” on page 100 for the basics on instrumentation):
 - a. Instrument source code by calling HPM runtime library functions. The application must be recompiled with some flags according to the run environment, as shown in Table 6-3. Refer to Table 6-4 for a quick reference to the runtime library API or consult the IBM HPC Toolkit manual at:

https://www.ibm.com/developerworks/wikis/download/attachments/91226643/hpct_guide_v5.1.0.pdf

Table 6-3 Build settings quick reference

	Compiler options	Linker options	Headers
Linux	-g -I/opt/ibmhpc/ppedev.hpct/include	-lhpc -L/opt/ibmhpc/ppedev.hpct/lib or -L/opt/ibmhpc/ppedev.hpct/lib64	libhpc.h f_hpc.h ^a or f_hpc_i8.h ^b
AIX	-g -I/opt/ibmhpc/ppedev.hpct/include	-lhpc -lpmpi -lpthreads ^c -L/opt/ibmhpc/ppedev.hpct/lib or -L/opt/ibmhpc/ppedev.hpct/lib64	libhpc.h f_hpc.h ^a or f_hpc_i8.h ^b

- a. Fortran applications
- b. Fortran applications compiled with -qintsize=8
- c. Optionally use xlc_r or xlf_r with IBM XL C/C++/Fortran compiler

Table 6-4 HPM library API quick reference

Description	C/C++	Fortran
Initialize the instrumentation library	hpmInit(id, progName)	f_hpmInit(id, progName)
Terminate the instrumentation library and generate the reports	hpmTerminate(id)	f_hpmTerminate(id)

Description	C/C++	Fortran
Identify the start of a section of code in which hardware performance counter events are counted	hpmStart(id, label)	f_hpmstart(id, label)
Identify the end of the section of code in which hardware performance counter events were counted	hpmStop(id)	f_hpmstop(id)

b. Instrument executable by leveraging the instrumentation tool that is going to produce a new binary renamed *<executableName>.inst*. Figure 6-13 shows the instrumentation tool allowing you to select any combination of three classes of instrumentation:

- Function call sites
- Entry and exit points of function
- User-defined region of code

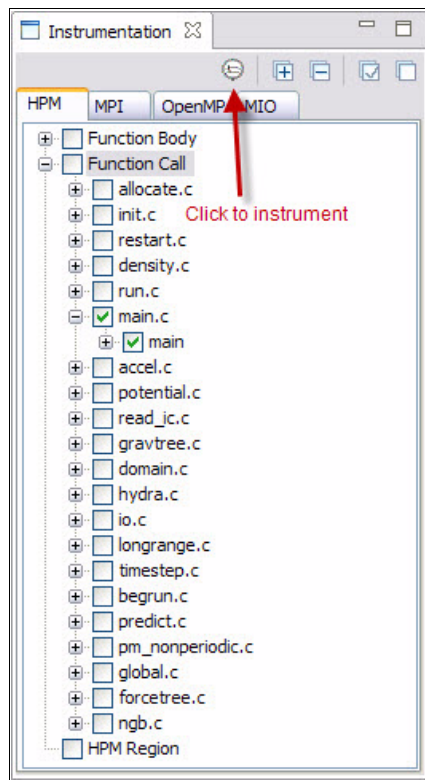


Figure 6-13 Instrumenting a binary for HPM profiling

3. Create an HPM launcher configuration where you must fulfill fields to control the data produced and gathered (refer to “Creating a profile launch configuration” on page 104). Figure 6-14 on page 109 shows the tool configuration screen that requires input of either a derived metric or counter group number.

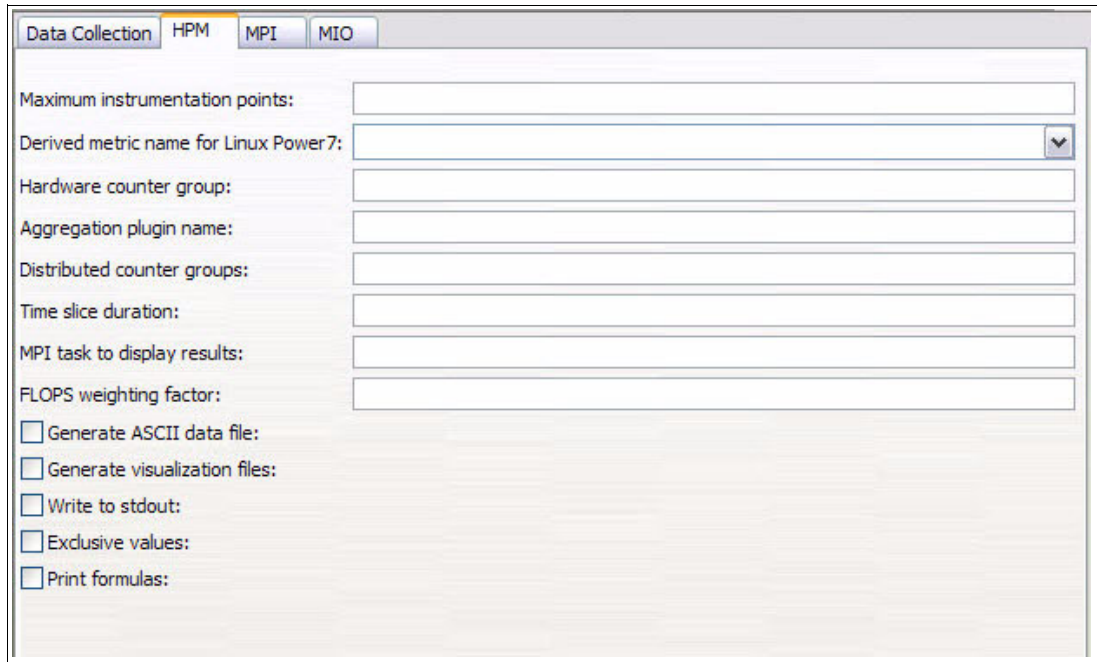


Figure 6-14 HPM configuration screen

The tool comes with existing derived metrics that are in most of the cases a good starting point for hardware performance analysis because they will result in higher-level information than just raw hardware events data. However, it still allows you to gradually pick events that show more hardware information in more low level hardware toward a performance bottleneck. Figure 6-15 lists every pre-built hardware performance metrics of Linux on IBM POWER7.

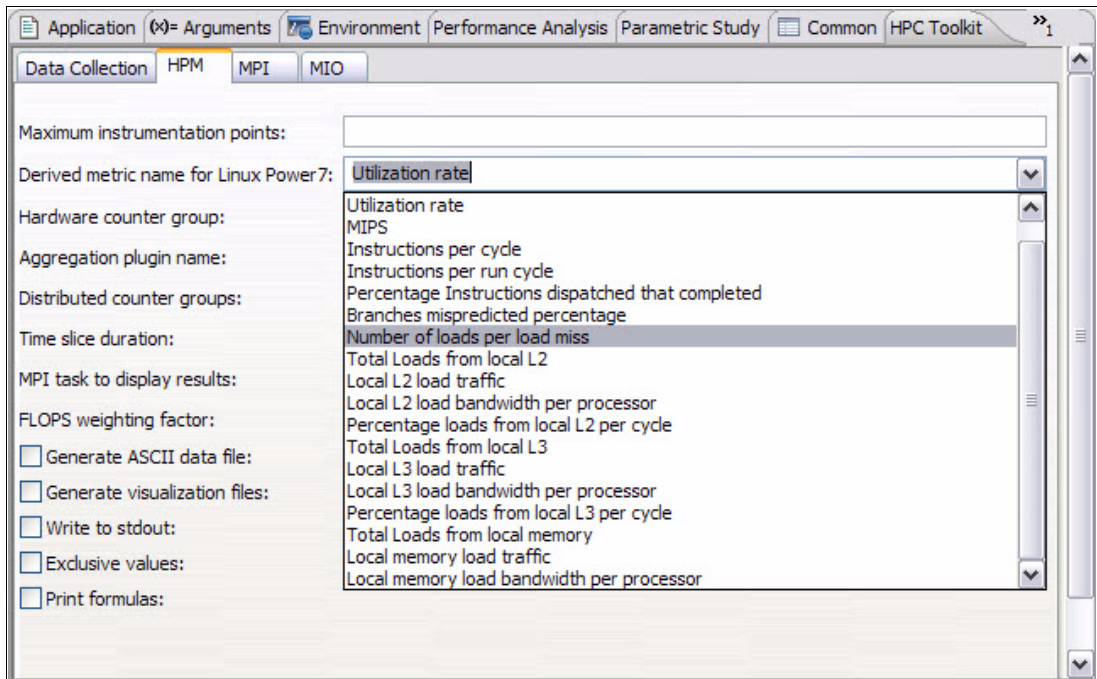


Figure 6-15 Derived hardware performance metrics for Linux on POWER7

However, you might want to use one of many counter groups available in your processor instead of the derived metrics. The listing of the counter groups must be obtained by manually connecting at the target system and executing the `hpccount` command, as shown in Example 6-2. Run `man hpccount` to open its manual and thus check out other useful options.

Example 6-2 Listing hardware performance groups

```
$ source /opt/ibmhpc/ppdev.hpct/env_sh
$ hpccount -l | less
```

Figure 6-16 shows the output of Example 6-2 executed in a POWER7 machine. Notice that the report shows the total of counter groups for the given processor and the complete listing of groups along with their associated hardware events.

```
Linux Power7: number of CPU groups supported: 255
Group 0:
PM_CYC ----- Cycles
PM_RUN_CYC ----- Run_cycles
PM_INST_DISP ----- # PPC Dispatched
PM_INST_CMPL ----- # PPC Instructions Finished
PM_RUN_INST_CMPL - Run_Instructions
PM_RUN_CYC ----- Run_cycles

Group 1:
PM_BR_PRED_CCACHE -- Count Cache Predictions
PM_BR_PRED_LSTACK -- Link Stack Predictions
PM_BR_MPRED_CCACHE - Branch Mispredict due to Count Cache prediction
PM_BR_MPRED_TA ----- Branch mispredict - target address
PM_RUN_INST_CMPL --- Run_Instructions
PM_RUN_CYC ----- Run_cycles

Group 2:
PM_BR_PRED ----- Branch Predictions made
PM_BR_PRED_CR ----- Branch predict - taken/not taken
PM_BR_PRED_CCACHE - Count Cache Predictions
PM_BR_PRED_LSTACK - Link Stack Predictions
PM_RUN_INST_CMPL -- Run_Instructions
PM_RUN_CYC ----- Run_cycles

Group 3:
PM_BRU_FIN ----- Branch Instruction Finished
PM_BR_TAKEN ----- Branch Taken
PM_BR_PRED ----- Branch Predictions made
PM_BR_MPRED ----- Number of Branch Mispredicts
PM_RUN_INST_CMPL - Run_Instructions
PM_RUN_CYC ----- Run_cycles
```

Figure 6-16 Example: output of command hpccount -l

Interpreting profile information

The HPCT perspective is opened as soon as profiling finishes. After which you are prompted to open the visualization files.

Figure 6-17 on page 111 shows an example of HPM results visualization for an application, which `main.c` function was instrumented (see bottom-left Instrumentation view). The tool collected hardware counter data, formatted it, and then wrote in visualization files (see listing in bottom-right Performance Data view).

All generated information regarding hardware performance data is displayed in the Performance Data Summary view, as shown in Figure 6-22 on page 114.

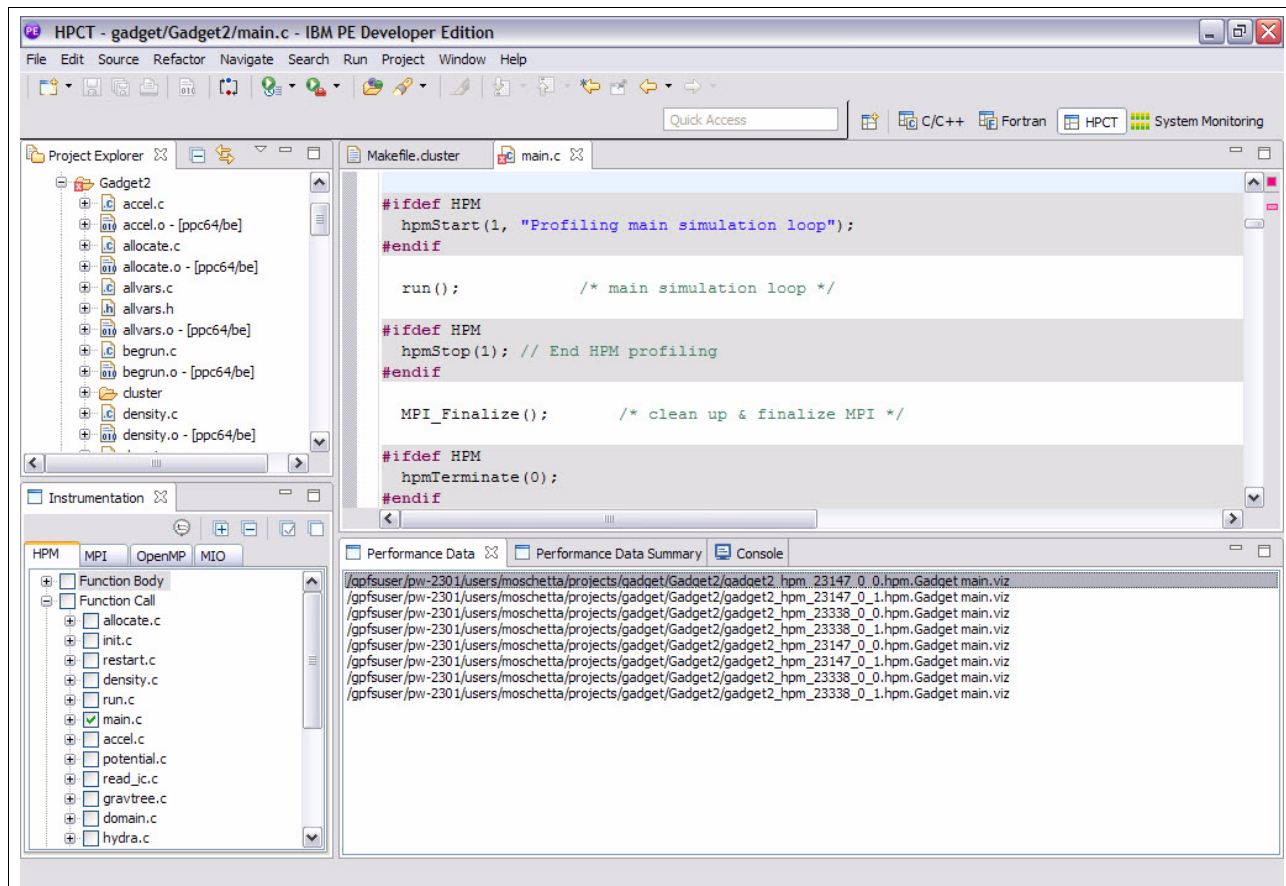


Figure 6-17 HPM performance data list

6.1.4 MPI profiling and trace

The MPI profiling and trace tool from IBM HPC Toolkit can gather data for all MPI routines and then generate profile reports and trace visualization of MPI primitives. However, currently the tool cannot create a trace for an application that issues MPI function calls from multiple threads in the application.

Instrumenting your application

To instrument your application, refer to the following steps:

1. To perform the profiling, either the binary can be instrumented by opening the binary directly using Eclipse or linking the source code with the libmpitrace.so for Linux and libmpitrace.a for AIX and then recompiling it. The application must be built using the required flags, as described in “Preparing the application for profiling” on page 100.

Currently the binary instrumenting without re-compilation approach is only supported on the POWER based architecture.

Figure 6-18 on page 112 illustrates how to link your source code to perform MPI profiling on a x86-based Linux machine. The source code mpi_1.c is utilized in this example as a Makefile project. Simply link the libmpitrace into your makefile.

```
mpib_164:  mpi_barrier_1.c
           mpcc $(CFLAGS) -m64 \
           -o ${OBJ}/mpi_barrier_64_1 -L/opt/ibmhpc/ppedev.hpct/lib64 -lmpitrace \
           mpi_barrier_1.c
```

Figure 6-18 Linking the libmpitrace

The mpi_1.c is linked to the mpitrace lib:

```
-L/opt/ibmhpc/ppedev.hpct/lib64 -lmpitrace
```

Notice that the linking is added to the end of this line; otherwise, the mpi tracing data might not be generated properly. Having done so, you might want to configure the profiling by clicking **Profile** and then choosing the profile configuration. On the left of the pop-up window, click **Parallel Application** to create a new profile configuration, as shown in Figure 6-19.

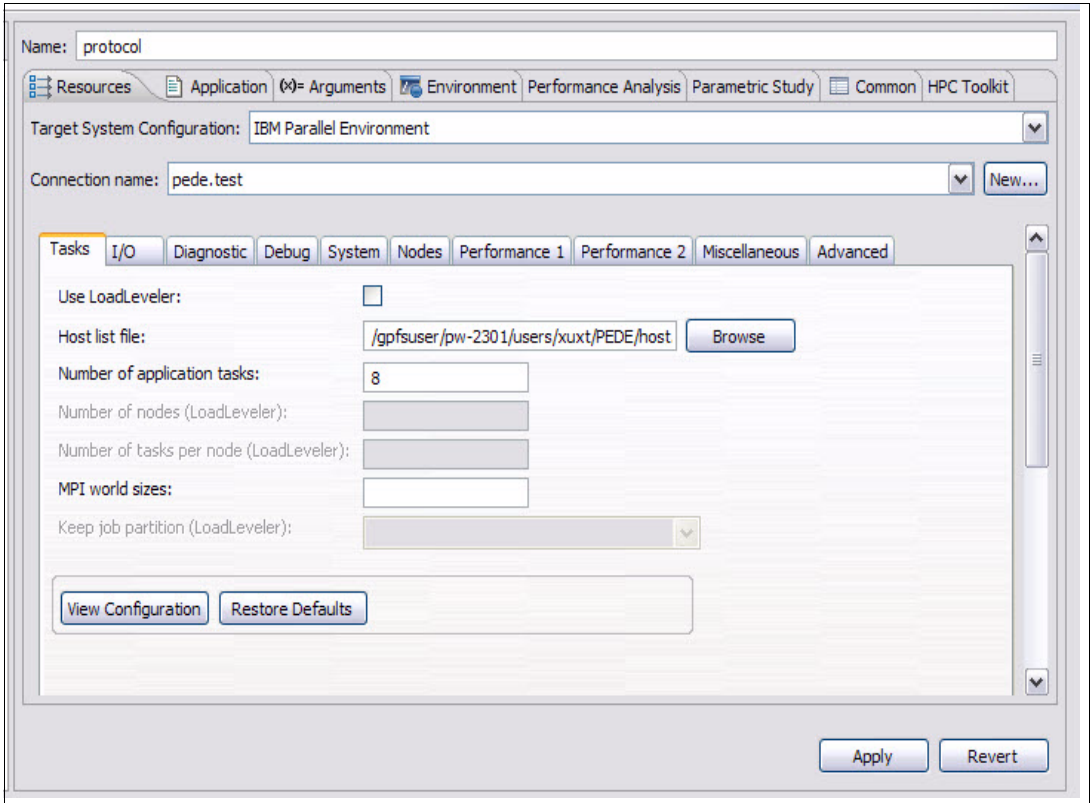


Figure 6-19 Creating a new profile configuration

In the resource tab, choose **IBM Parallel Environment** for the Target System configuration. Click the HPC Toolkit tab and populate the name for the profiling data files, as shown in Figure 6-20 on page 113.

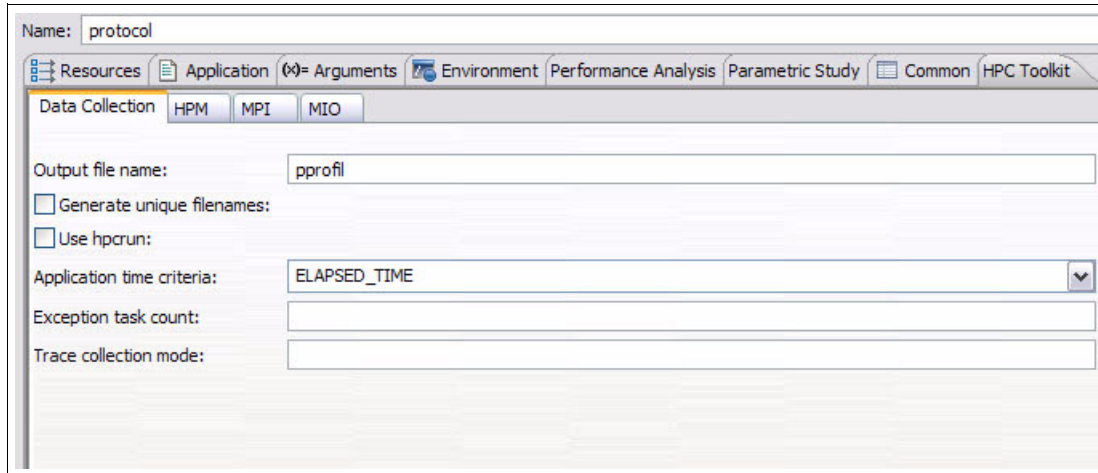


Figure 6-20 Adding the name for the profile data file

You can leave others as default. By default, the MPI profiling library will generate trace files only for the application tasks with the minimum, maximum, and median MPI communication time. This is also true for task zero, if task zero is not the task with minimum, maximum, or median MPI communication time. If you need trace files generated for additional tasks, make sure Output trace for all tasks or the `OUTPUT_ALL_RANKS` environment variable is set correctly. Depending on the number of tasks in your application, make sure Maximum trace rank and Limit trace rank or `MAX_TRACE_RANK` and `TRACE_ALL_TASKS` environment variables are set correctly. If your application executes many MPI function calls, you might need to set the value of Maximum trace events or the `MAX_TRACE_EVENTS` environment variable to a higher number than the default 30,000 MPI function calls. Click **Profile** on the bottom of the pop-up window. After the program completes, some profiling data is generated in your working directory (see `.viz` files in Figure 6-21 as an example) that is read in automatically by Eclipse.

```

hpct_0_6.mpi.viz      pprofil_0_0.mpi.viz  pprofil_0_4.mpi.viz
mpi_64_1             pprofil_0_1.mpi.txt  pprofil_0_7.mpi.txt
pprofil_0_0.mpi.mpt  pprofil_0_1.mpi.viz  pprofil_0_7.mpi.viz
pprofil_0_0.mpi.txt  pprofil_0_4.mpi.txt

```

Figure 6-21 Profiling data generation

The performance data then is shown as a readable format in the HPCT perspective window, as shown in Figure 6-22 on page 114. Each of the MPI routines is tracked, which presents the consumed time and the amount of invoking times. Hence it is easy to analyze the communication overhead of MPI in a parallel application with this data.

Data for rank 0, Aggregation for tasks: 0 1 4 7

Label	Count	WallClock	Transferred By
utils.c			
MPI_Setup(utils.c)			
MPI_Comm_rank_1038	1	0.000003	0.000000
MPI_Gather_1170	1	0.000007	4.000000
MPI_Barrier_1263	1	0.000005	0.000000
MPI_Comm_size_1046	1	0.000000	0.000000
MPI_Gather_1099	1	0.000049	128.000000
MPI_Cleanup(utils.c)			
MPI_Barrier_1332	1	0.042502	0.000000
mpi_1.c			
main(mpi_1.c)			
MPI_Barrier_409	1	0.060408	0.000000
MPI_Barrier_448	1	0.059972	0.000000
MPI_Bcast_628	1622	61.332127	6488.000000
MPI_Recv_511	1622	58.628111	3321856.000000
MPI_Send_502	1622	0.005406	3321856.000000
SUMMARY			
MPI_Bcast	1622	61.332127	6488.000000
MPI_Recv	1622	58.628111	3321856.000000
MPI_Send	1622	0.005406	3321856.000000
MPI_Gather	2	0.000056	132.000000
MPI_Comm_rank	1	0.000003	0.000000
MPI_Barrier	4	0.162887	0.000000
MPI_Comm_size	1	0.000000	0.000000

Figure 6-22 Performance data summary

In addition, the raw data can be presented in a more visual way, as shown in Figure 6-23 on page 115. The Y axis is the application task rank, and the X. If you put the cursor on it in PEDE, more detailed information is displayed.

To open the MPI trace view:

1. Right-click in the performance data summary, and click the **load trace** option in the pop-up menu.
2. Select a path to store a local copy of the trace file in the pop-up file selector dialog. Click **OK** → **Open**. In the MPI Trace window, click **Load Trace**.
3. Choose the corresponding file with the extension name .mpt for loading.

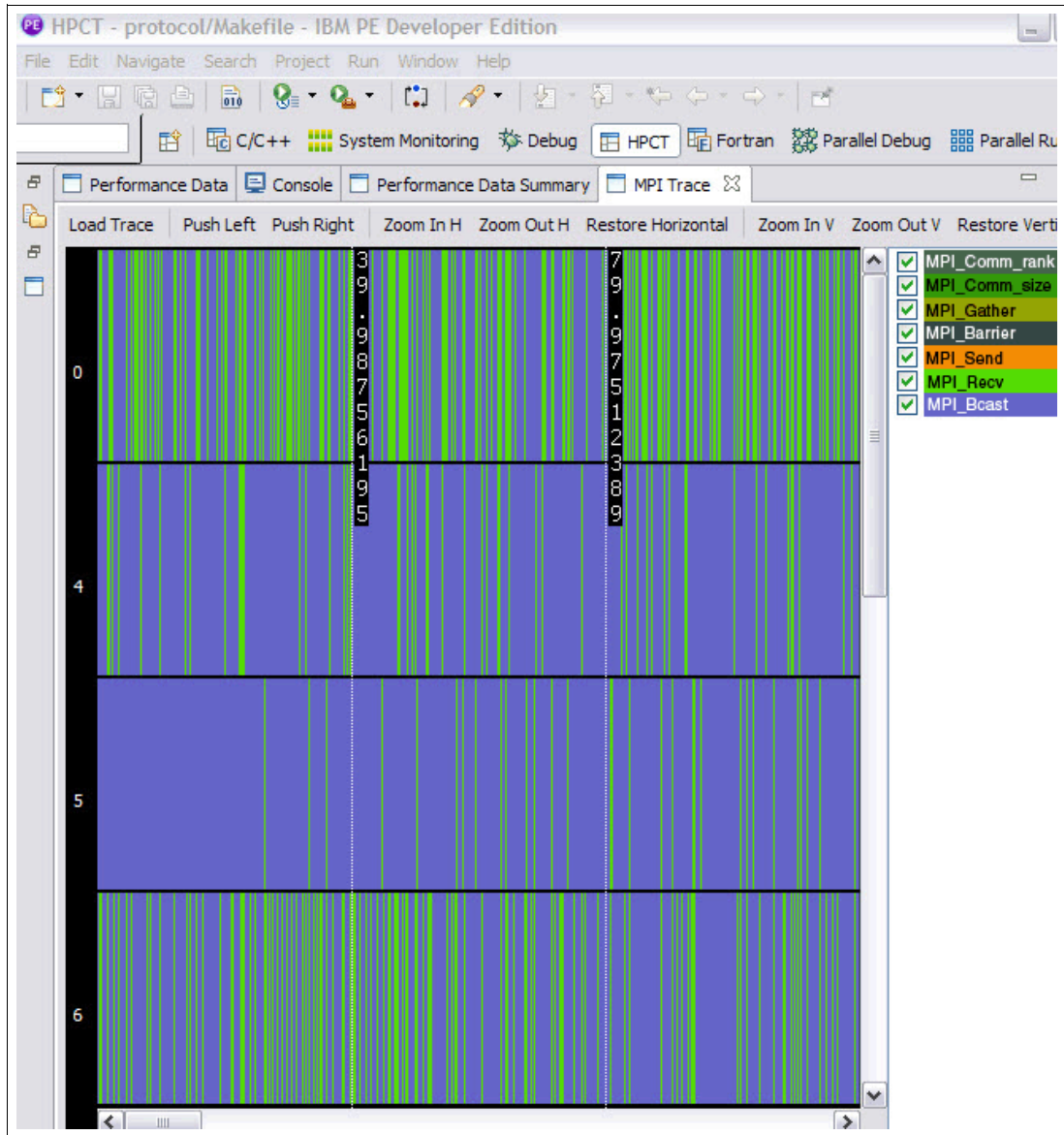


Figure 6-23 Data representation pictorially

6.1.5 OpenMP profiling

Binary instrumenting is the only way to profile OpenMP, and the application must be compiled with `-g -Wl,--hash-style=sysv -emit-stub-syms` compiler flags on a Linux on POWER based system (x86 architecture is not currently supported) or with `-g` for AIX. Besides, the IBM HPC Toolkit only supports OpenMP profiling instrumentation for OpenMP regions that are not nested within other OpenMP constructs at run time. If you set up instrumentation so that nested parallel constructs are instrumented, the results are unpredictable:

1. After the application is properly compiled, right-click the executable from the Eclipse project window, and choose the **HPCT** → **Open** executable, as shown in Figure 6-24 on page 116.
2. Choose what you want to instrument from the OpenMP tab in the Instrumentation view.

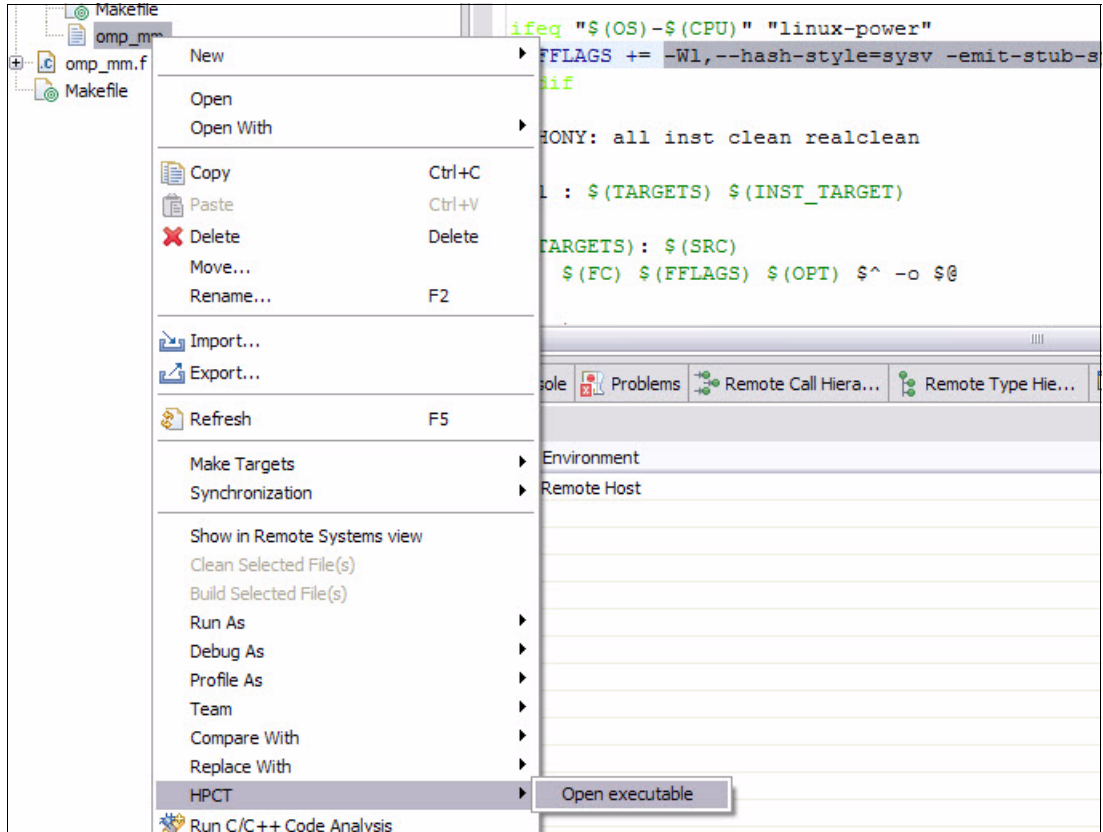


Figure 6-24 Open a executable file for instrumentation

3. Right-click what you chose, and perform the instrumentation, as shown in Figure 6-25. A instrumented executable is created with the `.inst` extension name.

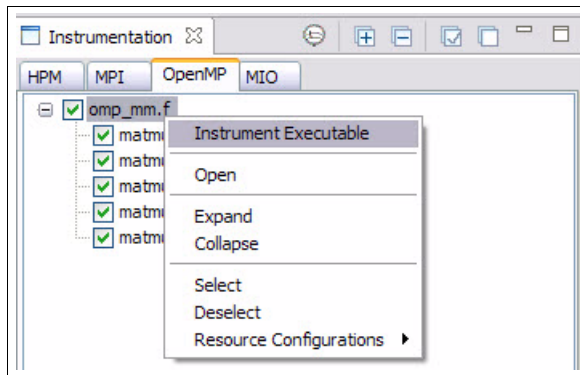


Figure 6-25 Choose openmp source code to instrument

1. Some profiling raw data is generated by running the instrumented binary. After synchronizing from the remote machine, try to open the `.viz` files using Eclipse. You will see profiling data, as shown in Figure 6-26 on page 117.

Label	Count	Excl. Time	Incl. Time	%Total Overhead	%Imbalance	Avg. Thread Time
omp_mm.f						
preion_42	1	0.003125	0.004880	40.932187	68.666295	0.002125
loop_72	1	0.001289	0.001289	7.917129	2665.555556	0.000584

Figure 6-26 Sample OpenMP profiling result

6.1.6 I/O profiling

I/O profiling is where you can obtain information about I/O calls made in your application to help you understand application I/O performance and identify possible I/O performance problems in your application. For example, when an application exhibits the I/O pattern of sequential reading of large files, when environment variables are set appropriately, MIO detects the behavior and invokes its asynchronous prefetching module to prefetch user data.

Currently the MIO tool is only available for parallel applications written in C language and also only able to collect data regarding system I/O library calls (not standard I/O).

Preparing your application

Your application must be compiled and linked with the `-g` compiler flag. When you compile an application on a Power Linux system, you must also use the `-Wl,--hash-style=sysv -emit-stub-syms` compiler flags. For example, there is a sample included in the HPCT package located in the `/opt/ibmhpc/ppedev.hpct/examples/mio` directory. The original Makefile under the `bin64` and the `bin32` subdirectory is already linked with the HPCT library. However we need to do some modifications, as shown in Example 6-3.

Example 6-3 Modification for Makefile in bin64 subdirectory

```
TARGETS = fbs
OBSJS   = fbs.o FBS_encode_data.o FBS_str_to_long.o rtc.o
#LDFLAGS += -L$(IHPCT_BASE)/lib64 -lhpctkio
LDFLAGS += -Wl,--hash-style=sysv -emit-stub-syms
```

To prepare your application:

1. Create a new project named `mio`, and synchronize the project with the remote host.
2. Open the HPCT prospective, as shown in Figure 6-27 on page 118.

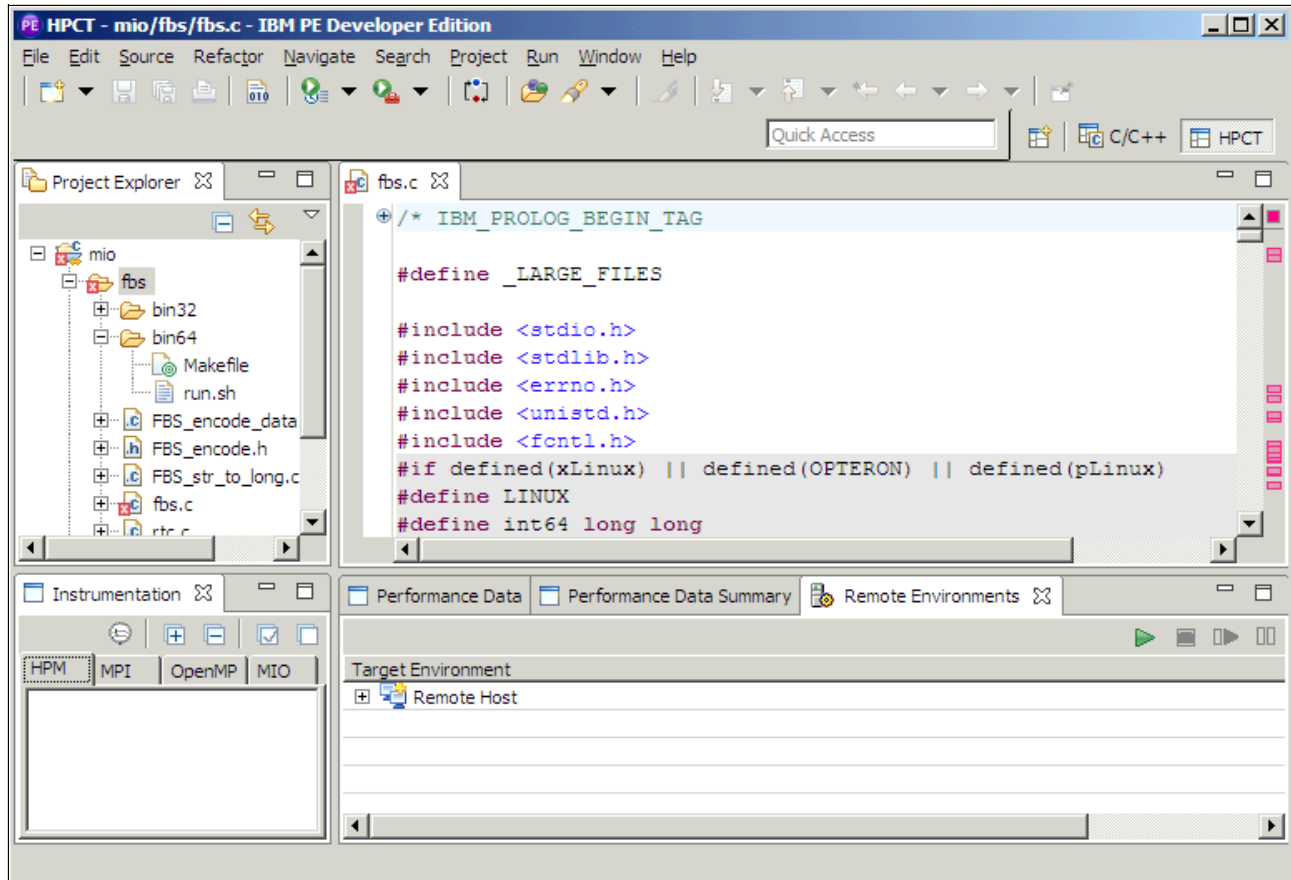


Figure 6-27 Open the project mio

3. Perform the **Make Targets** → **Build** to generate executable binary fbs.

Instrumenting your application

To instrument your application:

1. Select fbs and right-click it. Select **HPCT** → **Open executable**, in the Instrumentation Tab we can see images as shown in Figure 6-28 on page 119.

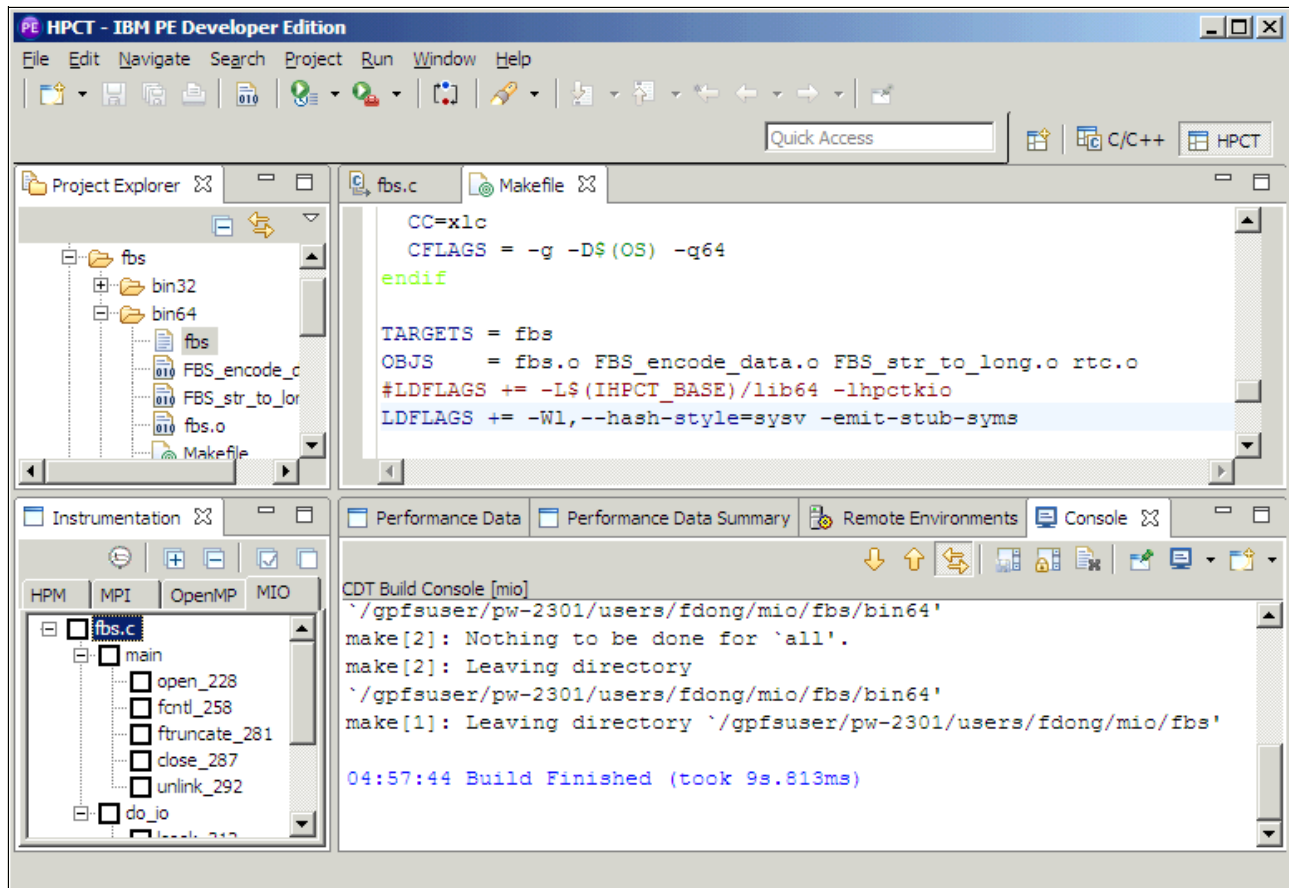


Figure 6-28 HPCT → Open executable

The MIO view shows the application structure tree fully expanded. The leaf nodes are labeled with the name of the system call at the location and the line number in the source file. If you select leaf nodes, instrumentation is placed only at these specific instrumentation points. If you select a non-leaf node, instrumentation is placed at all leaf nodes that are child nodes of the selected non-leaf node.

For I/O profiling to work correctly, you must instrument at least the open and close system calls that open and close any file for which you want to obtain performance measurements.

2. After you select the set of instrumentation that you want, instrument the application by right-clicking the selected node.
3. Click **Instrument Executable**, as shown in Figure 6-29 on page 120.

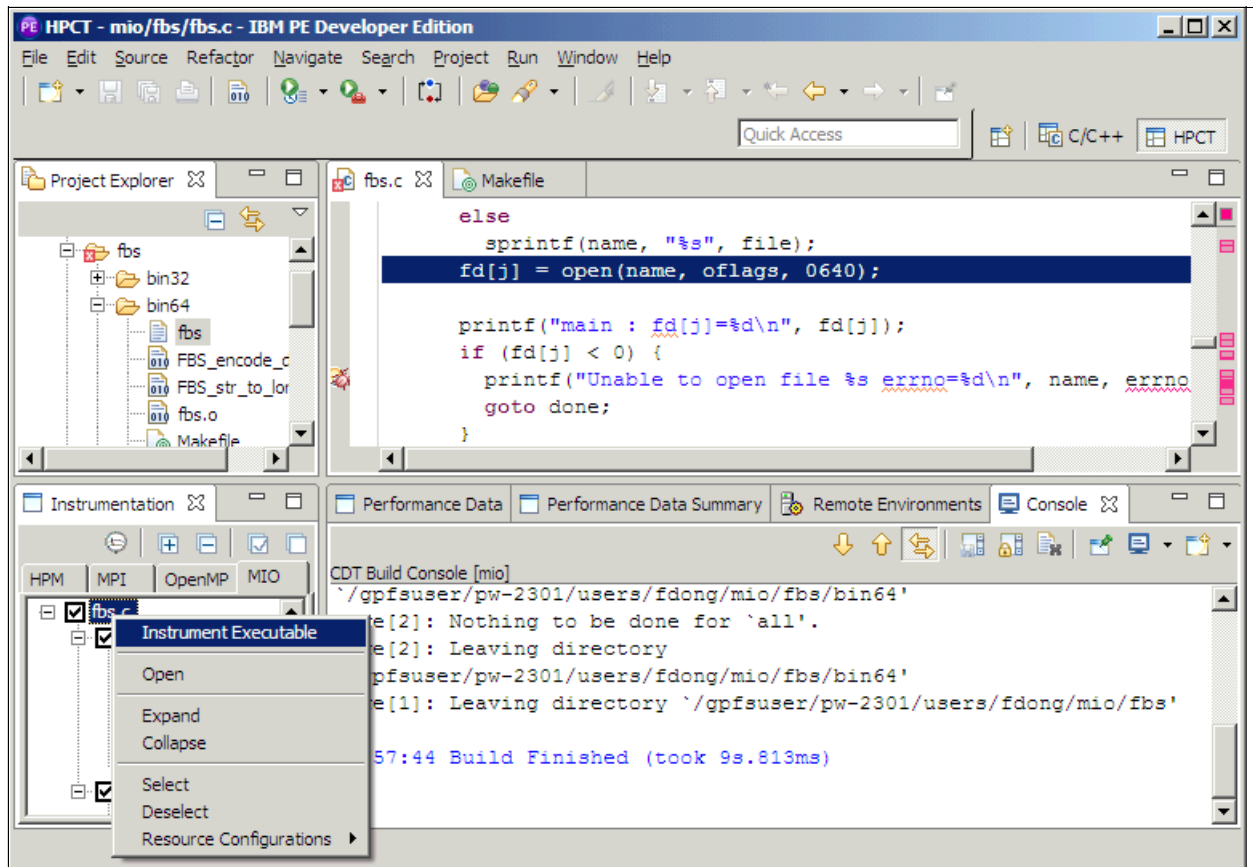


Figure 6-29 Instrument executable

The message in Figure 6-30 is displayed.

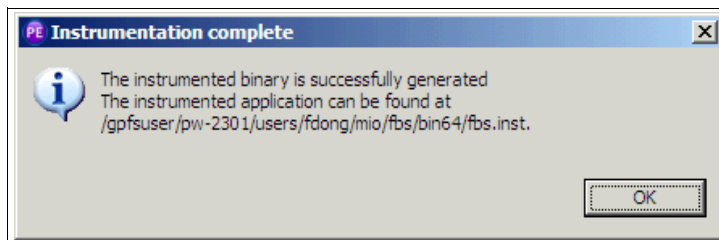


Figure 6-30 The instrumented binary is successfully generated

Instrumenting your application manually

Sometimes we must instrument manually. You must ensure that several environment variables required by the IBM HPC Toolkit are properly set before you use the I/O profiling library. Run the set up scripts located in the top-level directory of your installation, which is normally in the `/opt/ibmhpc/ppdev.hpct` directory. If you use `sh`, `bash`, `ksh` or a similar shell command, invoke the `env_sh` script as `.env_sh`. If you use `csh`, invoke the `env_csh` script as `source env_csh`.

To profile your application, you must link your application with the `libhpctkio` library using the `-L$IHPCCT_BASE/lib` and `-lhpcctkio` linking options for 32-bit applications or using the `-L$IHPCCT_BASE/lib64` and `-lhpcctkio` linking options for 64-bit applications.

You must also set the TKIO_ALTLIB environment variable to the path name of an interface module used by the I/O profiling library before you invoke your application:

- ▶ For 32-bit applications, set the TKIO_ALTLIB environment variable to \$IHPCT_BASE/lib/get_hpcmio_ptrs.so.
- ▶ For 64-bit applications, set the TKIO_ALTLIB environment variable to \$IHPCT_BASE/lib64/get_hpcmio_ptrs.so.

Optionally, the I/O profiling library can print messages when the interface module is loaded, and it can abort your application if the interface module cannot be loaded.

For the I/O profiling library to display a message when the interface module is loaded, you must append /print to the setting of the TKIO_ALTLIB environment variable. For the I/O profiling library to abort your application if the interface module cannot be loaded, you must append /abort to the setting of the TKIO_ALTLIB environment variable. You might specify one, both, or non of these options.

Note that there are no spaces between the interface library path name and the options. For example, load the interface library for a 64-bit application, display a message when the interface library is loaded, and abort the application if the interface library cannot be loaded. Issue the following command:

```
export TKIO_ALTLIB="$IHPCT_BASE/lib64/get_hpcmio_ptrs.so/print/abort"
```

During the run of the application, the following message prints:

```
TKIO : fbs : successful  
load("/opt/ibmhpc/ppdev.hpct//lib64/get_hpcmio_ptrs.so") version=3013
```

Environment variables for I/O profiling

I/O profiling works by intercepting I/O system calls for any files that you want to obtain performance measurements. To obtain the performance measurement data, the IBM HPC Toolkit uses the I/O profiling options (MIO_FILES) settings and other environment variables.

The first environment variable is MIO_FILES, which specifies one or more sets of file names and the profiling library options to be applied to that file, where the file name might be a pattern or an actual path name.

The second environment variable is MIO_DEFAULTS, which specifies the I/O profiling options to be applied to any file whose file name does not match any of the file name patterns specified in the MIO_FILES environment variable. If MIO_DEFAULTS is not set, no default actions are performed.

The file name that is specified in the MIO_FILES variable setting might be a simple file name specification, which is used as-is, or it might contain wildcard characters, where the allowed wildcard characters are:

- ▶ A single asterisk (*), which matches zero or more characters of a file name.
- ▶ A question mark (?), which matches a single character in a file name.
- ▶ Two asterisks (**), which match all remaining characters of a file name.

The I/O profiling library contains a set of modules that can be used to profile your application and to tune I/O performance. Each module is associated with a set of options. Options for a module are specified in a list and are delimited by / characters. If an option requires a string argument, that argument must be enclosed in brackets {}, if the argument string contains a / character.

Multiple modules can be specified in the settings for both MIO_DEFAULTS and MIO_FILES. For MIO_FILES, module specifications are delimited by a pipe (|) character. For MIO_DEFAULTS, module specifications are delimited by commas (,).

Multiple file names and file name patterns can be associated with a set of module specifications in the MIO_FILES environment variable. Individual file names and file name patterns are delimited by colon (:) characters. Module specifications associated with a set of file names and file name patterns follow the set of file names and file name patterns and are enclosed in square brackets ([]).

The run.sh script under bin64 subdirectory, already include MIO_DEFAULTS and MIO_FILES environment variable settings.

As an example of the MIO_DEFAULTS environment variable setting, assume that the default options for any file that does not match the file names or patterns specified in the MIO_FILES environment variable are that the trace module is to be used with the stats, mbytes, and inter options, and the pf module is to be used with the stats option.

```
export MIO_DEFAULTS="trace/mbytes/stats/inter,pf/stats"
```

As an example of using the MIO_FILES environment variable, assume that the application does I/O to *.dat. The following setting will cause files matching *.dat to use the trace module with global cache, stats, xml, and events options.

```
export MIO_FILES="*.dat[trace/global=pre_pf/stats={stats}/xml/events={evt} ]"
```

You can just include those environment variable settings in run.sh or put them in the MIO sub-tab under the HPC Toolkit tab in Profile Configurations, as shown in Figure 6-31.

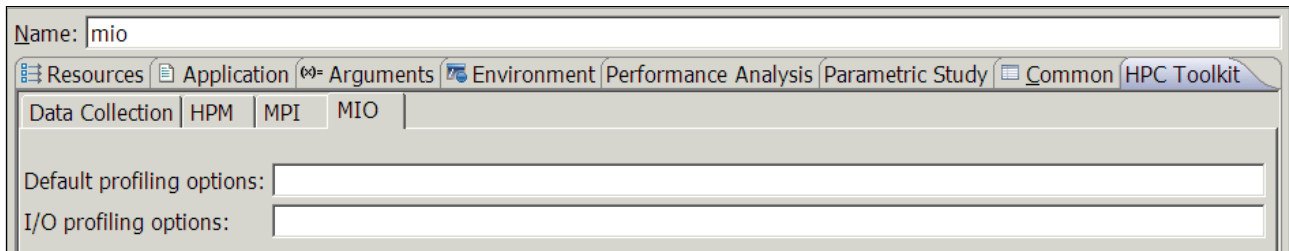


Figure 6-31 MIO sub-tab under HPC Toolkit tab

MIO_DEFAULTS refer to Default profiling options, and MIO_FILES refer to I/O profiling options.

Specifying I/O profiling library module options

Table 6-5 shows the modules that are available in the I/O profiling library.

Table 6-5 MIO analysis modules

Module	Purpose
mio	The interface to the user program
pf	A data prefetching module
trace	A statistics gathering module
recov	Analyzes failed I/O accesses and retries in case of failure

The **mio** module has the following options as shown in Table 6-6 on page 123.

Table 6-6 MIO module options

Option	Purpose
mode=	Override the file access mode in the open system call.
nomode	Do not override the file access mode.
direct	Set the O_DIRECT bit in the open system call.
nodirect	Clear the O_DIRECT bit in the open system call.

The default option for the mio module is nomode. The pf module has the options, as shown in Table 6-7.

Table 6-7 MIO pf module options

Option	Purpose
norelease	Do not free the global cache pages when the global cache file usage count goes to zero. The release and norelease options control what happens to a global cache when the file usage count goes to zero. The default behavior is to close and release the global cache. If a global cache is opened and closed multiple times, there can be memory fragmentation issues at some point. Using the norelease option keeps the global cache opened and available, even if the file usage count goes to zero.
release	Free the global cache pages when the global cache file usage count goes to zero.
private	Use a private cache. Only the file that opens the cache might use it.
global=	Use global cache, where the number of global caches is specified as a value between 0 and 255. The default is 1, which means that one global cache is used.
asynchronous	Use asynchronous calls to the child module.
synchronous	Use synchronous calls to the child module.
noasynchronous	Alias for synchronous.
direct	Use direct I/O.
nodirect	Do not use direct I/O.
bytes	Stats output is reported in units of bytes.
kbytes	Stats is reported in output in units of kbytes.
mbytes	Stats is reported in output in units of mbytes.
gbytes	Stats is reported in output in units of gbytes.
tbytes	Stats is reported in output in units of tbytes.
cache_size=	The total size of the cache (in bytes), between the values of 0 and 1GB, with a default value of 64 K.
page_size=	The size of each cache page (in bytes), between the value of 4096 bytes and 1 GB, with a default value of 4096.
prefetch=	The number of pages to prefetch, between 1 and 100, with a default of 1.
stride=	Stride factor, in pages, between 1 and 1G pages, with a default value of 1.
stats=	Output prefetch usage statistics to the specified file. If the file name is specified as mioout, or no file name is specified, the statistics file name is determined by the setting of the MIO_STATS environment variable.

Option	Purpose
nostats	Do not output prefetch usage statistics.
inter	Output intermediate prefetch usage statistics on kill -USR1.
nointer	Do not output intermediate prefetch usage statistics.
retain	Retain file data after close for subsequent reopen.
noretain	Do not retain file data after close for subsequent reopen.
listio	Use listio mechanism.
nolistio	Do not use listio mechanism.
tag=	String to prefix stats flow.
notag	Do not use prefix stats flow.

The default options for the pf module are:

```
/nodirect/stats=mioout/bytes/cache_size=64k/page_size=4k/
prefetch=1/asynchronous/global/release/stride=1/nolistio/notag
```

The trace module has the options shown in Table 6-8.

Table 6-8 MIO trace module options

Option	Purpose
stats=	Output trace statistics to the specified file name. If the file name is specified as mioout, or no file name is specified, the statistics file name is determined by the setting of the MIO_STATS environment variable.
nostats	Do not output statistics on close.
events=	Generate a binary events file. The default file name if this option is specified is trace.events.
noevents	Do not generate a binary events file.
bytes	Output statistics in units of bytes.
kbytes	Output statistics in units of kilobytes.
mbytes	Output statistics in units of megabytes.
gbytes	Output statistics in units of gigabytes.
tbytes	Output statistics in units of terabytes.
inter	Output intermediate trace usage statistics on kill -USR1.
nointer	Do not output intermediate statistics.
xml	Generate statistics file in a format that can be viewed using peekperf.

The default options for the trace module are:

```
/stats=mioout/noevents/nointer/bytes
```

The recov module has the options, as shown in Table 6-9 on page 125.

Table 6-9 MIO recov module options

Option	Purpose
fullwrite	All writes are expected to be full writes. If there is a write failure because of insufficient space, the recov module retries the write.
partialwrite	All writes are not expected to be full writes. If there is a write failure because of insufficient space, there will be no retry.
stats=	Output recov module statistics to the specified file name. If the file name is specified as mioout, or no file name is specified, and the statistics file name is determined by the setting of the MIO_STATS environment variable.
nostats	Do not output recov statistics on file close.
command=	The system command to be issued on a write error.
open_command=	The system command to be issued on open error resulting from a connection that was refused.
retry=	Number of times to retry, between 0 and 100, with a default of 1.

The default options for the recov module are:

partialwrite/retry=1

Running your application

To run your application:

1. Right-click the run.sh script under bin64 subdirectory, and select **Profile As** → **Profile Configurations**.
2. Using the profile configuration dialog, generate a mio profile configuration under the Parallel Application, as shown in Figure 6-32 to create a profile configuration.

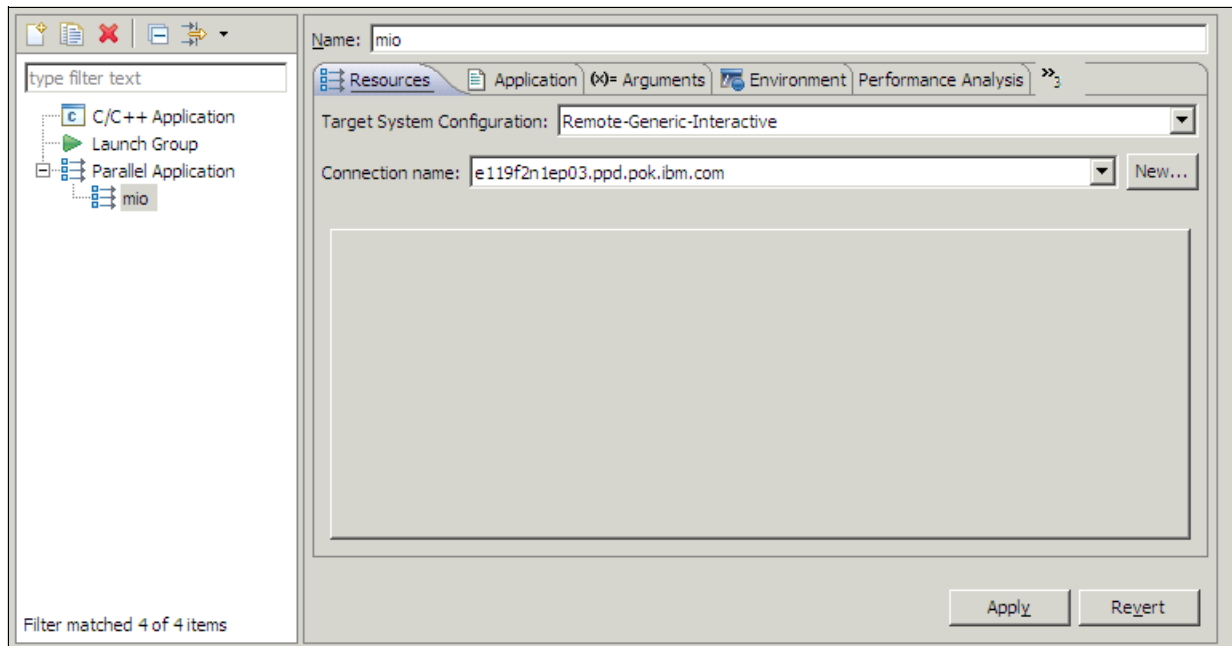


Figure 6-32 Create profile launch configuration

3. In the Application tab, select the run.sh script, as shown in Figure 6-33 on page 126.

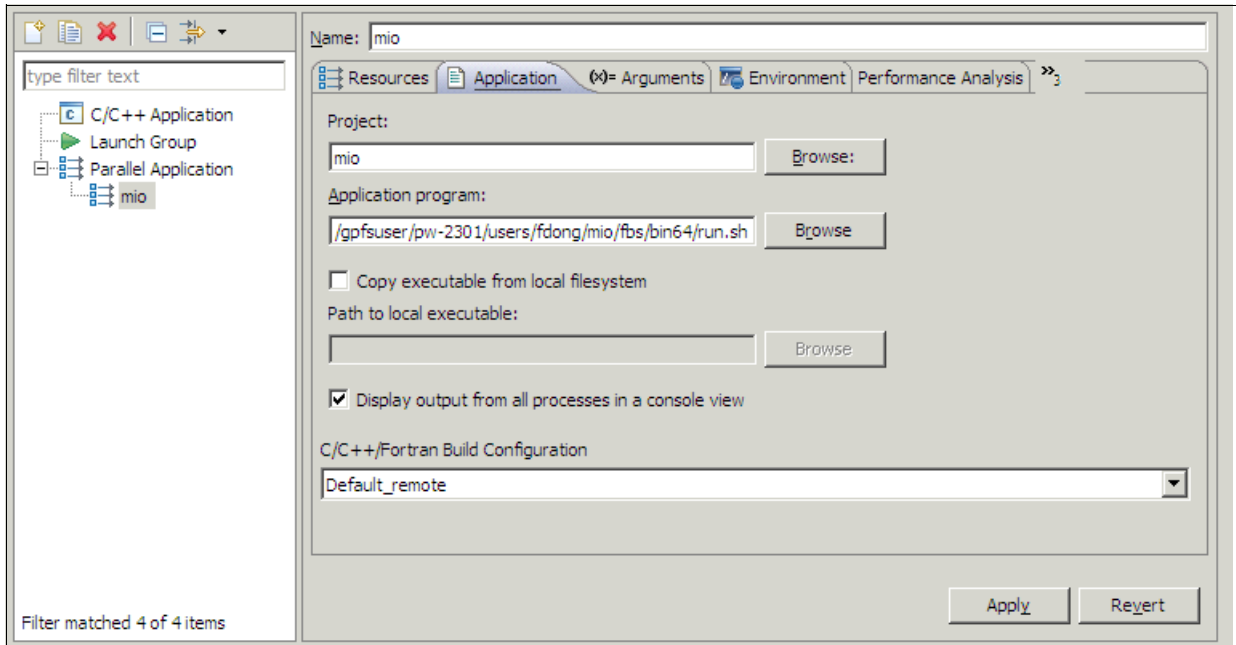


Figure 6-33 Application tab

4. Input fbs.inst as the argument for run.sh, and set the working directory as /gpfsuser/pw-2301/users/fdong/mio/fbs/bin64, as shown in Figure 6-34.

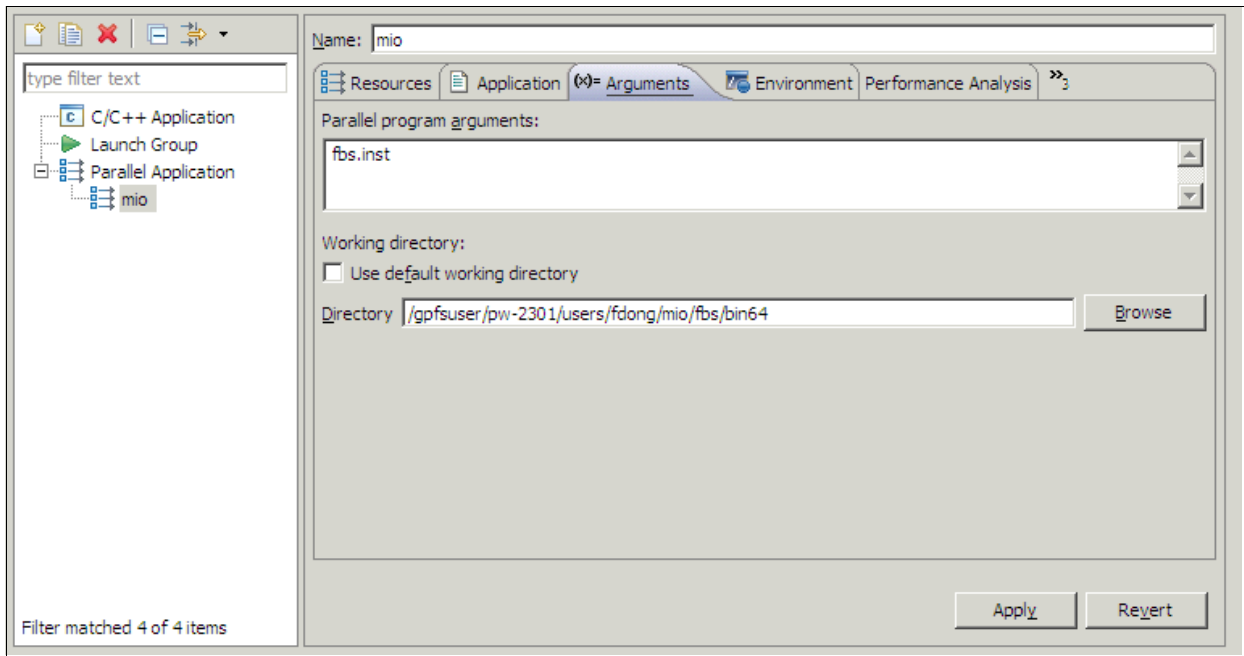


Figure 6-34 Arguments tab

5. Click **Apply** → **Profile**, after the application complete, asks Do you want to automatically open these visualization files, as shown in Figure 6-35 on page 127.

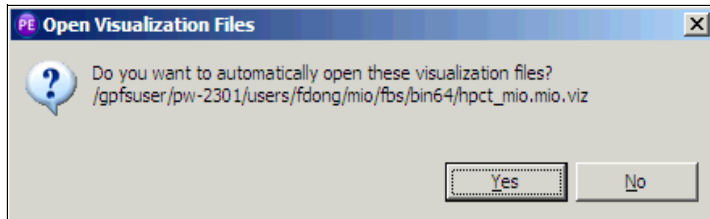


Figure 6-35 Open visualization files

- Click **Yes**. The plug-in attempts to display the I/O profiling data that was collected when the application was run. You get the visualization file open in the Performance Data tab, as shown in Figure 6-36 (visualization files in the performance data tab).

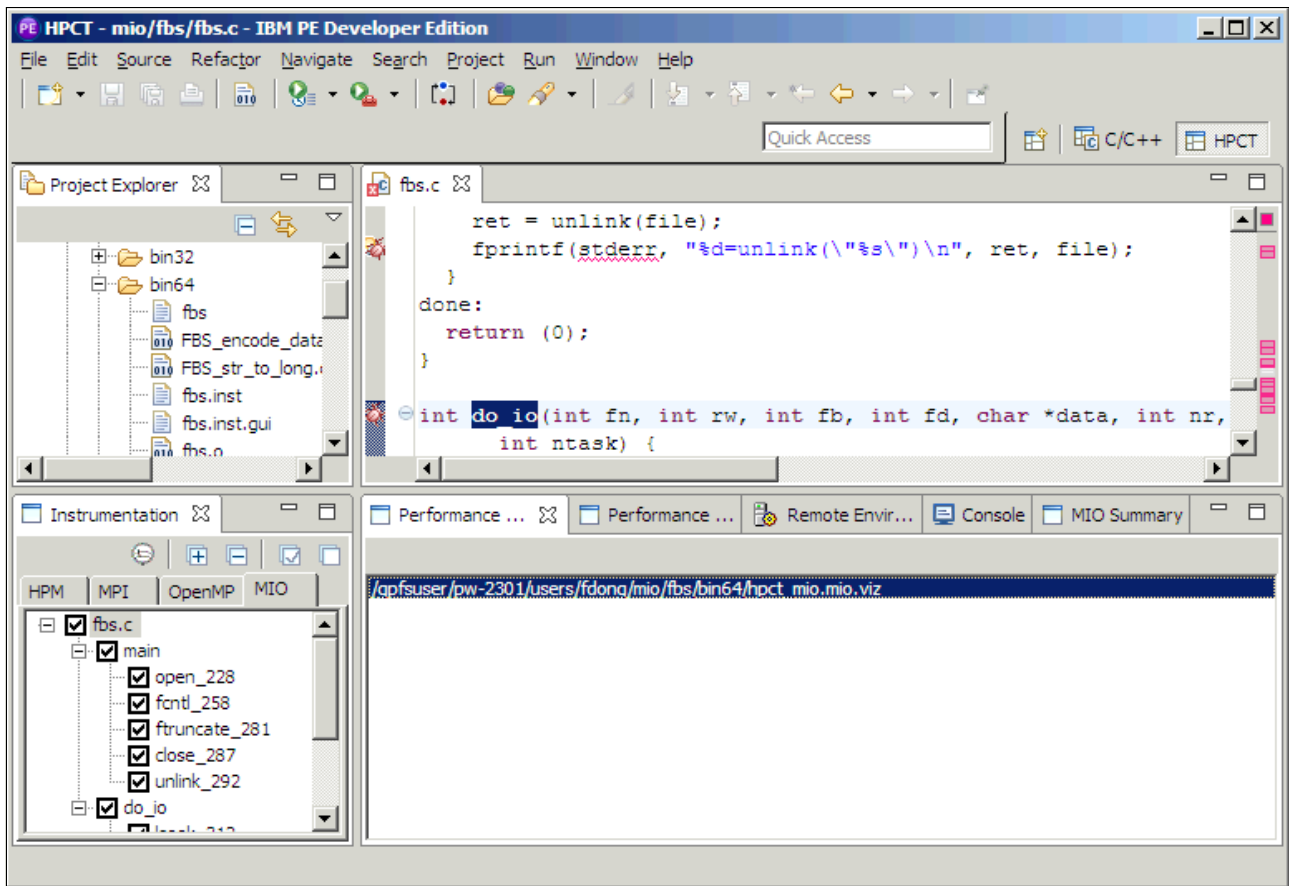


Figure 6-36 Visualization files in performance data tab

The plug-in displays the data in a tree format, in which the top-level node is the file that the application read or wrote and the leaf nodes are the I/O function calls your application issued for that file. Figure 6-37 on page 128 shows the data visualization window with this tree fully expanded.

Label	EVENT COUNT	CUMULATIVE TIME[SECS]
/tmp/spill_test_19648.dat		
seek	303	0.00
read	600	0.01
write	300	0.01
close	1	0.00
fcntl	2	0.00
open	1	0.00

Figure 6-37 Performance data summary view with I/O profiling data

Each row shows the time spent in an I/O function call and the number of times that the function call is executed.

7. You can view detailed data for a leaf node by right-clicking over it and selecting **Show Metric Browser** from the pop-up menu. A metric browser window contains data for each process that executed that I/O function. You can view all of your performance measurements in a tabular form by selecting the **Show Data as a Flat Table** option from the pop-up menu that appears when you right-click within the **Performance Data Summary** view.
8. You can view a plot of your I/O measurements by right-clicking in the **Performance Data Summary** view, selecting **Load IO Trace** from the pop-up menu that appears, and specifying the location to download the I/O trace file like `hpct_mio.mio.evt.iot`. After the trace is loaded, the Eclipse window looks like Figure 6-38 (MIO summary).

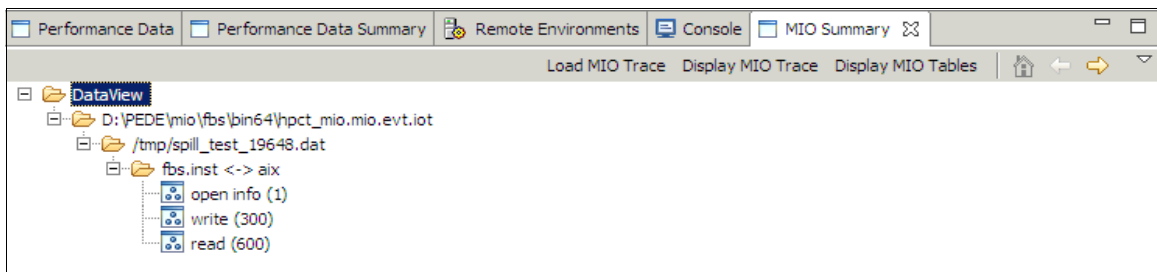


Figure 6-38 MIO summary

The MIO Summary view contains a tree view of the MIO performance data files. The top-level nodes represent individual performance data files. The next level nodes represent individual files that the application accessed. The next level nodes represent the application program. You can select leaf nodes to include the data from those nodes in the plot window.

You can use the buttons in the view's toolbar or the menu options in the view's drop-down menu to perform the following actions (Table 6-10).

Table 6-10 MIO trace processing actions

Button	Action
Load MIO Trace	Load a new I/O trace file.
Display MIO Trace	Display a new I/O trace file.
Display MIO Tables	Display data from the I/O trace in a tabular format.

After you select write and read leaf nodes from the tree and click the **Display MIO Trace** button, the Eclipse window looks like Figure 6-39 on page 129 (MIO trace view).

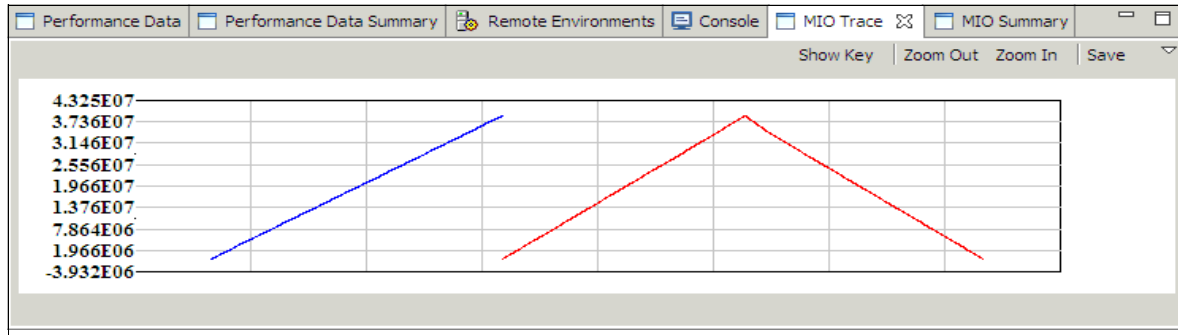


Figure 6-39 MIO trace view

We can see that, the application is writing a file sequentially, after finished, read the file from beginning to the end, and then reversed. The blue line is write operation, and the red line is read operation.

When the graph is initially displayed, the Y axis represents the file position, in bytes. The X axis of the graph always represents time in seconds.

You can zoom into an area of interest in the graph by left-clicking at one corner of the desired area and dragging the mouse while holding the left button to draw a box around the area of interest and then releasing the left-mouse button. When you release the left-mouse button, the plug-in redraws the graph, showing the area of interest. You can then zoom in and out of the graph by clicking the **Zoom In** and **Zoom Out** buttons at the top of the graph window. As you drag the mouse, the plug-in displays the X and Y coordinates of the lower-left corner of the box, the upper-right corner of the box, and the slope of the line between those two corners as text in the status bar area at the bottom of the view.

You can determine the I/O data transfer rate at any area in the plot by right-clicking over the desired starting point in the plot and holding down the right-mouse button, while tracing over the section of the plot of interest. The coordinates of the starting and ending points of the selection region and the data transfer rate (slope) are displayed in the status area at the bottom of the view.

You can save the current plot to a jpeg file by clicking **Save** at the top of the view. A file selector dialog appears, which allows you to select the path name of the file to which the screen image will be written.

You can display a pop-up dialog that lists the colors in the current graph and the I/O functions they are associated with by clicking **Show Key** at the top of the view.

You can view the I/O profiling data in tabular form and modify the characteristics of the current plot by selecting **Display MIO Tables** at the top of the MIO Summary view. A window similar to Figure 6-40 on page 130 (dataview table view) is displayed.

start	delta	pos	nbytes	new_pos	finish	data rate
9.2443E-02	1.8835E-05	1	131072	1.7957E07	9.2462E-02	6.9589E09
9.3170E-02	2.0027E-05	1	131072	1.5991E07	9.3190E-02	6.5447E09
9.6173E-02	1.9073E-05	7733248	131072	7.8643E06	9.6192E-02	6.8719E09
8.9012E-02	1.9073E-05	2	131072	2.7263E07	8.9031E-02	6.8719E09
8.0684E-02	1.9073E-05	2	131072	2.7918E07	8.0703E-02	6.8719E09
7.4682E-02	2.0027E-05	1	131072	1.1665E07	7.4702E-02	6.5447E09
9.8590E-02	1.9073E-05	1179648	131072	1.3107E06	9.8609E-02	6.8719E09
7.2847E-02	2.0981E-05	6553600	131072	6.6847E06	7.2868E-02	6.2472E09
9.5543E-02	1.9073E-05	9437184	131072	9.5683E06	9.5563E-02	6.8719E09

Figure 6-40 DataView table view

There are four widgets at the top of the table view that you can use to modify the characteristics of the current plot. You can change the values in these widgets as desired. The selections you make in this view are effective the next time you click **Display MIO Trace** in the MIO Summary view.

The colored square at the upper left specifies the color to use when drawing the plot. If you click this square, a color selector dialog appears, which allows you to select the color you want to be used in drawing the plot.

The second widget from the left, labeled file position activity, selects the metric to be used for the Y and X axis of the plot and also affects the format of the plot. If you select **file position activity**, the Y axis represents the file position and the X axis represents time. If you select **data delivery rate**, the Y axis represents the data transfer rate and the X axis represents time. If you select rate versus pos, the Y axis represents the data transfer rate and the X axis represents the start position in the file.

The third widget from the left specifies the pixel width for the graph that is drawn when the file position metric is selected from the second widget from the left.

The right most widget specifies the metric that has its numeric value displayed next to each data point. You can select any column displayed in the table, or **none** to plot each point with no accompanying data value.

6.1.7 X Windows Performance Profiler

The X Windows Performance Profiler (Xprof) is a fronted tool for profiling data generated by running an application that was compiled and linked with the **-pg** option. It assists in identifying most CPU-Intensive functions in parallel applications. It comes with the IBM HPC Toolkit, although it is not currently integrated within Eclipse. As a consequence, the tool must be started in the target system manually and exported to the graphical view using either X-forwarding or VNC techniques.

Preparing your application

The parallel application must be compiled using the **-pg** flag. Optionally, it can also be compiled with **-g** flag so that Xprof can get the connection to the line of source code.

Profiling with Xprof

To profile with Xprof:

1. Compile the application using the **-pg** option.
2. Run the application to generate gmon.out profile data files.

3. Open the Xprof GUI passing the binary and gmon.out files as arguments (Example 6-4). Optionally you can just start Xprof without arguments and then select **File** → **Load Files** to select and load the required files.

Example 6-4 Starting Xprof GUI on Linux

```
$ source /opt/ibmhpc/ppdev.hpct/env_sh
$ Xprof ./Gadget2 profdir.0_0/gmon.out profdir.0_1/gmon.out
```

Observe that gmon.out files are generated with different names in Linux and AIX operating systems, respectively, *profdir.<world_id>_<task_id>/gmon.out* and *gmon.<world_id>_<task_id>.out*.

Interpreting profile information

After binary and profile files are loaded, the main panel is going to display a call graph chart of application execution for a consolidated visualization of all data collected (Figure 6-41 on page 132). In that chart, nodes are application methods and arcs are method calls, so a pair of node-arc represents the caller/callee relationship at runtime. The rectangles embodying a node represents time spent in a method and its callees, with a representation of time spent in the method itself plus its callees and height represents the time spent only in that method.

It is also possible to do several operations in the call graph main view, such as:

- ▶ Apply filters to cluster or uncluster methods (menu **Filter** → **uncluster Function**).
- ▶ Access detailed information about each function (right click in a rectangle).
- ▶ Access detailed information about specific caller/callee flow, including the number of times that pair was executed (right-click in an arc).

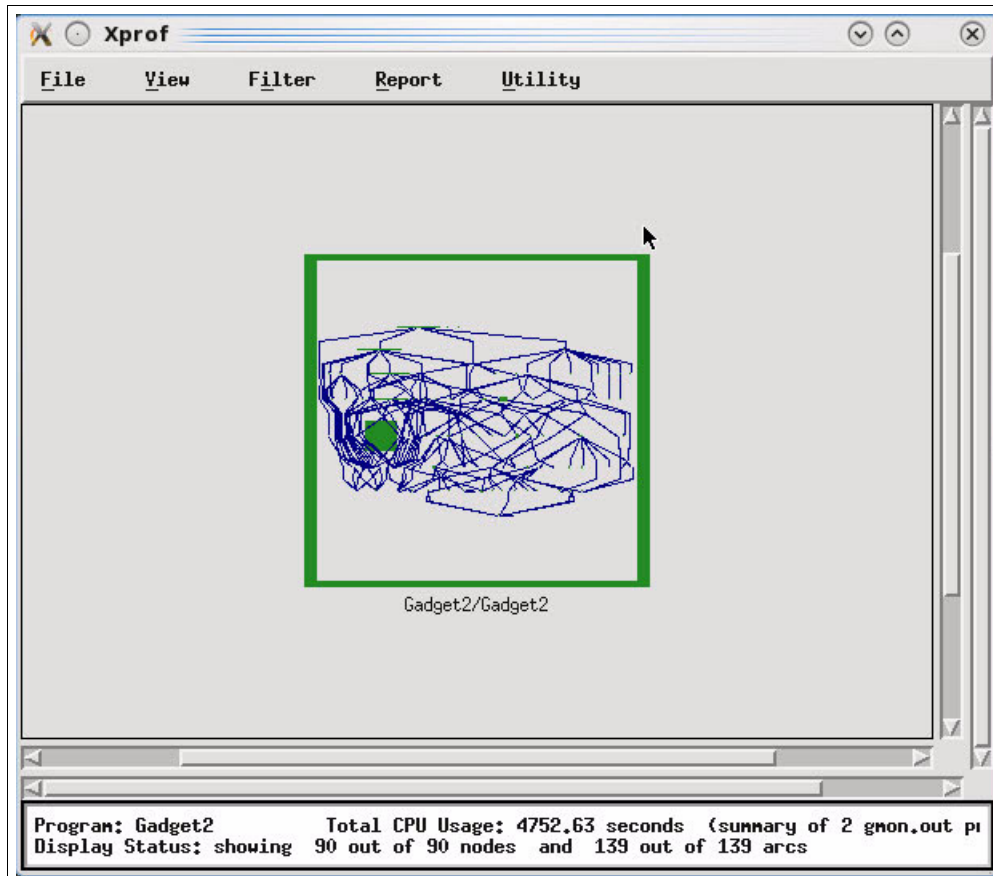


Figure 6-41 Xprof main window: call graph

The tool supports several visualization modes and reports that are accessible from the Reports menu. It is also possible to navigate from a report back (and fourth) to the call graph view. Some of the available reports are:

- ▶ Flat profile report
- ▶ Call graph (plain text) report
- ▶ Function call statistics report
- ▶ Library statistics report

Indeed those reports are rich and useful to easily identify hotspots in the source code, for example, the flat profile report sorts out the application functions by accumulated time spent in each one, thus highlighting the most CPU-intensive on the top (Figure 6-42 on page 133).

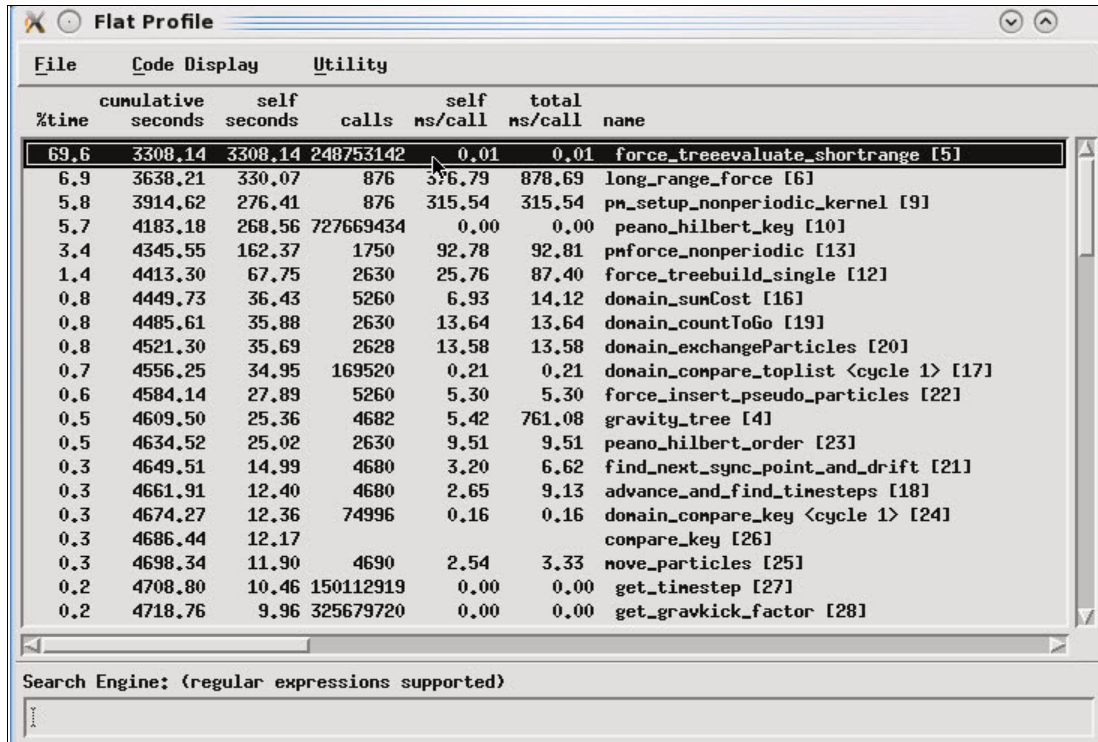


Figure 6-42 Xprof: flat profile report

6.2 Debugging

This section provides debugging information.

6.2.1 Parallel Static Analysis

The IBM Parallel Environment (PE) Developer Edition comes with a set of tools for static analysis of parallel application source code. They can show artifacts and make analysis for the parallel technologies shown in Table 6-11.

Table 6-11 Parallel static analysis capabilities versus parallel technologies

Technology	Show artifacts	Analysis
MPI	Yes	Yes
OpenMP	Yes	No
LAPI	Yes	No
OpenACC	Yes	No
OpenSHMEM	Yes	No
PAMI	Yes	No
UPC	Yes	No

Any of the parallel static analyze tools of Table 6-11 are executed from the drop-down menu in the Eclipse toolbar, as shown in Figure 6-43 on page 134.

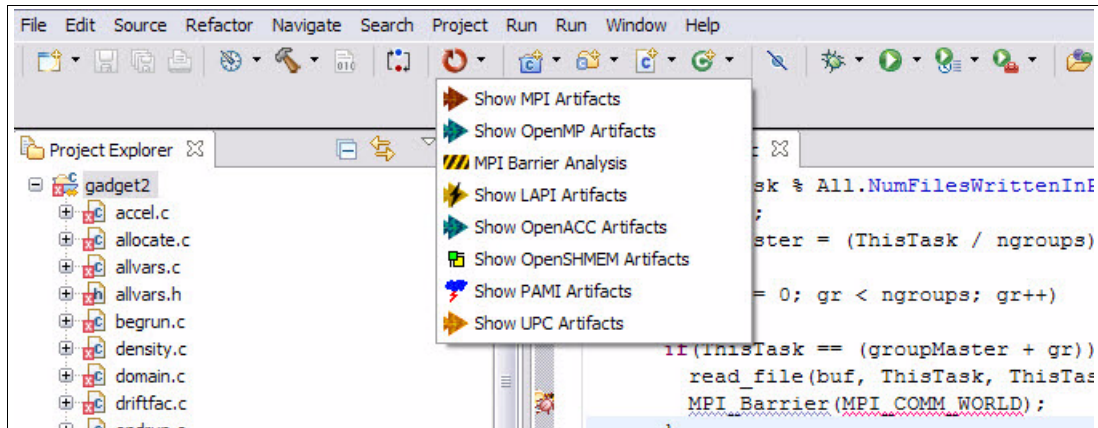


Figure 6-43 Parallel static analysis menu

The tool scan the project files to gather data and then generate reports with artifact types (Figure 6-44) being used and their exact location in the source code, as shown in Figure 6-45.

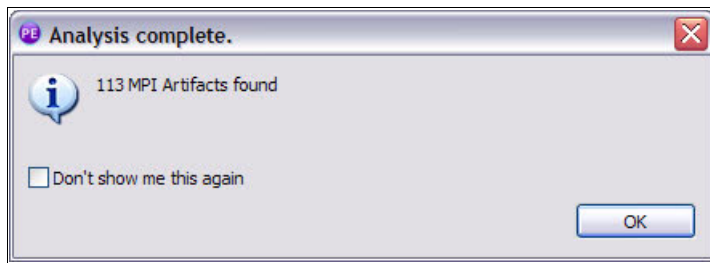


Figure 6-44 Parallel analysis message displayed after finish analysis

 A screenshot of the 'MPI Artifact View' window in Eclipse. The window title bar includes 'Problems', 'Tasks', 'Console', 'Properties', 'Remote Environments', and 'MPI Artifact View'. Below the title bar are several status indicators: 'MPI Barrier Matches', 'MPI Barriers', and 'MPI Barrier Errors'. The main area contains a table with the following data:

Artifact	Filename	LineNo	Construct
MPI_Allreduce	timestep.c	593	Function Call
MPI_Allreduce	timestep.c	594	Function Call
MPI_Allgather	timestep.c	597	Function Call
MPI_Bcast	run.c	86	Function Call
MPI_Barrier	run.c	91	Function Call
MPI_Bcast	run.c	119	Function Call
MPI_Allreduce	run.c	165	Function Call
MPI_Allreduce	run.c	173	Function Call
MPI_Allgather	run.c	195	Function Call

Figure 6-45 Parallel analysis report view for MPI project

If the tools cannot find the artifacts in your source code, some additional configuration might be needed on Eclipse. For OpenMP artifacts, select **Window** → **Preferences** and choose **Parallel Tools** → **Parallel Language Development Tools** → **OpenMP**. Make sure to enable the option **Recognize OpenMP Artifacts by prefix (omp_) alone?**, and add the OpenMP include paths of your local system. If you do not have the OpenMP include files in your local system, you can add any path (for example, the path to the project on your workspace). Figure 6-46 on page 135 shows the screen used for OpenMP artifacts configuration.

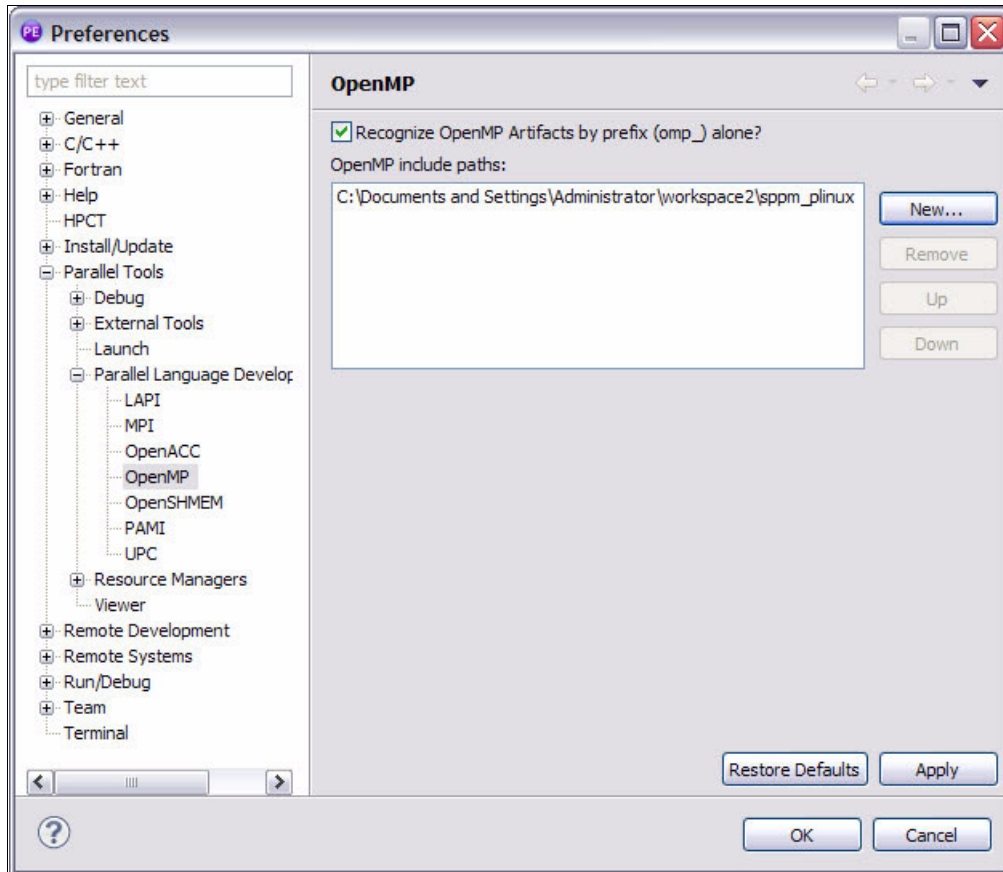


Figure 6-46 OpenMP artifacts configuration

To be able to identify the UPC artifacts, a similar configuration might be needed as well. Go to menu **Window** → **Preferences**, and choose **Parallel Tools** → **Parallel Language Development Tools** → **UPC**. Make sure to enable the option **Recognize APIs by prefix (upc_) alone?**, and add the UPC include paths of your local system. If you do not have the UPC include files in your local system, you can add any path (for example, the path to the project on your workspace). Figure 6-47 on page 136 shows the screen used for UPC artifacts configuration.

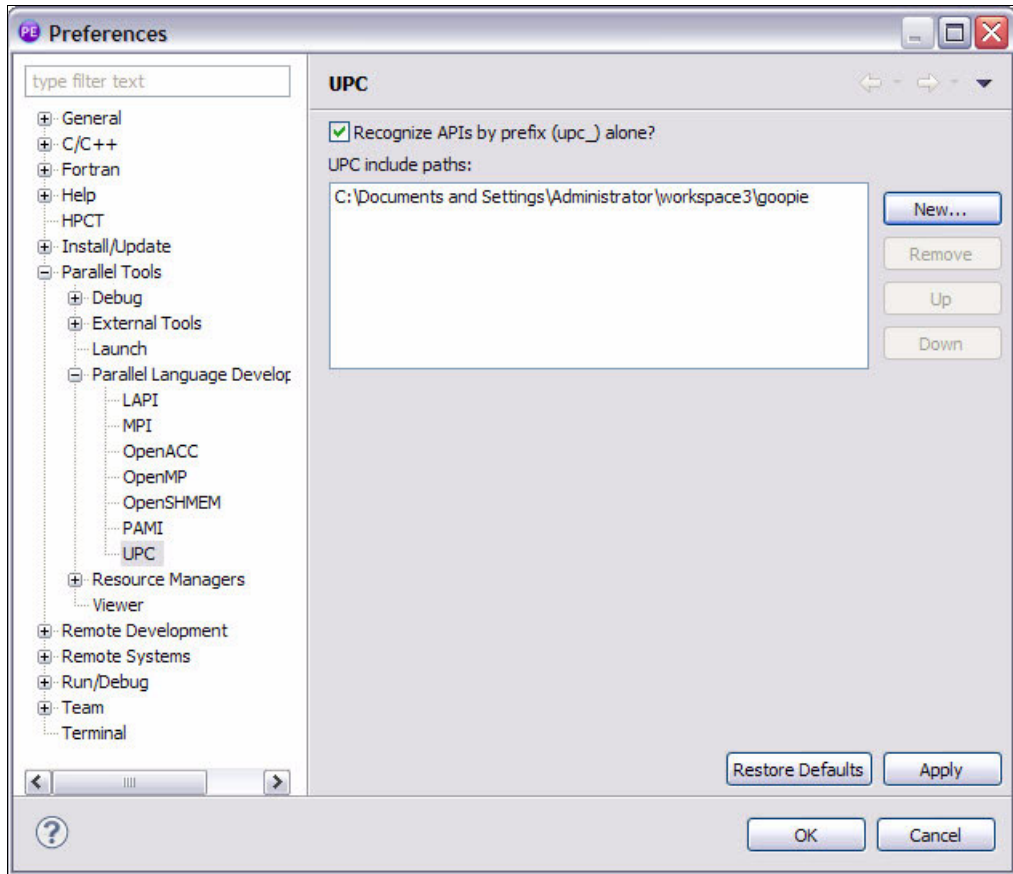


Figure 6-47 UPC artifacts configuration

If your project uses Fortran code and the parallel artifacts were still not recognized, change the configuration regarding how Eclipse handles the source form of your Fortran source files. Select your project on the Project Explorer view, select **File** → **Properties**, and choose **Fortran General** → **Source Form**. On this screen, change the source form for the *.F and *.f file extensions to Fixed Form - INCLUDE lines ignored, as shown in Figure 6-48 on page 137.

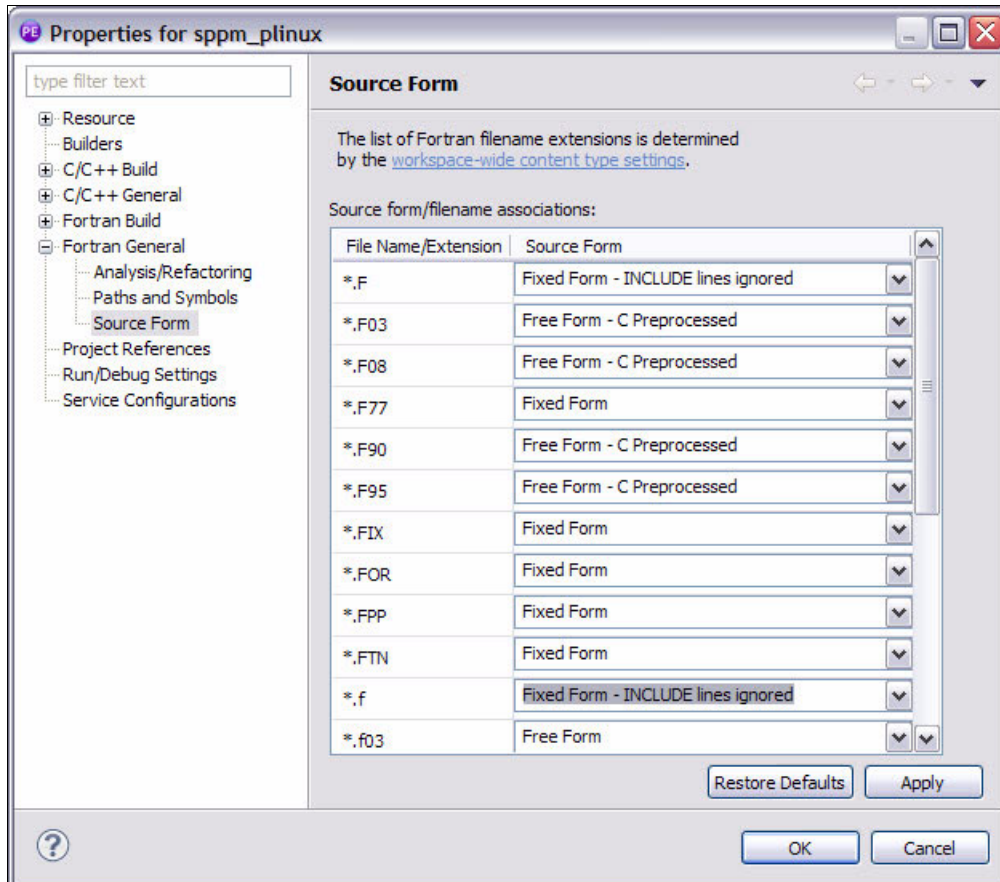


Figure 6-48 Fortran source form configuration

MPI barrier analysis

This tool can generate statistics about MPI artifacts and also assist with identify barrier problems while implementing a parallel application. The tools makes the following analysis across multiple source files:

- ▶ Potential deadlocks
- ▶ Barrier matches
- ▶ Barrier synchronization errors

Figure 6-49 shows the barrier matches report that assist to browse through the MPI barriers in your source code.

Barrier Matching Set	Function	Filename	LineNo	IndexNo
Barrier 10 (1)	run	run.c	91	10
Barrier 1 (1)	density	density.c	218	1
Barrier 9 (1)	restart	restart.c	282	9
Barrier 9	restart	restart.c	282	9
Barrier 3 (1)	hydro_force	hydra.c	239	3
Barrier 2 (1)	gravity_tree	gravtree.c	221	2
Barrier 5 (1)	read_jc	read_jc.c	72	5
Barrier 6 (1)	read_jc	read_jc.c	104	6
Barrier 7 (1)	read_jc	read_jc.c	151	7
Barrier 8 (0)	read_jc	read_jc.c	157	8

Figure 6-49 MPI analysis: barrier matches report

If any barrier problem is found during the analysis, a message is displayed, as shown in Figure 6-50. The MPI Barrier Errors view is opened, displaying the barrier errors report shown in Figure 6-51. This view can also be used to easily find the line on the source code where the problems was found.

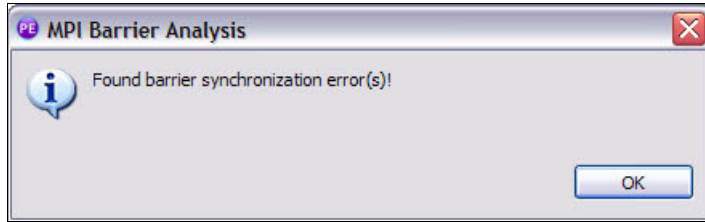


Figure 6-50 MPI barrier error found

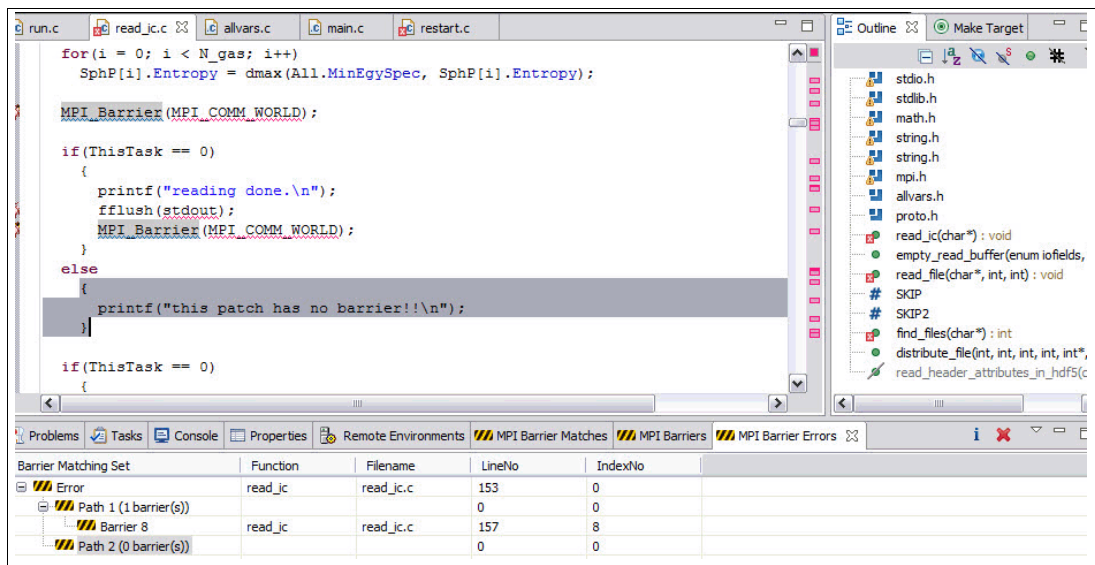


Figure 6-51 MPI analysis: barrier error report

6.2.2 Eclipse PTP Parallel debugger

This section describes basic procedures to use the eclipse built-in parallel debugger and state differences to the single process (or thread) one. For further details regarding this topic, we suggest you consult the PTP parallel debugger help, accessible through Eclipse menu bar (**Help** → **Help Contents** → **Parallel Tools Platform (PTP) User Guide** → **Parallel Debugging**).

The parallel debugger provides some specific debugging features for parallel applications that distinguish it from Eclipse debugger for serial applications. In particular, it is designed to threat parallel application as a set of processes, allowing a group to:

- ▶ Visualize their relationships with jobs
- ▶ Enable their management
- ▶ Apply common debugging operations

Debugging is still based on the breakpoint concept, but here it provides a special type known as a *parallel breakpoint*, also designed to operate in a set rather than a single process (or thread). There are two types of parallel breakpoints:

- ▶ Global breakpoints: Apply to all processes in any job
- ▶ Local breakpoints: Apply only to a specific set of processes for a single job

The current instruction pointer is also particular for parallel applications in the sense that:

- ▶ It shows one instruction pointer for every group of processes at the same location.
- ▶ The group of processes represented by an instruction pointer is not necessarily the same as a process set; therefore, different markers are used to indicate the types of processes stopped at a given location.

The parallel debugger relies on a server-side agent called Scalable Debug Manager (SDM) that is in charge of controlling the debug session. You need to properly set its path at the Debugger tab in the new debug launcher configuration window (Figure 6-52).

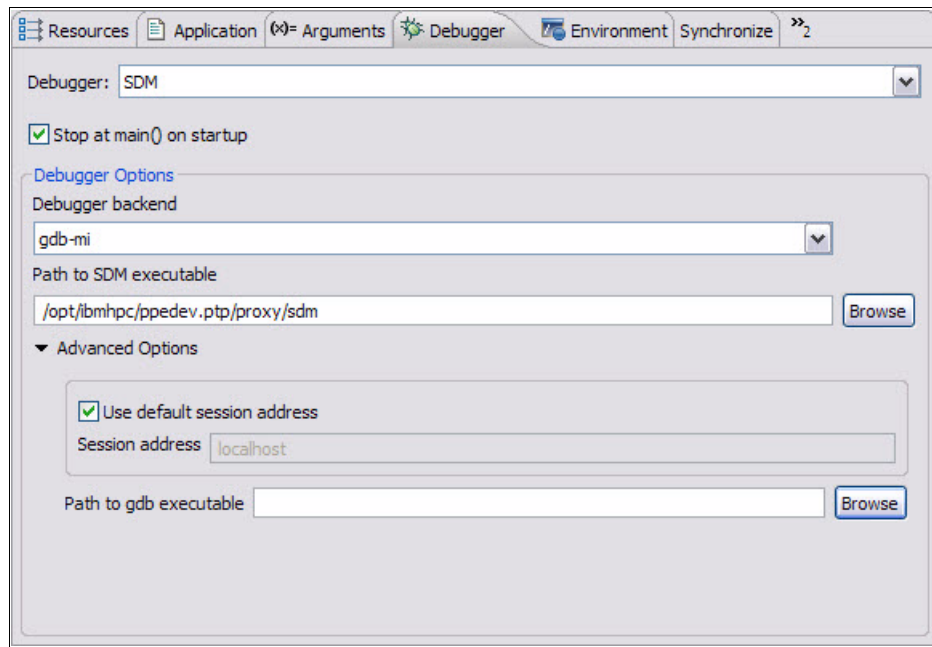


Figure 6-52 Debug launcher configuration: Debugger tab

Important: Notice in Figure 6-52 that you must set SDM path to `/opt/ibmhpc/ppdev.ptp/proxy/sdm`, which is the default location where the `ppdev_ptp_rte` package installed it. The `gdb` executable path is optional and the debugger selects it from the system `PATH` if it is not set.



Application profiling and tuning

In this chapter, we use some of the HPC applications to illustrate how to employ the PEDE and HPCT tools for profiling and tuning.

Usually people try to locate the hotspots of an application and then optimize it. There are some points that are usually considered for an HPC application profiling and tuning, such as the synchronization, load balance, communication overhead and the memory, file I/O issues, along with the real computation effort. Consequently to work on an existing HPC application for optimization, the first step is profiling to recognize the hot spot, Xprofile, or gprof to provide for this function. Then other tools for different profiling and tuning purpose can be utilized, such as the HPCT from IBM.

A hyper HPC program sPPM [S. Anderson et al], which is written by C and Fortran utilizing MPI and OpenMP for parallel programming is described. The contents describe the procedure to locate the hotspot in this program and analyze the possible reason based on the profiling data, followed by possible tuning hints.

The following topics are discussed in this chapter:

- ▶ Profiling and tuning hints for sPPM
- ▶ Profiling and analyzing Gadget 2
- ▶ Analyzing a ScaLAPACK routine using the HPCT Toolkit

7.1 Profiling and tuning hints for sPPM

This section introduces tuning for sPPM.

7.1.1 A glance at the application sPPM

Look for information about the sPPM application at:

<http://www.lcse.umn.edu/research/sppm/README.html>

The experiment for sPPM occurs based on Power Linux. This program computes a 3-D hydrodynamics problem on a uniform mesh using a simplified version of the PPM (Piecewise Parabolic Method) code.

The coordinates are -1:1 in x and y, and 0:zmax in z, where zmax depends on the overall aspect ratio prescribed. A plane shock traveling up the +z axis encounters a density discontinuity, at which the gas becomes denser. The shock is carefully designed to be simple, but strong, about Mach 5. The gas initially has a density of 0.1 ahead of the shock. At over 5dz at the discontinuity, it changes to 1.0.

This version of sPPM is implemented primarily in Fortran with system-dependent calls in C. All message passing is done with MPI. Only a small set of MPI routines are actually required:

- ▶ MPI_Init, MPI_Comm_rank, MPI_Comm_size, MPI_Finalize
- ▶ MPI_Wtime, MPI_IRecv, MPI_Isend
- ▶ MPI_Wait, MPI_Allreduce, MPI_Request_free

The parallelization strategy within an SMP is based on spawning extra copies of the parent MPI process. This is done with the *sproc* routine, which is similar but not identical to the standard UNIX *fork* call. It differs in that *sproc* causes all static (for example, not stack based) storage to be implicitly shared between processes on a single SMP node. This storage is always declared in Fortran with either SAVE or COMMON.

Synchronization is accomplished by examining shared variables and by use of UNIX System V semaphore calls. (See *twiddle.m4* and *cstuff.c*.) In addition, a fast-vendor supplied implementation of an atomic fetch and add (*test_and_add*) is used.

7.1.2 Use gprof to view the profiling data

As described in advance, various approaches can be applied to get different types of profiling data. To get the cpu time profiling data of a program, we must compile the source code with **-p -pg**. After execution, for each MPI task, a *gmon.out* file is generated, making it possible to use *gprof* to generate data issuing the command:

```
-bash-4.1$ gprof ../sppm gmon.out > prof.txt
```

In this command:

- ▶ *sppm* is the executable
- ▶ *gmon.out* is the raw profiling data
- ▶ the *prof.txt* contains the readable profiling data

Specifically for the *sppm* application, the output is shown in Figure 7-1 on page 143.

Flat profile:

Each sample counts as 0.01 seconds.

%	cumulative	self	calls	self	total	name
time	seconds	seconds		s/call	s/call	
46.30	354.85	354.85	13371314	0.00	0.00	.sppm
24.70	544.15	189.30	13376282	0.00	0.00	.difuze
10.16	621.99	77.84	66859566	0.00	0.00	.interf
6.31	670.36	48.37				.__vrec_P7
3.61	698.03	27.67	26752519	0.00	0.00	.dintrf
1.72	711.24	13.21	161	0.08	0.77	.hydyz
1.65	723.89	12.65	153	0.08	0.81	.hydzy
1.06	732.01	8.12	168	0.05	0.71	.hydxy
0.91	738.96	6.95	158	0.04	0.74	.hydyx
0.66	744.01	5.05	178	0.03	0.65	.hydzz
0.57	748.35	4.34				.__vsqrt_P7
0.56	752.64	4.29	396	0.01	0.01	.bdrys1s
0.42	755.86	3.22				.__vrsqrt_P7
0.31	758.27	2.41	397	0.01	0.01	.bdrys1r
0.18	759.62	1.35	201	0.01	0.01	.bdrys1as
0.13	760.63	1.01	192	0.01	0.01	.bdrys1bs
0.11	761.49	0.86	397	0.00	0.00	.bdrys2s
0.11	762.35	0.86	175	0.00	0.63	.hydxx
0.08	762.97	0.62	398	0.00	0.00	.bdrys2r
0.08	763.58	0.61	197	0.00	0.00	.bdryslar
0.07	764.13	0.55	398	0.00	0.00	.bdrys3s
0.05	764.49	0.36	190	0.00	0.00	.bdrys2fs
0.04	764.76	0.27	398	0.00	0.00	.bdry2o1s
0.03	765.02	0.26	198	0.00	0.00	.bdrys1br
0.03	765.27	0.25	199	0.00	0.00	.bdrys2fr
0.02	765.44	0.17	188	0.00	0.00	.bdrys3fs
0.02	765.60	0.16	1	0.16	0.16	.initbuf
0.02	765.75	0.15	399	0.00	0.00	.bdry2o1r
0.02	765.89	0.14	396	0.00	0.00	.bdrys3r

Figure 7-1 The CPU time profiling for sPPM

So from the flat profiling result, the function *sppm*, *difuze* and *interf* took over 80% of the time. Besides, these functions are invoked millions of times, so it is useful if we do optimization for these functions. Because if any slight benefit is brought in, it can be multiplied with millions due to the amount of calls. However it is possible that it is hard to get these functions optimized because for each single call, it only takes few times, which means that these function can be highly optimized already. For the function *_vrec_P7*, it is impossible to do anything with it because it is a built-in function of the compiler library. Thus it is still worthwhile to look at the functions named as *hydyz*, *hydzy*, *hydxy*, *dydyx*, *hydzz*. So let us look further into the call graph profiling data. We describe the details for each function call. See Figure 7-2 on page 144.

[3]	84.8	354.85	294.81	13371314	.sppm [3]
		189.30	0.00	13376282/13376282	.difuze [4]
		77.84	0.00	66859566/66859566	.interf [11]
		27.67	0.00	26752519/26752519	.dintrf [13]

[4]	24.7	189.30	0.00	13376282/13376282	.sppm [3]
		189.30	0.00	13376282	.difuze [4]

[11]	10.2	77.84	0.00	66859566	.interf [11]
		12.65	111.28	153/153	.runhyd [2]

Figure 7-2 Detailed call graph

Based on the call graph, we can also ascertain that difuze and interf are called by sppm, and it does not have any further children to call. So it is a good choice to start to optimize the code of difuze and interf. Also some other functions must be noticed. Figure 7-3 gives two of the examples.

[6]	16.2	13.21	110.64	161/161	.runhyd [2]
		13.21	110.64	161	.hydyz [6]
		59.10	49.10	2226877/13371314	.sppm [3]
		1.07	0.00	99/396	.bdrys1s [15]
		0.59	0.00	98/397	.bdrys1r [17]
		0.22	0.00	100/397	.bdrys2s [20]
		0.17	0.00	91/190	.bdrys2fs [24]
		0.15	0.00	99/398	.bdrys2r [21]
		0.12	0.00	99/199	.bdrys2fr [27]
		0.07	0.00	99/398	.bdry2o1s [25]
		0.04	0.00	100/399	.bdry2o1r [30]
		0.00	0.00	99/398	.bdry3o2s [36]
		0.00	0.00	99/595	.flag_set [41]
		0.00	0.00	99/2191	.deltat [38]
		0.00	0.00	99/99	.bdrys0yz [49]
		0.00	0.00	98/396	.bdry3o2r [43]

[7]	15.5	8.12	110.93	168/168	.runhyd [2]
		8.12	110.93	168	.hydxy [7]
		59.14	49.13	2228418/13371314	.sppm [3]
		1.07	0.00	99/396	.bdrys1s [15]
		0.61	0.00	100/397	.bdrys1r [17]
		0.50	0.00	95/192	.bdrys1bs [19]
		0.14	0.00	98/398	.bdrys3s [23]
		0.13	0.00	99/198	.bdrys1br [26]
		0.07	0.00	200/394	.bdry3o1s [31]
		0.07	0.00	100/398	.bdry2o1s [25]
		0.04	0.00	99/399	.bdry2o1r [30]
		0.04	0.00	100/396	.bdrys3r [33]
		0.00	0.00	199/396	.bdry3o1r [42]
		0.00	0.00	100/595	.flag_set [41]
		0.00	0.00	99/396	.bdry3o2r [43]
		0.00	0.00	99/2191	.deltat [38]
		0.00	0.00	99/99	.bdrys0xy [47]

Figure 7-3 Detailed call graph for hydyz and dydxy

In addition, tools can help to figure out what is the potential problem in some of the hotspots if it is hard to sort out by inspection of the code. Usually the hotspots are either caused by heavy computation with lots of “load and store” instructions or the communication took plenty of time along with load balance issues. Hence in the later sections, possible causes of the hotspots are analyzed.

7.1.3 Using binary instrumentation for MPI profiling

We presented the general steps about binary instrumentation in the previous chapters. However since it is desired to profile all MPI communications for the program, we choose all of the source code file in the instrumentation view, as shown in Figure 7-4.

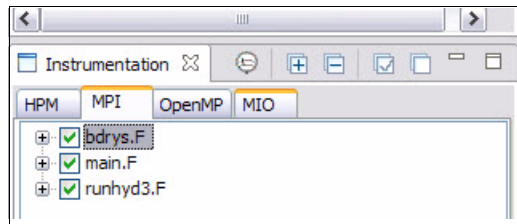


Figure 7-4 Instrumentation panel

After the instrumented binary is generated, the default profiling setting is sufficient for information that we need to generate trace files only for the application tasks with the minimum, maximum, and median MPI communication time. This is also true for task zero (people might use task), if task zero is not the task with minimum, maximum, or median MPI communication time. If you need trace files generated for additional tasks, make sure the necessary options under the HPC Toolkit menu are checked during the profiling configuration phase. Refer to Chapter 6, “Parallel Environment Developer Edition tools” on page 95 for the details of each options.

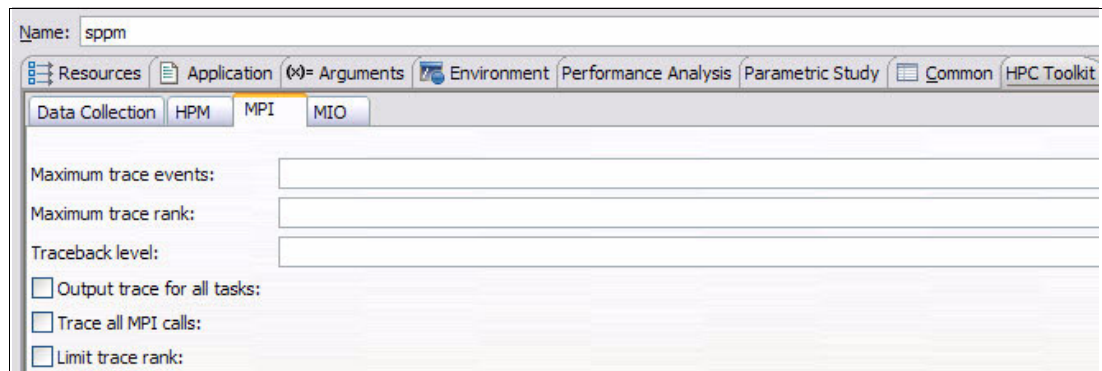


Figure 7-5 Options for MPI profiling

Depending on the number of tasks in your application, make sure that the Maximum trace rank and Limit trace rank are set correctly. If your application executes many MPI function calls, you might need to set the value of Maximum trace events to a higher number than the default 30,000 MPI function calls. Refer to Figure 7-5.

Chapter 6, “Parallel Environment Developer Edition tools” on page 95 explains how to do the launching configuration with an application; therefore, using the same setting, just click **Profile**, and the program runs correctly. Synchronize your project for the generated profiling data. You can now open them with HPCT after you right-click the files. The elapsed cpu time for the MPI function calls is displayed in the summary section. Under that, notice the details of

each MPI function call in each source code file and further down to each function that invoked MPI routines. For sPPM, we got the summary data in Figure 7-6.

Performance Data Summary Console Synchronized Merge View De

Data for rank 3, Aggregation for tasks: 3

Label	Count	WallClock	Transferred Bytes
[-] SUMMARY			
MPI_Comm_size	2	0.000004	0.000000
MPI_Comm_rank	2	0.000014	0.000000
MPI_Barrier	1	0.001074	0.000000
MPI_Irecv	3808	0.004286	6704680000.000000
MPI_Isend	3804	0.055717	6702340000.000000
MPI_Wait	7608	98.536649	0.000000
[-] main.F			
[-] main(main.F)			
MPI_Comm_size_207	1	0.000004	0.000000
MPI_Comm_rank_206	1	0.000012	0.000000
MPI_Barrier_201	1	0.001074	0.000000
[-] layout(main.F)			
MPI_Comm_size_1885	1	0.000000	0.000000
MPI_Comm_rank_188	1	0.000002	0.000000
[-] bdrys.F			
[-] hdrv3n2s(hdrv3.F)			

Figure 7-6 sPPM mpi profiling data

We can see the most time consumable MPI routine is MPI_Wait, technically each asynchronous MPI_Isend and MPI_Irecv might have one MPI_Wait, so the column of Count illustrates the MPI_Wait has the most frequent invoking, so the wall clock is the highest one. However the communication overhead might not a big issue in this project if comparing with the entire execution time (near 10 minutes). It is still interesting that based on the result shows here, the assumption is the work load on each MPI tasks is not well balanced; otherwise, the MPI_Wait routine only takes a short time. If we look at the MPI trace picture, which can be generated by opening the .mpt file through the mpi trace menu in Figure 7-7 on page 147, we can see that the MPI_wait waits for a longer time with mpi rank 4, 5, 6, 7, which means these rank can get less workload than rank 0, 1, 2, 3. Despite that workload issue, the communication happened many times, it is possible to use depth halo with more boundary data layers to reduce the communication frequency. Another point is it is worthwhile to think about if there is a possible way to reduce the total transferred data size, which can be done with altering the major algorithm.

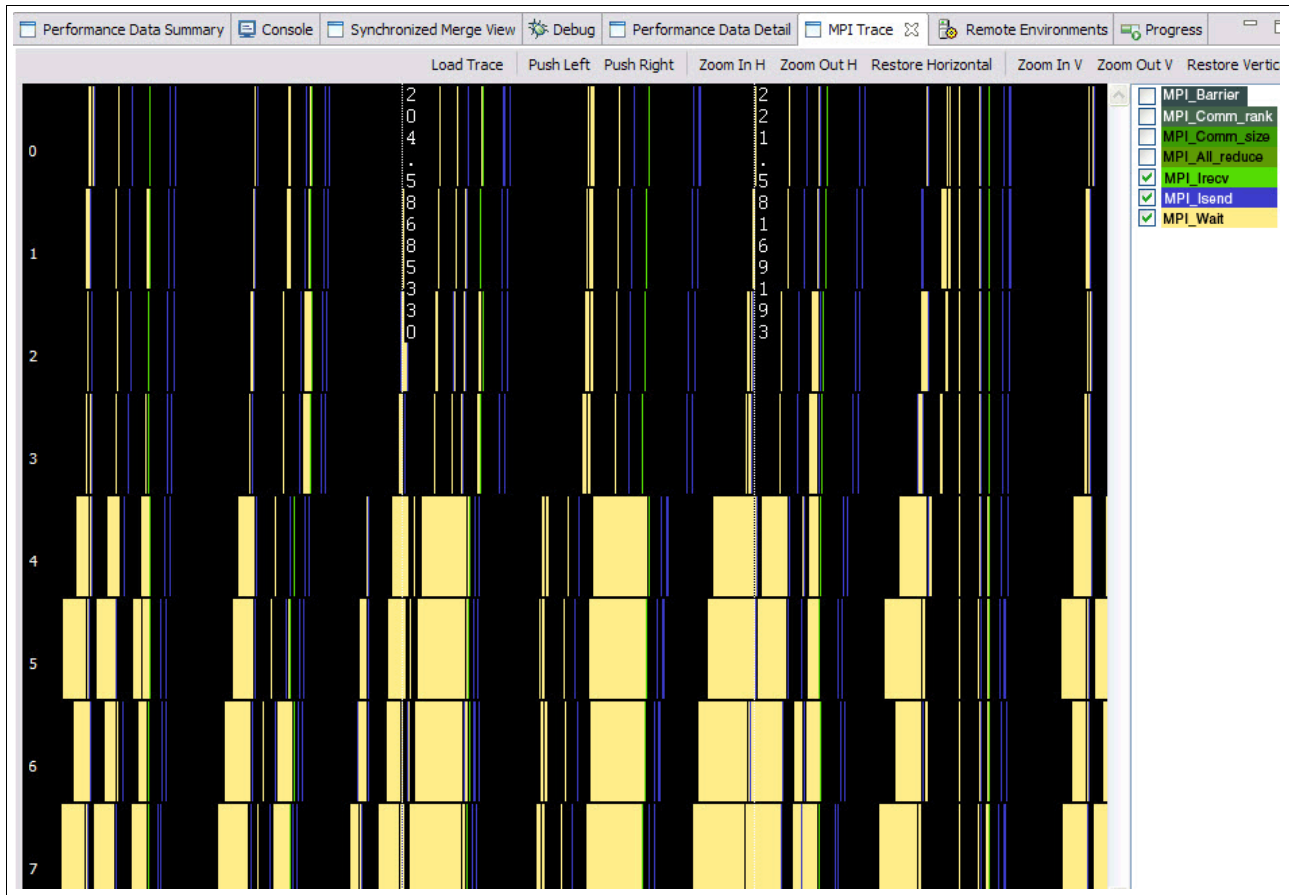


Figure 7-7 MPI trace picture for sPPM

7.1.4 Use OpenMP profiling to identify if the workload for threads

Because the sPPM code utilizes OpenMP threads for some of the hotspots, we can also profile it to see if there is any load balance issue. Refer to “OpenMP profiling” on page 115 for details about how to get your program instrumented for OpenMP profiling. Here, we choose to profile all of the OpenMP regions listed in Figure 7-8 on page 148. Because of the limitation that it is not supported to profile nested OpenMP parallel region, some of the OpenMP parallel regions cannot get profiling data.

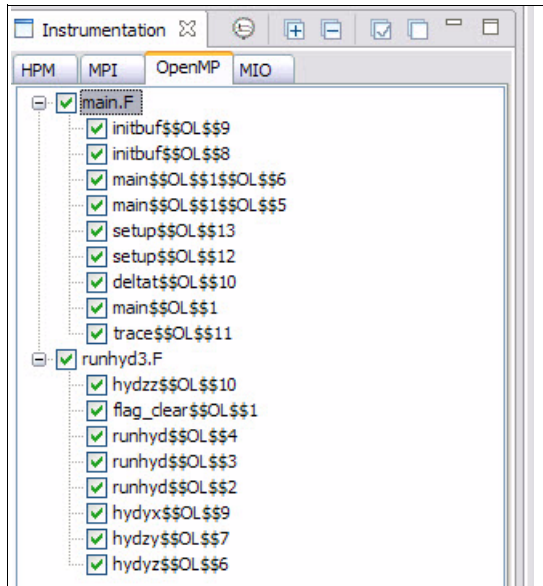


Figure 7-8 Instrument all the OpenMP parallel regions for sPPM

After the execution of the instrumented binary, the OpenMP profiling data was generated. Open the .viz files. The summary data is shown in Figure 7-9.

Label	Count	Excl. Time	Indl. Time	%Total Overhead	%Imbalance	Avg. Thread Time
Data for rank 0, Aggregation for tasks: 0 1 2 3						
main.F						
loop_1778	1	0.028678	0.028678	0.226131	0.253114	0.028577
loop_1534	1	0.007871	0.007871	1.266168	0.000000	0.007776
preigion_383	1	130.283865	379.702500	0.153721	1.028447	377.189147
loop_970	1	0.022916	0.022916	0.846893	0.000000	0.022911
loop_1823	1	0.027115	0.027115	0.327096	0.000000	0.027229
loop_905	1	0.039406	0.039406	0.265003	0.804144	0.039145

Figure 7-9 OpenMP profiling data

Double-click the most time consumable region. The tool navigates you to which line the code is in the source file. So it is easily located where the OMP issue might be. By further inspecting the code, in this parallel region we have many single and master directives that made the code fragment in serial. So to get better performance, it is necessary to think about how to get those directives less. However based on Figure 7-8 and Figure 7-9, notice that there is no profiling data for the runhyd3.F source code because there are nested OMP regions. As mentioned before, under this circumstance, HPCT does not support OpenMP profiling currently.

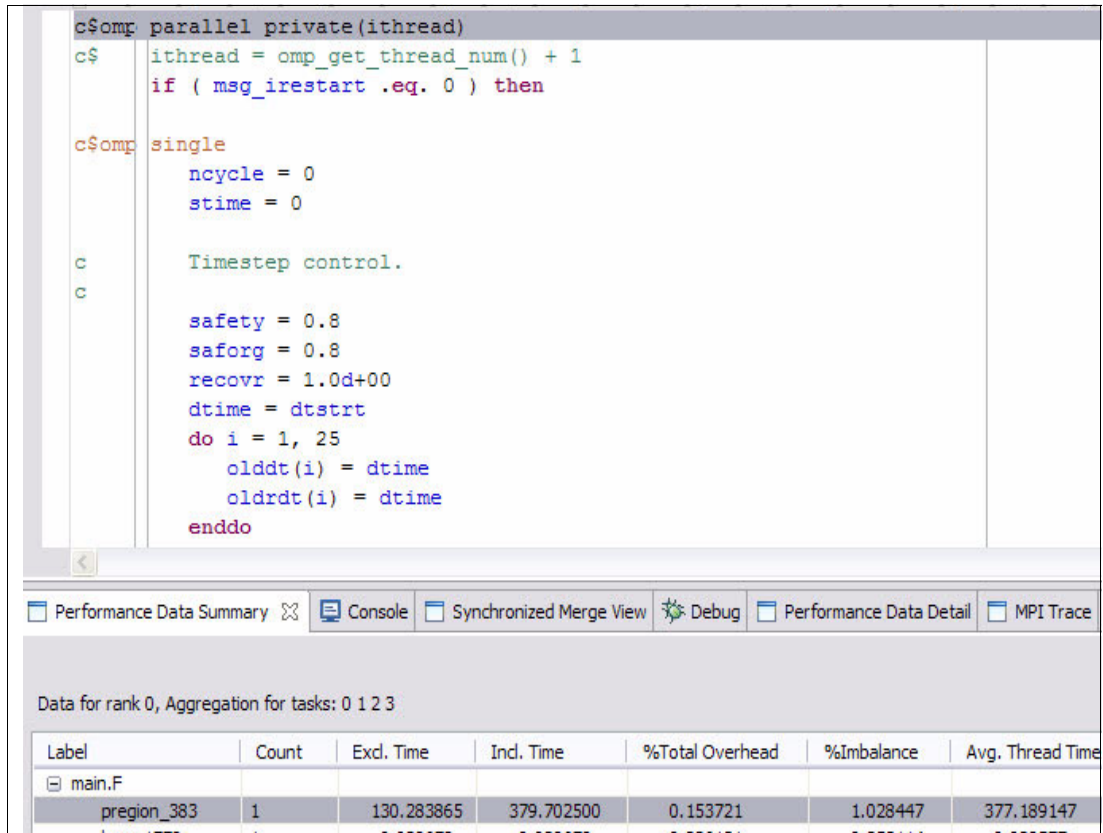


Figure 7-10 hotspot in OpenMP code

In a nut shell, the HPC Toolkit, which is part of the Parallel Environment Developer Edition, can generate profiling data for HPC application developers quickly (Figure 7-10). Based on this profiling data, it is easier to get an idea about how to optimize an existing HPC application without understanding some scientific theory. However there are some other good tools in HPCT with PEDE that are not mentioned here, for example, the I/O profiling, due to sPPM, it does not have heavy I/O operations.

7.2 Profiling and analyzing Gadget 2

Gadget 2 is a freely available application for cosmological simulations widely used in the HPC area, which can be used to address several problems, such as colliding and merging galaxies, dynamics of the gaseous intergalactic medium, or to address star formation and its regulation by feedback processes. Further information about the project along with source code is at:

<http://www.mpa-garching.mpg.de/gadget/>

This study case takes Gadget2 to:

- ▶ Show how to configure a project that requires special build set up.
- ▶ Show how to use some basic Eclipse editor features to browse the source code.
- ▶ Show how the tools drill-down the application source code looking for performance enhancements opportunities.

Importing Gadget 2 as an existing Makefile application

The Gadget 2 source code was downloaded and uncompressed in the remote Linux on Power cluster node. Next, it was imported in the PEDE Eclipse environment as a synchronized project (see Figure 7-11). The rationale behind our choices are:

- ▶ Although the majority of the Gadget's source code is written in C, we chose to import it as a Synchronized Fortran project because it also contains a small portion of Fortran code and by doing that we ensured that Eclipse handles both Fortran and C/C++ languages properly.
- ▶ We imported the project using the empty Makefile project template because Gadget relies on the GNU Makefile build system.
- ▶ In one of the steps to import the project, you need to set the connection to the remote node holding the application source code. We managed to create a new connection since there was not one for the PowerLinux node (see Figure 7-12 on page 151).
- ▶ We chose **--Other Toolchain--** from the Local Toolchain field because PEDE Eclipse was running in a Windows XP notebook, so there was no use in choosing a local toolchain.

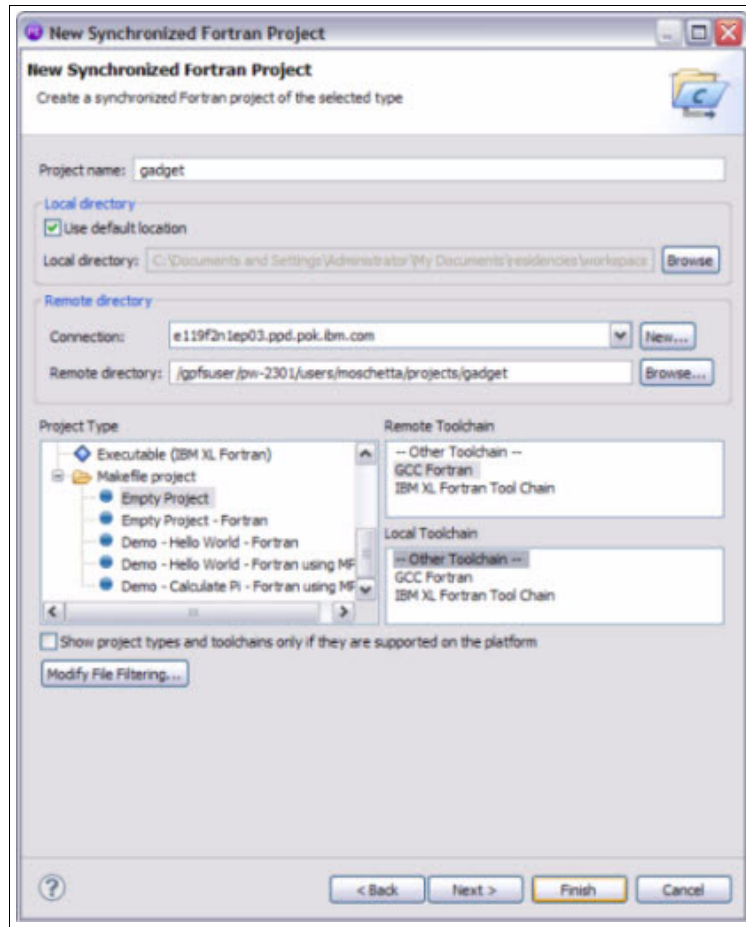


Figure 7-11 Creating a synchronized project of Gadget 2

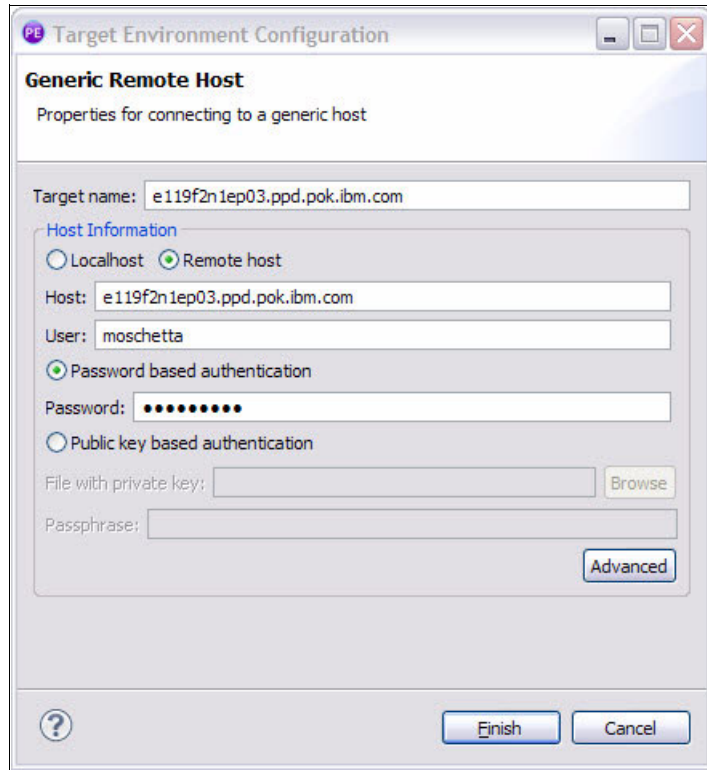


Figure 7-12 Creating a connection with remote PowerLinux cluster node

Configured build settings

The Gadget 2 source code comes from some examples of simulations that are built by different makefiles located in the Gadget2 folder (see Figure 7-13). We chose the Cosmological formation of a cluster of galaxies example for this section, which is built by the Makefile.cluster makefile. So it was needed to change the default settings to properly build the project:

1. Change the default build directory to Gadget2 in the gadget project properties (see Figure 7-14 on page 152).
2. Create a new make target that instructs Eclipse to run `make -f Makefile.cluster` from the build directory (see Figure 7-15 on page 152). For further information about management of makefile targets, consult the Eclipse help in the menu bar: **Help** → **Help Contents** → **C/C++ Development User Guide** → **Tasks** → **Building projects** → **Creating a Make target**.

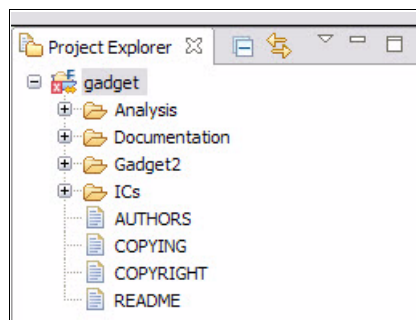


Figure 7-13 Gadget 2 folders hierarchy

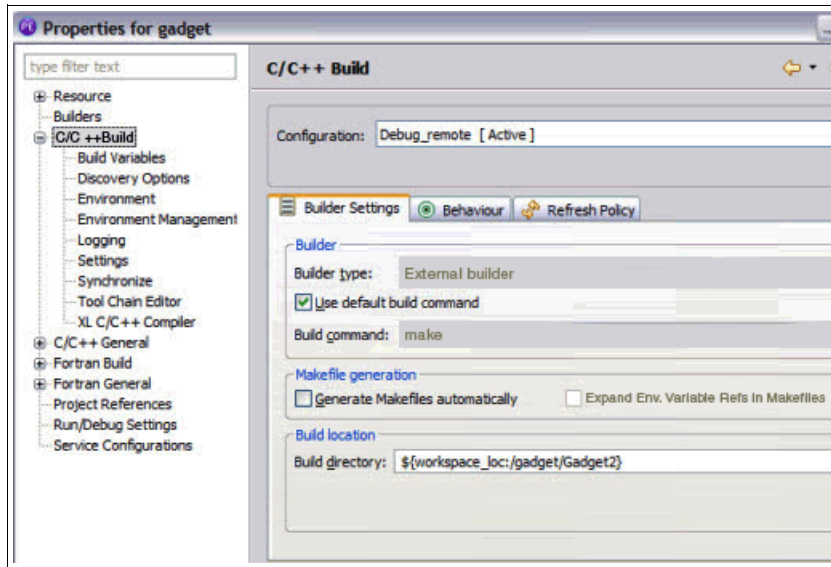


Figure 7-14 Changing default build directory

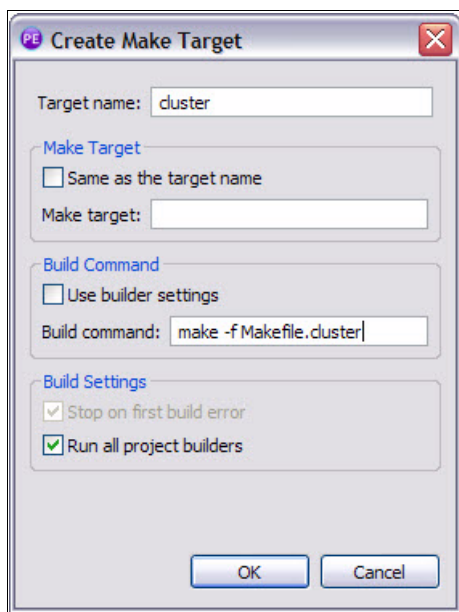


Figure 7-15 Creating a new make target

The project is now built using the cluster target instead of default all. In Figure 7-16 on page 153, the right-column box, Make Target, has an icon for the cluster target where we just double-click it to trigger a fresh build.

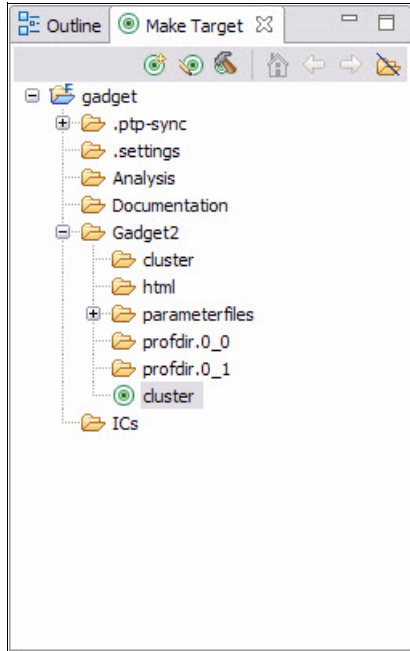


Figure 7-16 New created cluster make target

There is another configuration that many users do not realize is important regarding include paths and preprocessor symbols. If they are not properly set, some features, such as code search and completion, will not work perfectly because the Eclipse C/C++ parse does not understand application source code content completely (Figure 7-17).

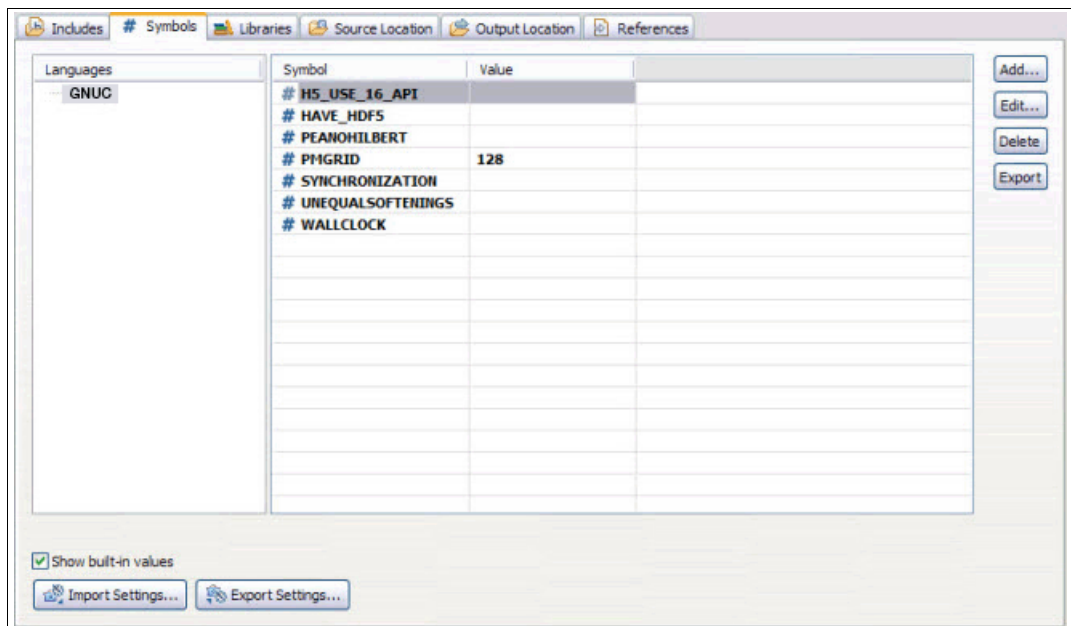


Figure 7-17 Gadget 2 preprocessor symbols

More details about how to set Include paths and preprocessor symbols are at the menu: **Help** → **Help Contents** → **C/C++ Development User Guide** → **Tasks** → **Building projects** → **Adding Include paths and symbols**.

Creating a run launcher configuration

The next task was to create a run launcher configuration, which often served as a template for other launch configurations, such as debugger and IBM HPC Toolkit tools.

Our newly created configuration leveraged IBM Parallel Environment (PE) as the execution environment, as shown in Figure 7-18. However, any of the several other execution environments were applicable, and integration is provided by PEDE.

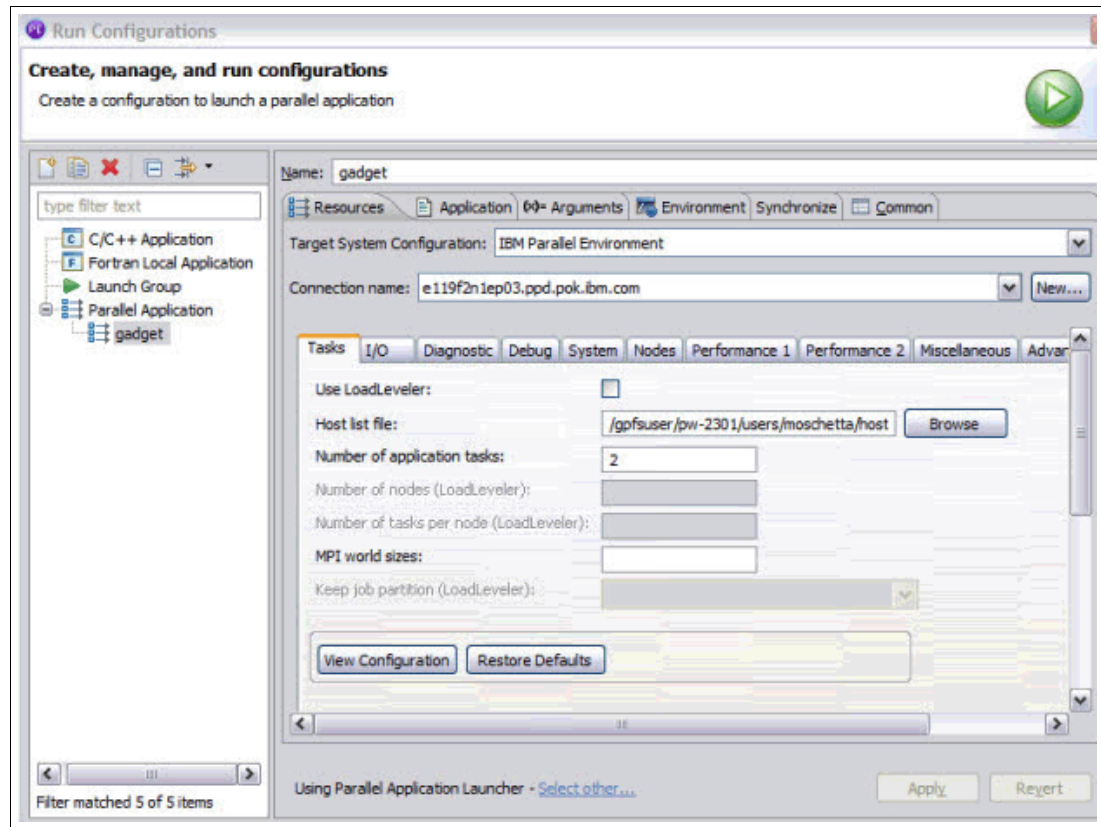


Figure 7-18 Set execution environment

Other tabs in the run launcher configuration were properly fulfilled:

- ▶ In the Application tab (Figure 7-19 on page 155): It set the gadget2 executable path.
- ▶ In the Arguments tab (Figure 7-20 on page 155): It set the input control file for the Gadget2 simulation as an argument of the executable. It also changed the default working directory because the input control file makes references to others relative to the Gadget2 folder.
- ▶ In the Environment tab (Figure 7-21 on page 156): It was set to a runtime variable that otherwise would fail gadget 2 run. For example, LD_LIBRARY_PATH was exported in runtime because the application depends on third-party shared library and not in a default path.

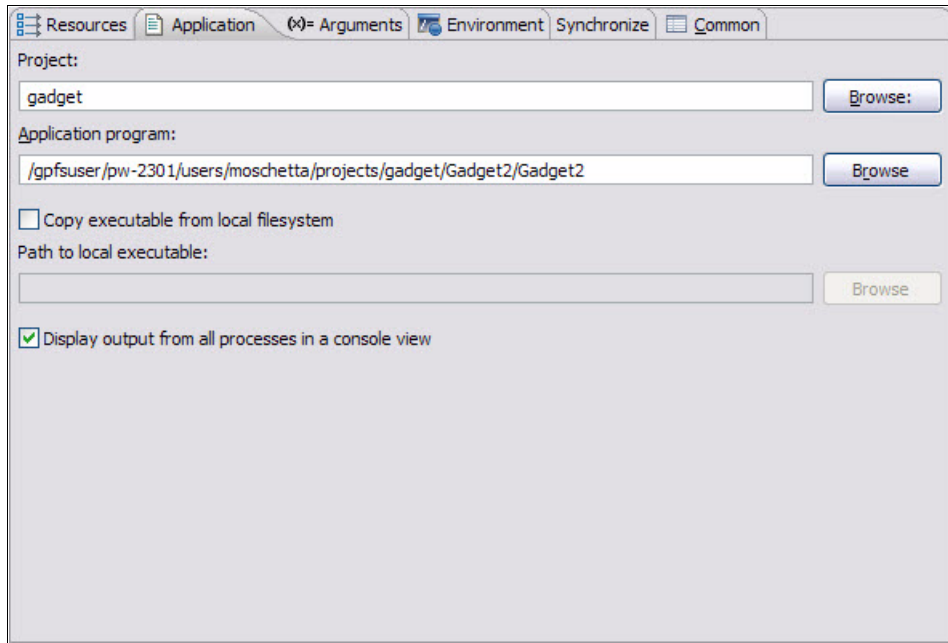


Figure 7-19 Run launcher configuration: Set application path

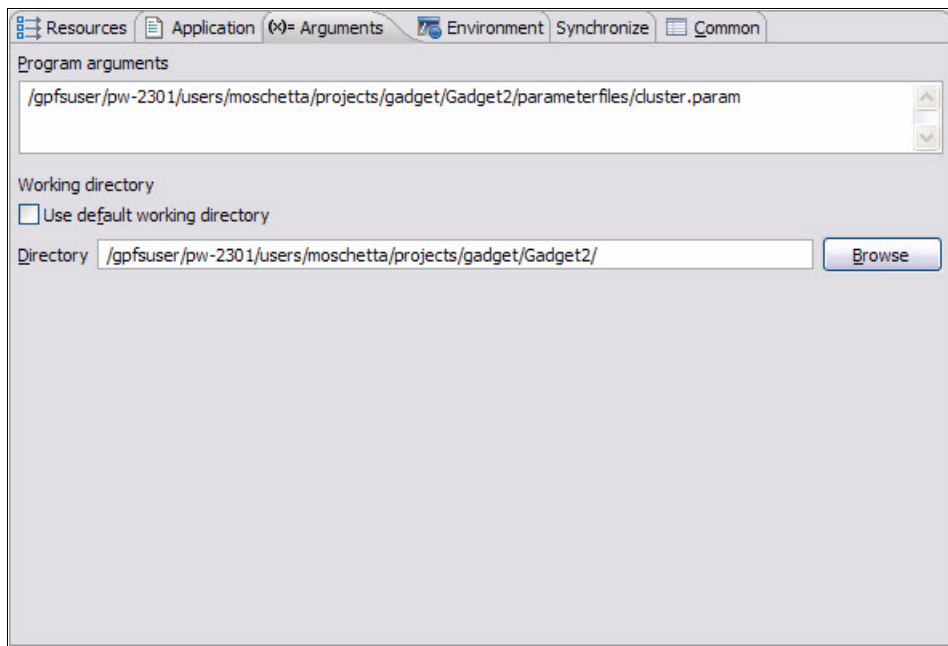


Figure 7-20 Run launcher configuration: Set gadget2 arguments

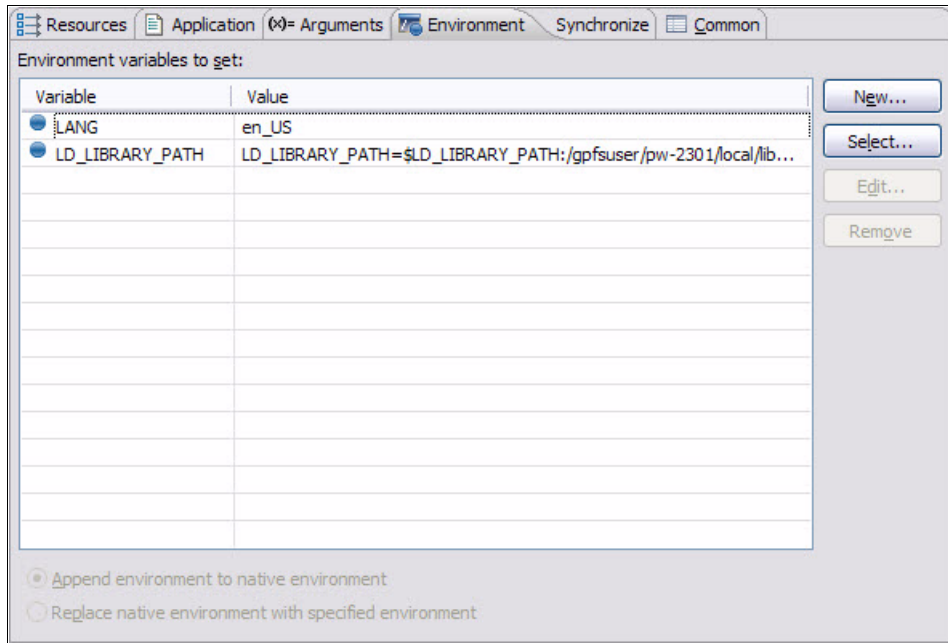


Figure 7-21 Run launcher configuration: Set environment variables

Searching for hot spots using Xprof

We chose to use the Xprof tool (refer to “X Windows Performance Profiler” on page 130) as the first tool toward performance analysis of Gadget 2 because it allows discovery tasks and methods that most impact overall execution time.

The application is built with the required **-pg** flag, executed with the previous mentioned run configuration to get gprof data files and then loaded into Xprof (see Figure 7-22 on page 157).

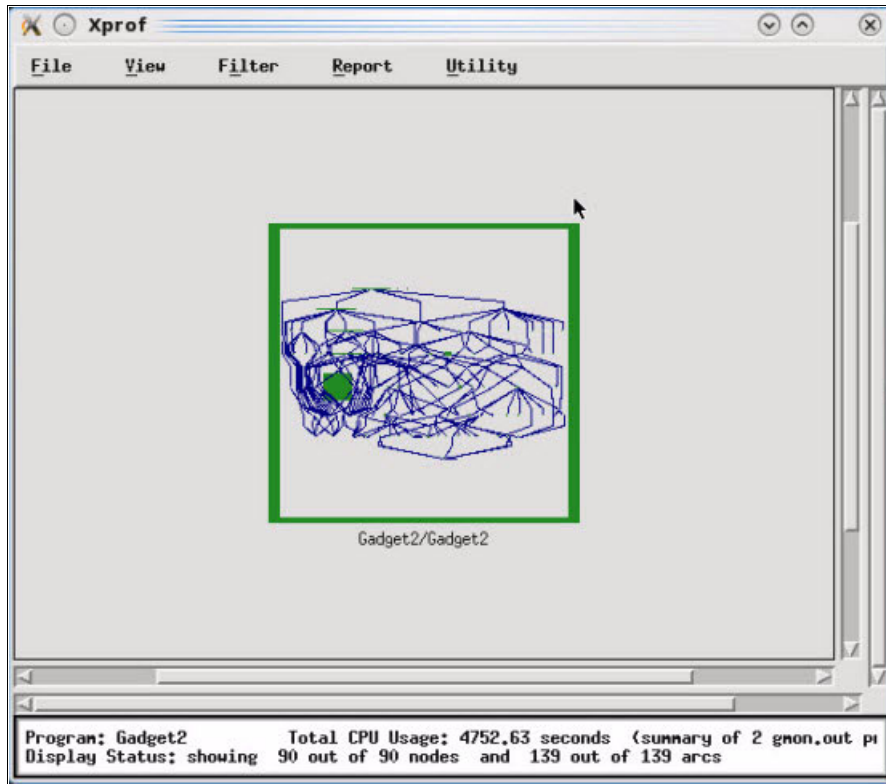


Figure 7-22 Xprof main view

Switching to flat report (menu **Report** → **Flat Profile**) enabled us to visualize a rank of methods with the most CPU-intensive on top. Figure 7-23 on page 158 shows that 69.6% of (cumulated) time is spent in a single method named `force_treeevaluate_shortrange`. The `force_treeevaluate_shortrange` seemed to be a good candidate for our investigation, but before taking a look at this code we preferred to seek more information.

Flat Profile						
File	Code Display	Utility				
%time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name
69.6	3308.14	3308.14	248753142	0.01	0.01	force_treeevaluate_shortrange [5]
6.9	3638.21	330.07	876	376.79	878.69	long_range_force [6]
5.8	3914.62	276.41	876	315.54	315.54	pn_setup_nonperiodic_kernel [9]
5.7	4183.18	268.56	727669434	0.00	0.00	peano_hilbert_key [10]
3.4	4345.55	162.37	1750	92.78	92.81	pnforce_nonperiodic [13]
1.4	4413.30	67.75	2630	25.76	87.40	force_treebuild_single [12]
0.8	4449.73	36.43	5260	6.93	14.12	domain_sunCost [16]
0.8	4485.61	35.88	2630	13.64	13.64	domain_countToGo [19]
0.8	4521.30	35.69	2628	13.58	13.58	domain_exchangeParticles [20]
0.7	4556.25	34.95	169520	0.21	0.21	domain_compare_toplist <cycle 1> [17]
0.6	4584.14	27.89	5260	5.30	5.30	force_insert_pseudo_particles [22]
0.5	4609.50	25.36	4682	5.42	761.08	gravity_tree [4]
0.5	4634.52	25.02	2630	9.51	9.51	peano_hilbert_order [23]
0.3	4649.51	14.99	4680	3.20	6.62	find_next_sync_point_and_drift [21]
0.3	4661.91	12.40	4680	2.65	9.13	advance_and_find_timesteps [18]
0.3	4674.27	12.36	74996	0.16	0.16	domain_compare_key <cycle 1> [24]
0.3	4686.44	12.17				compare_key [26]
0.3	4698.34	11.90	4690	2.54	3.33	move_particles [25]
0.2	4708.80	10.46	150112919	0.00	0.00	get_timestep [27]
0.2	4718.76	9.96	325679720	0.00	0.00	get_gravkick_factor [28]

Search Engine: (regular expressions supported)

Figure 7-23 Xprof flat report

After double-clicking **force_treeevaluate_shortrange** in the flat report, Xprof pointed out the major rectangle in the bottom left of Figure 7-22 on page 157. Graphically, its dimensions represent two measurements:

- ▶ The width represents how many times a method was called by its parents.
- ▶ The height represents the amount of execution time a method spent in its own code, for example, excluding time spent in its children.

To zoom in the call graph, we applied a filter (menu **Filter** → **uncluster Function**) that resulted in Figure 7-24 on page 159.

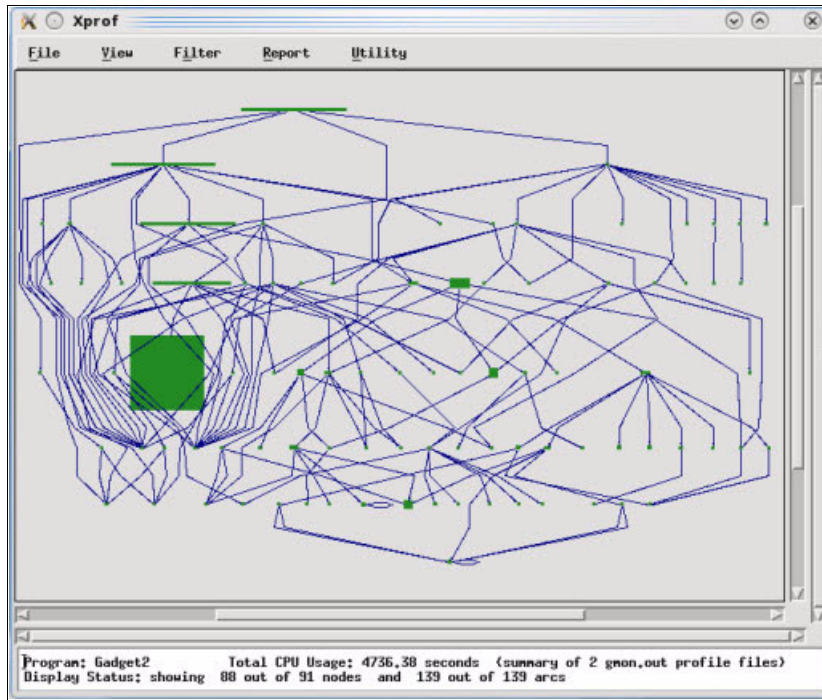


Figure 7-24 Xprof main view with uncluster functions

The `force_treeevaluate_shortrange`'s rectangle stands out on both width and height proportions. Moreover, Figure 7-25 shows that the total amount of seconds in the self and self+desc fields are equal; therefore, none of the execution time was spent in children methods.

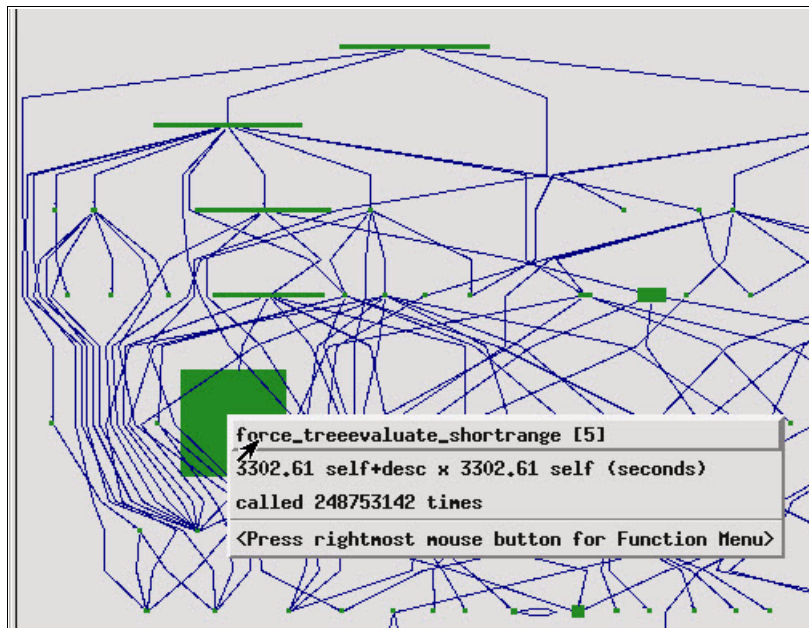


Figure 7-25 `force_treeevaluate_shortrange`: Self+desc versus self time

In fact, `force_treeevaluate_shortrange` was an obvious hot spot, but it was also needed to investigate its parent's methods. If you right-click the rectangle, and select menu options like in Figure 7-26 on page 160, a chain of parents is displayed. The

force_treeevaluate_shortrange method has only one parent method, as shown in Figure 7-27 on page 161.

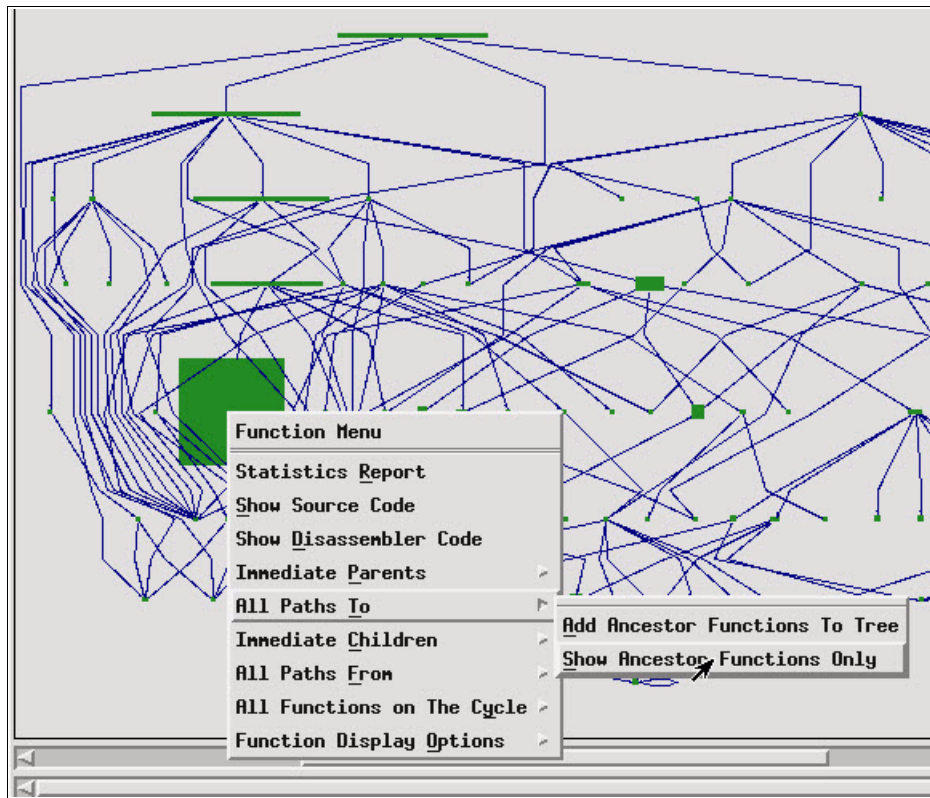


Figure 7-26 Show ancestor functions only

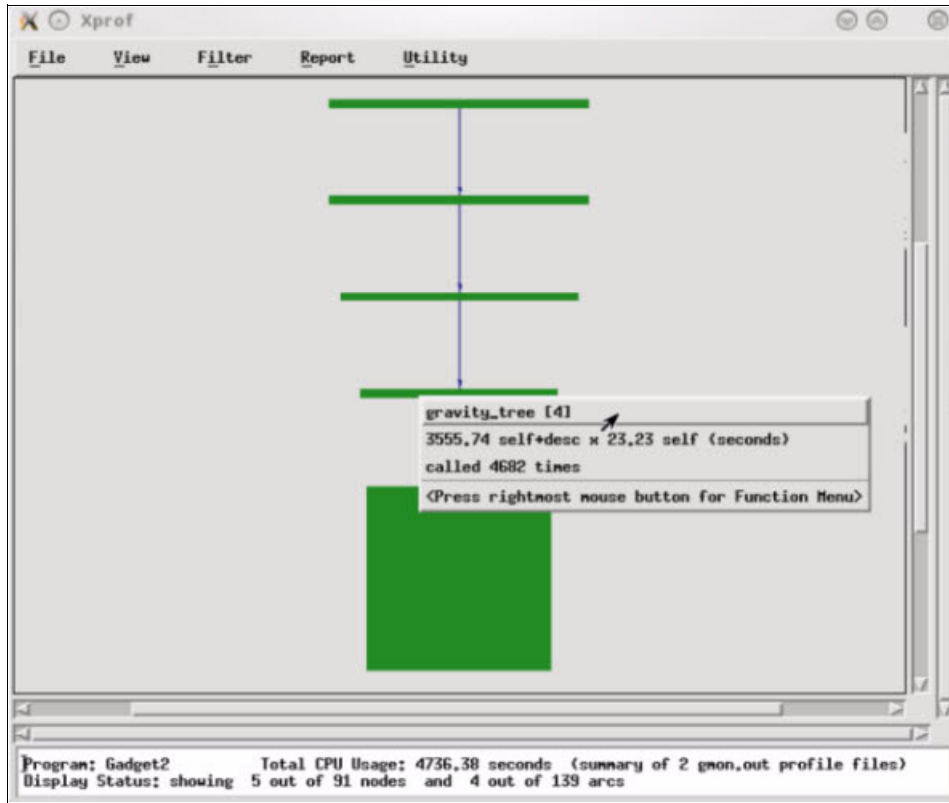


Figure 7-27 force_treeevaluate_shorrange ancestor functions

Searching for hot spot method declaration

After identified force_treeevaluate_shorrange is identified as a hot spot and gravity_tree has its own ancestor method, it is time to search for locations of their declaration into the Gadget 2 source code. Although Xprof provides facilities to open and browse application's code, it was preferred to use the project already imported into Eclipse to take advantage of other tools of PEDE for further analysis and to eventually make use of Eclipse's code editor rich capabilities.

The search window shown in Figure 7-28 on page 162 can be opened by clicking the C/C++ language option (menu **Search** → **C/C++**). The search result for force_treeevaluate_shorrange method is shown in Figure 7-29 on page 162.

Select the occurrence under forcetree.c, double-click, and its source code opens in the Eclipse editor. We also opened the occurrence under gravtree.c, which turned out to be the only location where force_treeevaluate_shorrange is called.

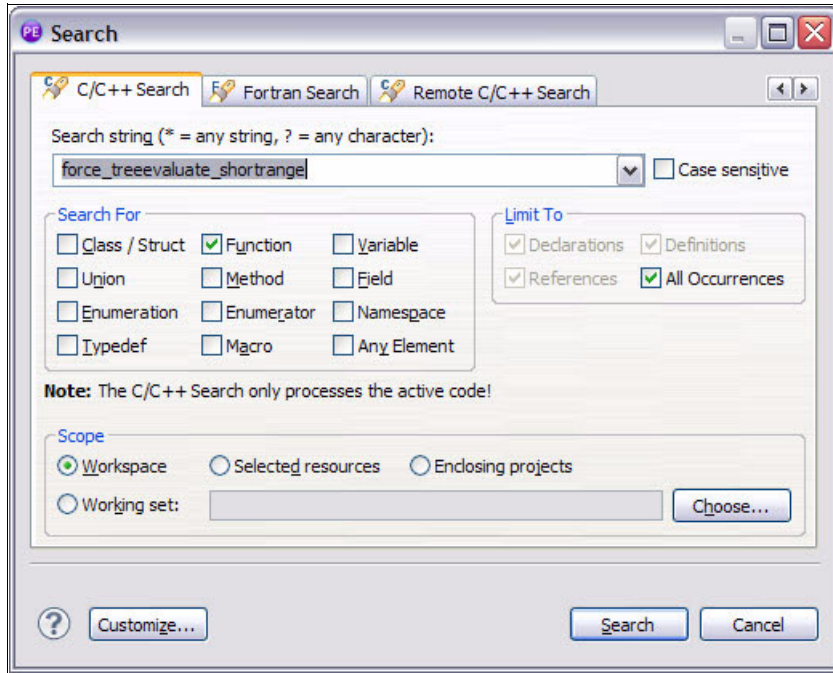


Figure 7-28 Searching for `force_treeevaluate_shortrange`

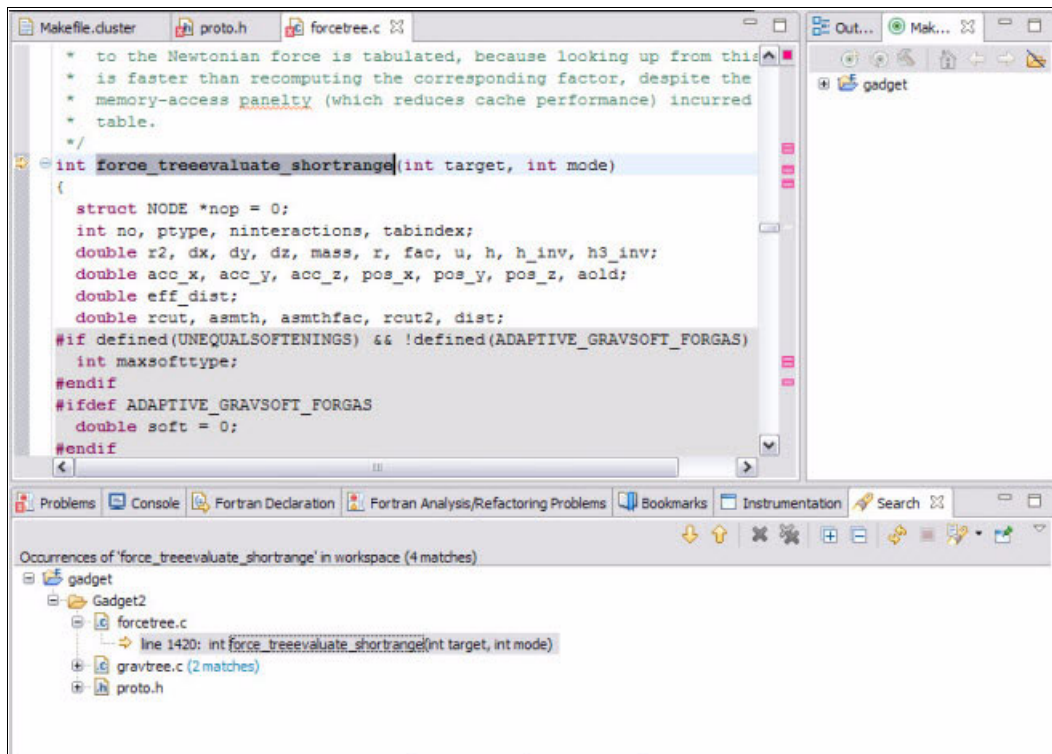


Figure 7-29 Opening `force_treeevaluate_shortrange` implementation

Further information about the Eclipse search in the C/C++ source code is at: **Help** → **Help contents** → **C/C++ Development User Guide** → **Tasks** → **Searching the CDT**.

7.3 Analyzing a ScaLAPACK routine using the HPCT Toolkit

ScaLAPACK is a library of high-performance linear algebra routines for parallel distributed memory machines. ScaLAPACK solves dense and banded linear systems, least squares problems, eigenvalue problems, and singular value problems.

In this section, we analyze the performance of the ScaLAPACK routine PDGETRF, which performs an LU factorization of a dense real matrix. This calculation is similar to that used in the well-known Linpack Benchmark.

7.3.1 The Structure of ScaLAPACK

ScaLAPACK is a continuation of the LAPACK project, which contains high-performance linear algebra software for shared memory computers. LAPACK itself is built on top of the well-known Basic Linear Algebra Subprograms (BLAS). The BLAS routines are a set of kernels for performing low-level vector and matrix operation. Since the interfaces to the BLAS routines have been defined to be a de facto standard, computer vendors can provide highly-tuned versions of the BLAS routines, designed to perform optimally on a given architecture.

ScaLAPACK extended the idea of the BLAS and defines a set of BLACS. In this way, high performance can be achieved by confining machine-specific optimizations to the BLAS and BLACS libraries.

The high-level structure of the ScaLAPACK and related library is shown in Figure 7-30.

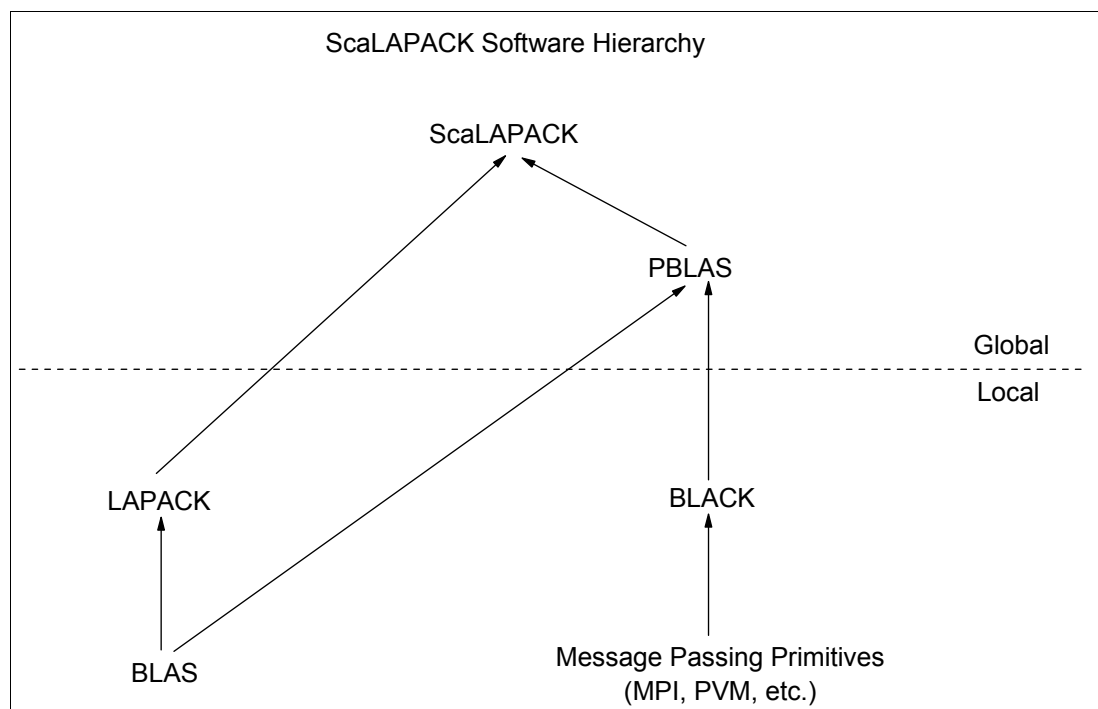


Figure 7-30 ScaLAPACK structure

The relevant components of the directory structure of the ScaLAPACK distribution are:

```
SCALAPACK
  lapack-3.4.1
    BLAS
```

```
    SRC
scalapack-2.0.2
  BLACS
  PBLAS
  SRC
  TESTING
  LIN
  EXAMPLE
```

The ScaLAPACK library contains an extensive suite of test programs in the `scalapack-2.0.2/TESTING` directory. Typically these test programs test an individual routine with many different input parameters to test the correctness of the installation. We modified the top-level driver `pdludriver.f`, used it to build the test program `xdlu` and to increase the value of the parameter `TOTMEM`, so that larger problem sizes are handled. We also modified the data file that drives the `xdlu` program to solve a single large problem, rather than many small ones.

7.3.2 The build process

Both LAPACK and ScaLAPACK have a top-level Makefile, which defines all the build targets and the dependencies. Each of these Makefiles includes a file (`make.inc` and `SLmake.inc`, respectively) that defines the compiler options to be used on a particular platform.

Within each directory, we run the following make commands:

```
lapack-3.4.1
  make blaslib
  make lapacklib
scalapack-2.0.2
  make lib
  make scalapackexe
```

Within the corresponding Eclipse/PTP project, we define these targets by right-clicking the `lapack-3.4.1` and `scalapack-2.0.2` directories in the Project Explorer window and selecting **Make Targets** → **Build (Alt-F9)** → **Add**.

7.3.3 CPU profiling

The first step in CPU profiling is to perform a CPU hot-spot analysis. To do this, we add the flags `-g -pg` to `lapack-3.4.1/make.inc` and `scalapack-2.0.2/SLmake.inc`, and rebuild the package.

Running the program on a single node:

```
poe ./xdlu -procs 16
```

The run generates a series of `gmon.out` files in the format used by the standard UNIX utility `gprof`. On the Power architecture, running either Linux or AIX, the `Xprof` utility can be used to explore these profiles:

```
source /opt/ibmhpc/ppdev.hpct/env_sh
Xprof ./xdlu prof*/gmon*
```

The following window appears as shown in Figure 7-31 on page 165.

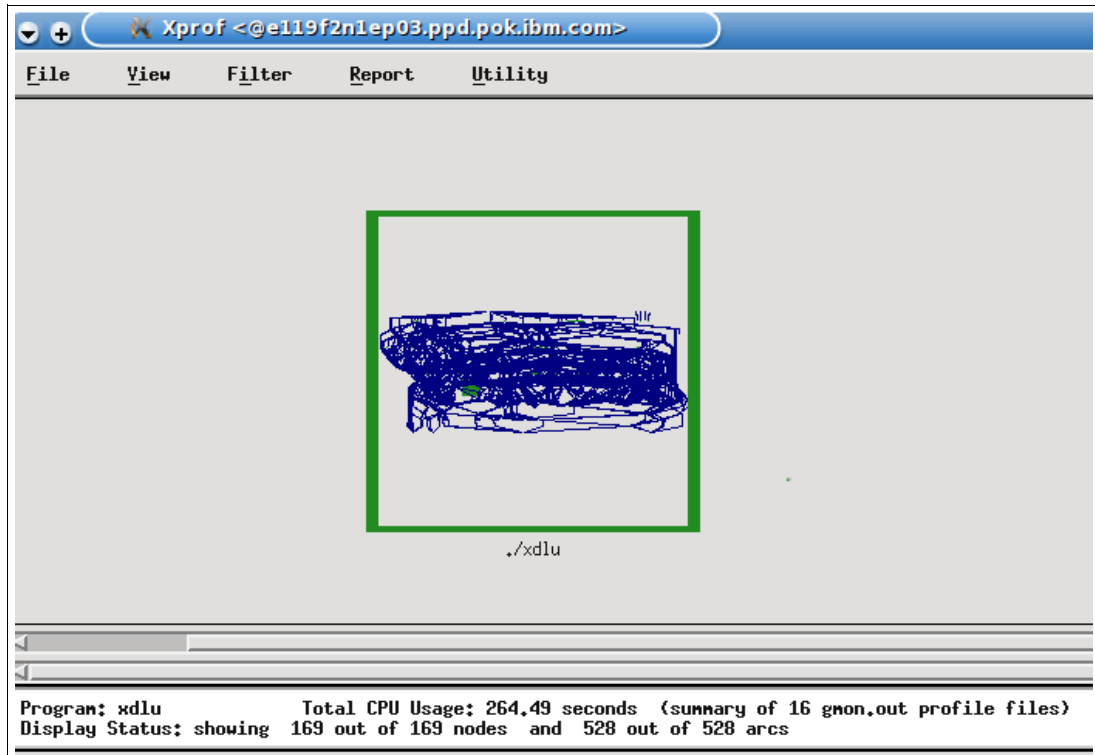


Figure 7-31 Profile report

We run **View** → **Uncluster** functions so that the program call tree is no longer enclosed in the box associated with the program code. This changes the appearance of the window, as shown in Figure 7-32 on page 166.

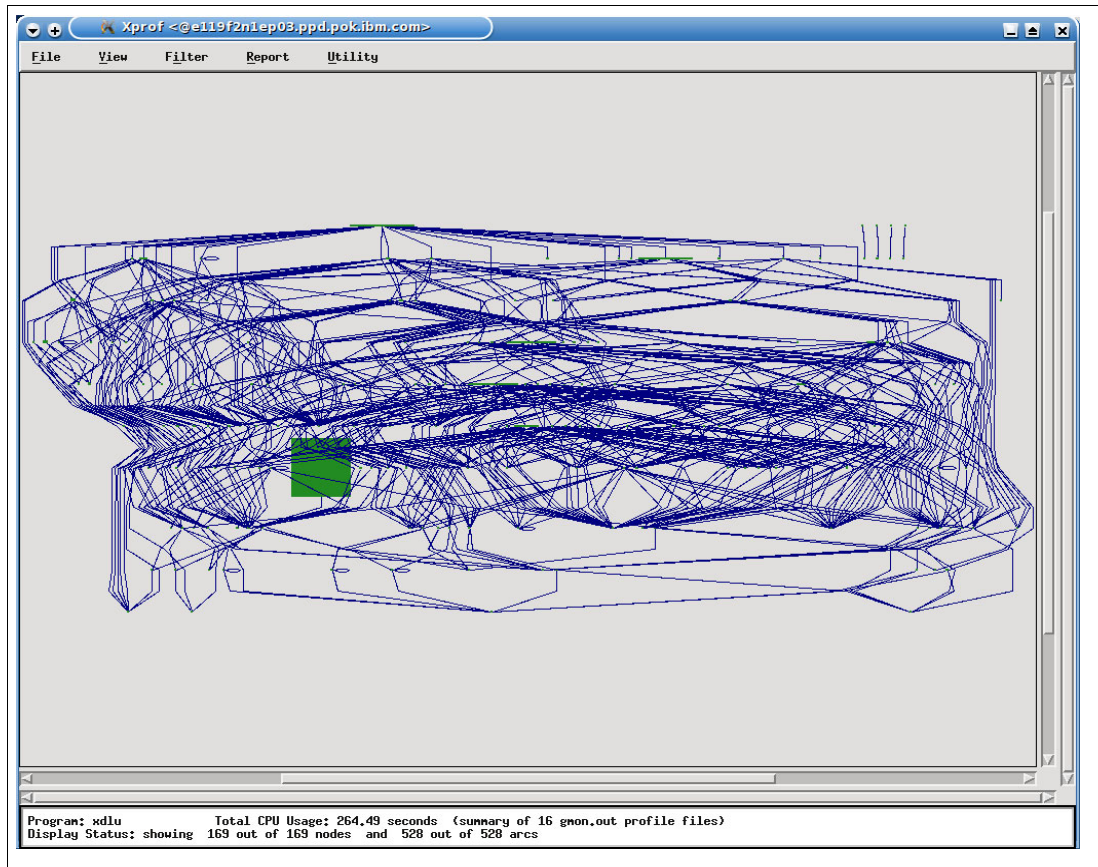


Figure 7-32 Profile report uncluster view

Next we examine the traditional gprof “flat profile” by selecting **Report** → **Flat Profile** (Figure 7-33 on page 167).

Flat Profile <@e119f2n1ep03.ppd.pok.ibm.com>

File		Code Display		Utility			
%time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name	
92.5	244.70	244.70	44262	5.53	5.53	dgemm [21]	
2.1	250.29	5.59	472484084	0.00	0.00	lnul [14]	
1.3	253.75	3.46	400040000	0.00	0.00	pdrand [12]	
1.1	256.71	2.96	445781188	0.00	0.00	ladd [15]	
0.6	258.37	1.66	23048	0.07	0.07	dgenv [19]	
0.5	259.81	1.44	2576	0.56	5.30	pdmatgen [10]	
0.4	260.86	1.05	6250	0.17	0.17	dtrsn [21]	
0.4	261.79	0.93	32	29.06	29.06	pdlange [22]	
0.3	262.48	0.69	41400	0.02	0.02	dswap [27]	
0.2	263.03	0.55	48	11.46	11.46	dgenv [30]	
0.2	263.45	0.42	38610	0.01	0.01	dger [33]	
0.1	263.60	0.15	3392796	0.00	0.00	PB_Cinfog2l [37]	
0.0	263.73	0.13	19019088	0.00	0.00	jumpit [31]	
0.0	263.86	0.13	3432948	0.00	0.00	PB_Cchkvec [38]	
0.0	263.97	0.11	368	0.30	0.30	pdchekpad [39]	
0.0	264.07	0.10	1637392	0.00	0.00	pdszap [20]	
0.0	264.14	0.07	39852	0.00	0.00	dscal [41]	
0.0	264.21	0.07	23184	0.00	0.04	xjumpn [25]	
0.0	264.24	0.03	39944	0.00	0.00	idanax [44]	
0.0	264.27	0.03	5008	0.01	0.13	pdgetf2 [28]	

Search Engine: (regular expressions supported)

Figure 7-33 Flat profile report view

This shows that the overwhelming majority of time, 92.5%, is spent in the Level 3 BLAS routine DGEMM, which is a matrix-matrix multiplication routine. At this point, the obvious course of action is to replace the Fortran BLAS library, distributed with the LAPACK library, with calls to vendor-specific tuned BLAS - ESSL for the IBM Power architecture, or the MKL library for the Intel x86_64 architecture. However, we continue working with the Fortran DGEMM BLAS routine to demonstrate the other tools in the HPCT suite.

It is unusual in real-world applications to find one routine consuming such a high proportion of CPU time, and it is easy to locate the rectangle corresponding to DGEMM in the call graph. Normally, from the flat profile view, we select the line corresponding to DGEMM, and select **Tools** → **Locate** in the graph to show the DGEMM rectangle, as shown in Figure 7-34 on page 168.

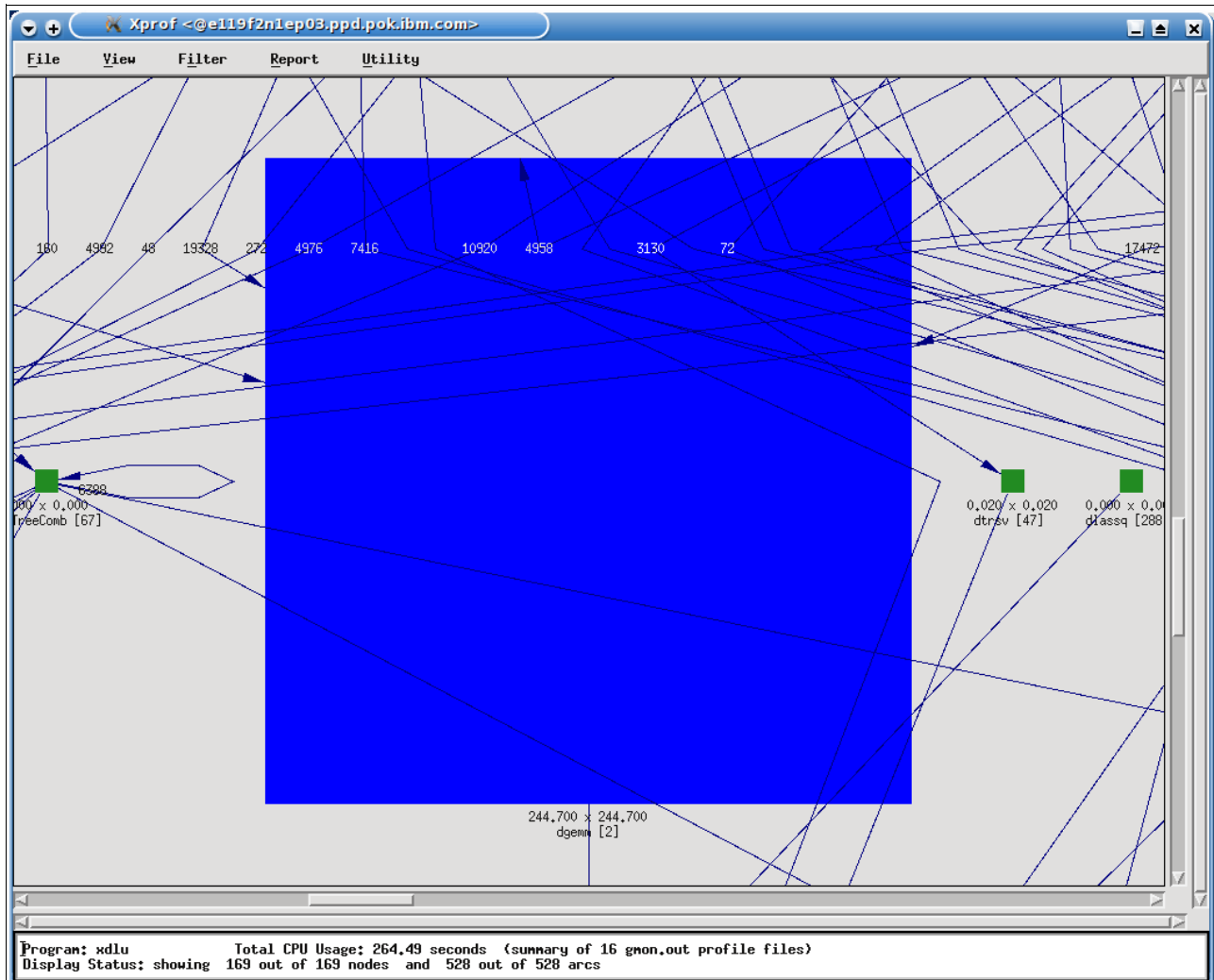


Figure 7-34 DGEMM rectangle

The rectangle is almost exactly square, showing a large amount of time spent in DGEMM, where all of it is spent in the routine itself and nothing in its children. By right-clicking the DGEMM rectangle and selecting **display source code**, a window is displayed that shows the source code of the DGEMM routine. Because all of the routines were compiled with the XL compilers' `-qfullpath` flag, the tool can locate the source code immediately, without specifying a path to the source directory.

By scrolling down the window, we find the lines of code where the time is being spent, as shown in Example 7-1.

Example 7-1 Lines of the program where the running time is spent

no. ticks		
line	per line	source code
303	2	DO 90 J = 1,N
304		IF (BETA.EQ.ZERO) THEN
305		DO 50 I = 1,M
306		C(I,J) = ZERO
307	50	CONTINUE
308		ELSE IF (BETA.NE.ONE) THEN

309		DO 60 I = 1,M
310		C(I,J) = BETA*C(I,J)
311	60	CONTINUE
312		END IF
313	18	DO 80 L = 1,K
314	129	IF (B(L,J).NE.ZERO) THEN
315		TEMP = ALPHA*B(L,J)
316	6	DO 70 I = 1,M
317	24093	C(I,J) = C(I,J) + TEMP*A(I,L)
318	70	CONTINUE
319		END IF
320	80	CONTINUE
321	90	CONTINUE

Each tick is a sample of 1/100th second so that 24093 ticks associated with line 317 corresponds to 240.93 seconds of CPU time associated with the line of code.

7.3.4 HPM profiling

The “DO 70” loop in Example 7-1 on page 168 is written so that the arrays are accessed in column major order, which is typically good for Fortran code. However, better performance can usually be achieved by unrolling DO-loops, so that more use is made of each array element that is loaded into cache before that element is discarded.

Such optimizations are typical of the customized BLAS routines available in the IBM ESSL and the Intel MKL libraries. In this case, we use compiler optimizations to show how different levels of optimization can affect the performance of this loop.

In general, hardware performance counters are extremely hard to interpret in absolute terms without detailed knowledge of both the underlying hardware and the application. However, in tuning an application, it is often possible to get some useful information by examining performance counters before and after tuning occurs.

For ScaLAPACK, we instrumented the xdlu executable for gathering HPM information using the command:

```
hpctInst -dhpm ./xdlu
```

This command creates the file xdlu.inst. Note that in general it is better to instrument just a few key areas of the code because the full HPM instrumentation adds significantly to the overall run time. We therefore instrumented xdlu using Eclipse/PTP, adding instrumentation ONLY to the call to pdgemm. We choose this, rather than dgemm itself, because this section of code remains unchanged even though next we completely replace the dgemm routine.

First, we run the application using the default HPM Event group. For each task, we obtain a .viz file and a .txt file. First, we examine one of the .txt files and search for the section labelled pdgemm, as shown in Figure 7-35 on page 170.

```

Instrumented section: 1 - Label: pdgemm
process: 3276958, thread: 1
file: pdgemm_.c, lines: 259 <--> 506
Context is process context.
No parent for instrumented section.

Inclusive timings and counter values:

Execution time (wall clock time)      : 43.9370487332344 seconds
Initialization time (wall clock time): 0.19987041503191 seconds
Overhead time (wall clock time)       : 0.0135871283710003 seconds

PM_FPU_1FLOP (FPU executed one flop instruction)      :          17912567
PM_FPU_FMA (FPU executed multiply-add instruction)     :          35711748287
PM_FPU_FSQRT_FDIV (FPU executed FSQRT or FDIV instruction) :              0
PM_FPU_FLOP (FPU executed 1FLOP, FMA, FSQRT or FDIV instruction) :          35729660854
PM_RUN_INST_CMPL (Run instructions completed)         :          190402795098
PM_RUN_CYC (Run cycles)                              :          184265923937

Utilization rate                :          99.759 %
Instructions per run cycle       :           1.033
Total scalar floating point operations :          71441.409 M
Scalar flop rate (flops per WCT)  :          1625.995 Mflop/s
Scalar flops per user time       :          1629.925 Mflop/s
Algebraic floating point operations :          71441.409 M
Algebraic flop rate (flops per WCT) :          1625.995 Mflop/s
Algebraic flops per user time    :          1629.925 Mflop/s
Scalar FMA percentage           :           99.975 %
Scalar percentage of peak performance :           9.693 %

```

Figure 7-35 pdgemm output

We note two specific values here: Scalar percentage of peak performance, which is 9.963%, and Algebraic flop rate (flops per WCT), which is 1625.925Mflop/s.

To show the difference, we can potentially tune the dgemm kernel optimally, link the code with the IBM ESSL (Engineering and Scientific Subroutine Library), and repeat the experiment. In this case, we obtain the following information for the pdgemm section, as shown in Figure 7-36 on page 171.


```

Instrumented section: 1 - Label: pdgemm
process: 3670092, thread: 1
file: pdgemm.c, lines: 259 <--> 506
Context is process context.
No parent for instrumented section.

Inclusive timings and counter values:

Execution time (wall clock time)      : 19.9932569861412 seconds
Initialization time (wall clock time): 0.202705550938845 seconds
Overhead time (wall clock time)       : 0.0138184614479542 seconds

PM_FPU_1FLOP (FPU executed one flop instruction)      :
18123119
PM_FPU_FMA (FPU executed multiply-add instruction)    :
35711748287
PM_FPU_FSQRT_FDIV (FPU executed FSQRT or FDIV instruction) :
0
PM_FPU_FLOP (FPU executed 1FLOP, FMA, FSQRT or FDIV instruction) :
35729871406
PM_RUN_INST_CMPL (Run instructions completed)         :
61851849877
PM_RUN_CYC (Run cycles)                               :
83586326217

Utilization rate           :          99.446 %
Instructions per run cycle :          0.740
Total scalar floating point operations :      71441.620 M
Scalar flop rate (flops per WCT)   :      3573.286 Mflop/s
Scalar flops per user time         :      3593.178 Mflop/s
Algebraic floating point operations :      71441.620 M
Algebraic flop rate (flops per WCT) :      3573.286 Mflop/s
Algebraic flops per user time      :      3593.178 Mflop/s
Scalar FMA percentage           :          99.975 %
Scalar percentage of peak performance :          21.368 %

```

Figure 7-36 pdgemm new output

We now see that the Scalar percentage of peak performance increased to 21.368%, and the Algebraic flop rate (flops per WCT) increased to 3573.286Mflop/s.

The DGEMM routine in ESSL was carefully constructed to make much effective use of the memory hierarchy, so that as much use is made of data loaded into the caches, close to the processor, before calculation needs to be delayed by accessing memory. We can see this by looking at a counter group that lists details of memory access. When we run the Fortran version, and look at group 11, we see the values for the counters shown in Figure 7-37 on page 172.

PM_DATA_FROM_MEM_DP (Data loaded from double pump memory)	:	0
PM_DATA_FROM_DMEM (Data loaded from distant memory)	:	0
PM_DATA_FROM_RMEM (Data loaded from remote memory)	:	10345114
PM_DATA_FROM_LMEM (Data loaded from local memory)	:	11189436
PM_RUN_INST_CMPL (Run instructions completed)	:	191101049438
PM_RUN_CYC (Run cycles)	:	185020305804

Figure 7-37 Memory counter group details

This can be contrasted with values obtained from the ESSL version of DGEMM, as shown in Figure 7-38.

PM_DATA_FROM_MEM_DP (Data loaded from double pump memory)	:	0
PM_DATA_FROM_DMEM (Data loaded from distant memory)	:	0
PM_DATA_FROM_RMEM (Data loaded from remote memory)	:	2878806
PM_DATA_FROM_LMEM (Data loaded from local memory)	:	2243331
PM_RUN_INST_CMPL (Run instructions completed)	:	62033373737
PM_RUN_CYC (Run cycles)	:	83109304433

Figure 7-38 Values from the ESSL version of DGEMM

We can see clearly from these statistics that the ESSL DGEMM routine loads far less data from memory.

Raw performance counters can also be examined using the peekperf GUI, as shown in Figure 7-39 on page 173.

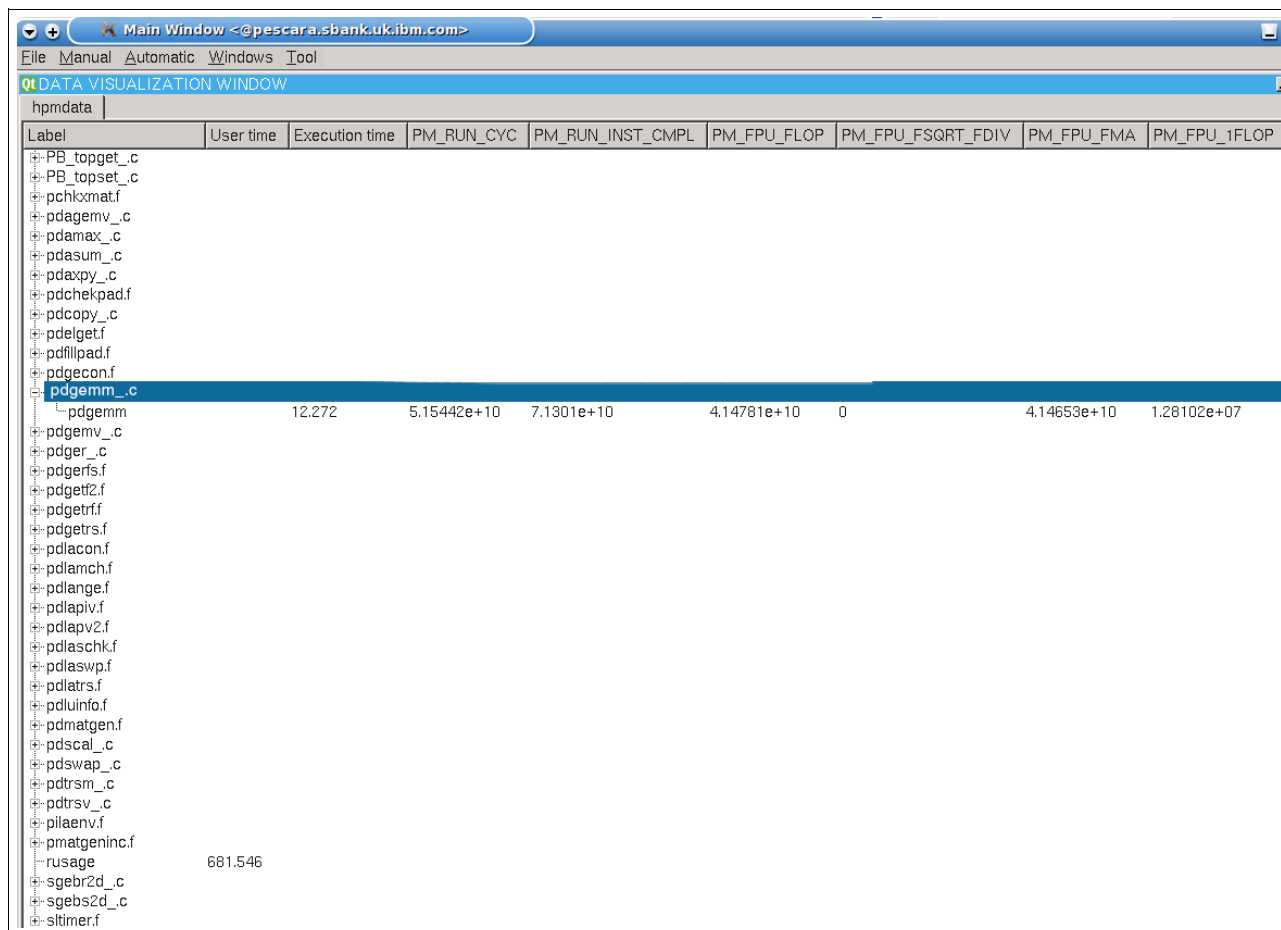


Figure 7-39 Raw performance counters shown using the peekperf GUI

7.3.5 MPI profiling

We next perform MPI profiling, which can be enabled by linking with the libmpitrace library. We access the PEDE environment variables source `/opt/ibmhpc/ppdev.hpct/env_sh` and add the following syntax to the link step in the file `TESTING/LIN/Makefile`:

```
-L$(IHPCT_BASE)/lib64 -lmpitrace
```

Alternatively, we can use dynamic instrumentation to add tracing to a prebuilt executable:

```
hpcInst -dmpi xdlu
```

Although we do not modify the source code to alter the communications patterns, we can still influence the way the code performs by modifying some of the parameters in the input file.

One of the key parameters that affects the performance of ScaLAPACK routines is the block size, given by `MB` in the input file. If the block size in linear algebra routines is too small, excessive amounts of communication with small messages is generated. On the other hand, if the block size is too large, the blocks will be too large to fit into the processor cache, so that the reduction in communications is offset by the reduced computational efficiency.

To illustrate this, we show the textual MPI profile information and the graphical display of MPI calls for the two cases where `NB=32` (which is a reasonably good value for the Fortran code), and `NB=256` (which is too large). In each case, we request that information be collected for all

tasks, and all events are also collected to the trace file. To achieve this, we set the following environment variables:

```
export OUTPUT_ALL_RANKS=yes
export TRACE_ALL_TASKS=yes
export TRACE_ALL_EVENTS=yes
```

We found initially that the default number of events to trace, 30,000, was too small, and so we increased it as follows:

```
export MAX_TRACE_EVENTS=500000
```

Examining the file `hpct_0_0.mpi.txt`, the trace from task 0, for `NB=32` is shown in Figure 7-40.

MPI Routine	#calls	avg. bytes	time(sec)
MPI_Comm_size	1	0.0	0.0000
MPI_Comm_rank	3	0.0	0.0000
MPI_Send	3428	32055.3	0.101
MPI_Rsend	6	8.0	0.0000
MPI_Isend	47755	2307.9	0.249
MPI_Recv	56648	3861.0	6.608
MPI_Testall	47755	0.0	0.048
MPI_Waitall	473	0.0	0.000
MPI_Bcast	9341	15476.4	1.248
MPI_Barrier	2	0.0	0.020
MPI_Reduce	789	1270.8	0.207
MPI_Allreduce	37	5.9	0.004

total communication time = 8.485 seconds.			
total elapsed time = 56.336 seconds.			

Figure 7-40 `hpct_0_0.mpi.txt` output file

The bottom of this file is shown in Figure 7-41.

<pre>Communication summary for all tasks: minimum communication time = 8.485 sec for task 0 median communication time = 9.411 sec for task 4 maximum communication time = 10.226 sec for task 14</pre>

Figure 7-41 `hpct_0_0.mpi.txt` bottom section

The corresponding sections of the task 0 trace file when `NB=256` are shown in Figure 7-42 on page 175.

MPI Routine	#calls	avg. bytes	time(sec)
MPI_Comm_size	1	0.0	0.0000
MPI_Comm_rank	3	0.0	0.000
MPI_Send	515	411610.2	0.442
MPI_Rsend	6	8.0	0.000
MPI_Isend	45691	2367.4	0.223
MPI_Recv	52190	6067.0	16.339
MPI_Testall	45691	0.0	0.046
MPI_Waitall	128	0.0	0.000
MPI_Bcast	6555	626.8	0.126
MPI_Barrier	2	0.0	0.031
MPI_Reduce	237	4286.7	0.860
MPI_Allreduce	37	5.9	0.002

total communication time = 18.069 seconds.
total elapsed time = 97.995 seconds.

Figure 7-42 Task 0, NB = 256

The communication summary of all tasks is shown in Figure 7-43.

Communication summary for all tasks:
minimum communication time = 14.504 sec for task 9
median communication time = 18.069 sec for task 0
maximum communication time = 24.945 sec for task 15

Figure 7-43 Communication summary

A summary of MPI communications can be obtained by loading the .viz files into the peekperf GUI, as shown in Figure 7-44 on page 176.

The screenshot shows a window titled 'Main Window - [DATA VISUALIZATION WINDOW] <@pecara.sbank.uk.ibm.com>'. The window contains a table with the following data:

Label	Count	WallClock	Transferred Bytes
BI_Arecv.c			
BI_Asend.c			
BI_Asend(BI_Asend.c)			
MPI_Isend_7	47755	0.248565	1.10216e+08
BI_BuffsFree.c			
BI_Rsend.c			
BI_Srecv.c			
BI_Srecv(BI_Srecv.c)			
MPI_Recv_8	56648	6.60792	2.18715e+08
BI_Ssend.c			
BI_Ssend(BI_Ssend.c)			
MPI_Send_6	3428	0.101119	1.09886e+08
blacs_barr.c			
blacs_map.c			
blacs_pinfo.c			
dgamx2d.c			
dgebr2d.c			
dgebs2d.c			
dgsum2d.c			
igamx2d.c			
igamx2d.c			
igebr2d.c			
igebs2d.c			
igsum2d.c			
sgebr2d.c			
sgebs2d.c			
SUMMARY			
MPI_Allreduce	37	0.003735	220
MPI_Barrier	2	0.020449	0
MPI_Bcast	9341	1.2476	1.44565e+08
MPI_Comm_rank	3	2e-05	0
MPI_Comm_size	1	3e-06	0
MPI_Irecv			
MPI_Isend	47755	0.248565	1.10216e+08
MPI_Recv	56648	6.60792	2.18715e+08
MPI_Reduce	789	0.20655	1.00264e+06
MPI_Rsend	6	2.4e-05	48
MPI_Send	3428	0.101119	1.09886e+08
MPI_Testall	47755	0.048196	0
MPI_Waitall	473	0.000389	0

Figure 7-44 .viz file into the peekperf GUI

It is also instructive to examine the communications patterns for both cases. We examine the .mpt file using the peekview GUI. In both cases, we present two views: the first is zoomed in to show approximately one second of execution, and the second zooms in further to show approximately 0.2 seconds.

For the NB=32 example, the second view is shown in Figure 7-45 on page 177.

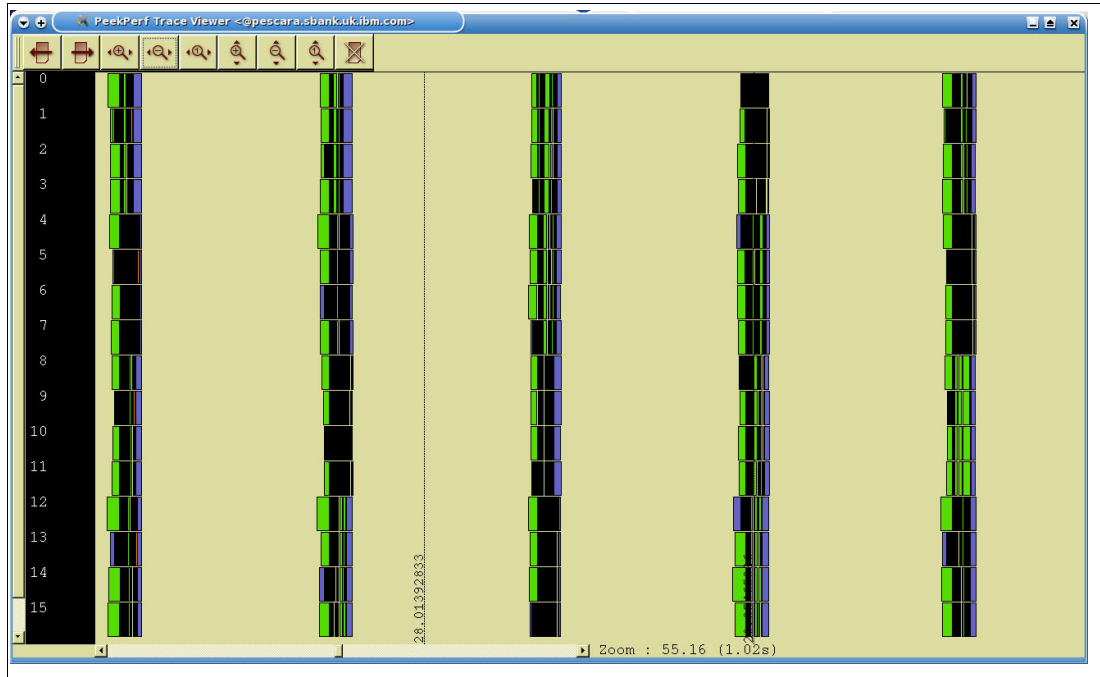


Figure 7-45 NB case second view

For the NB=32 case, the 0.2 second view is shown in Figure 7-46.

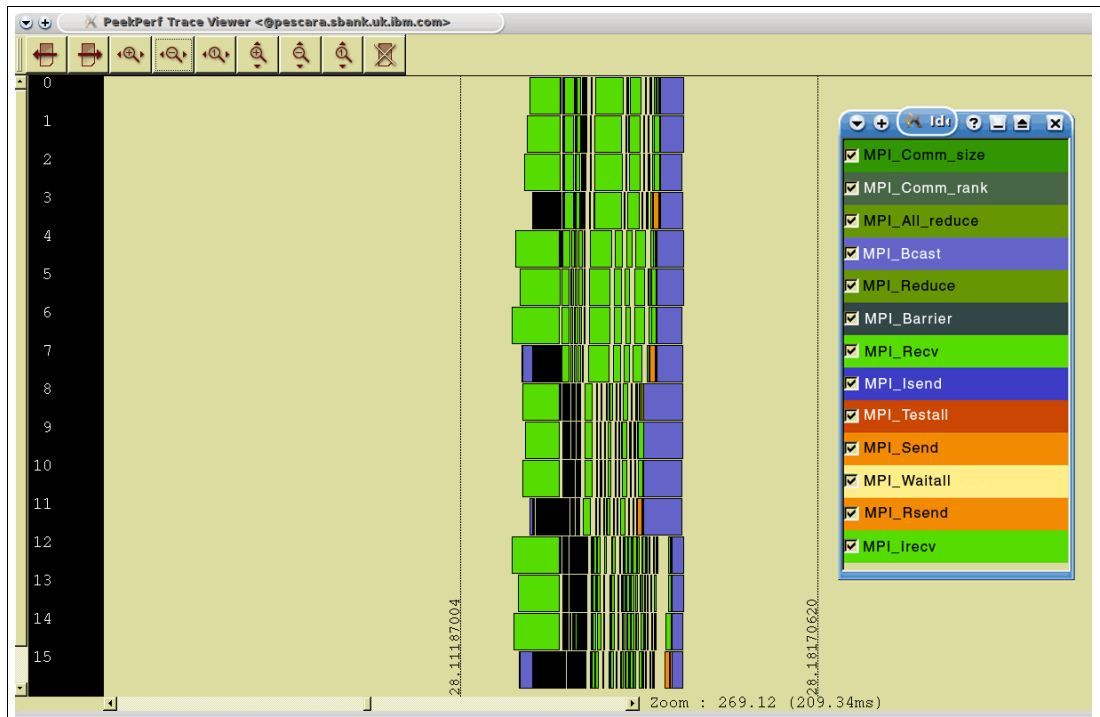


Figure 7-46 NB=32 case 0.2 view

For the NB=256 case, the one second view is shown in Figure 7-47 on page 178.

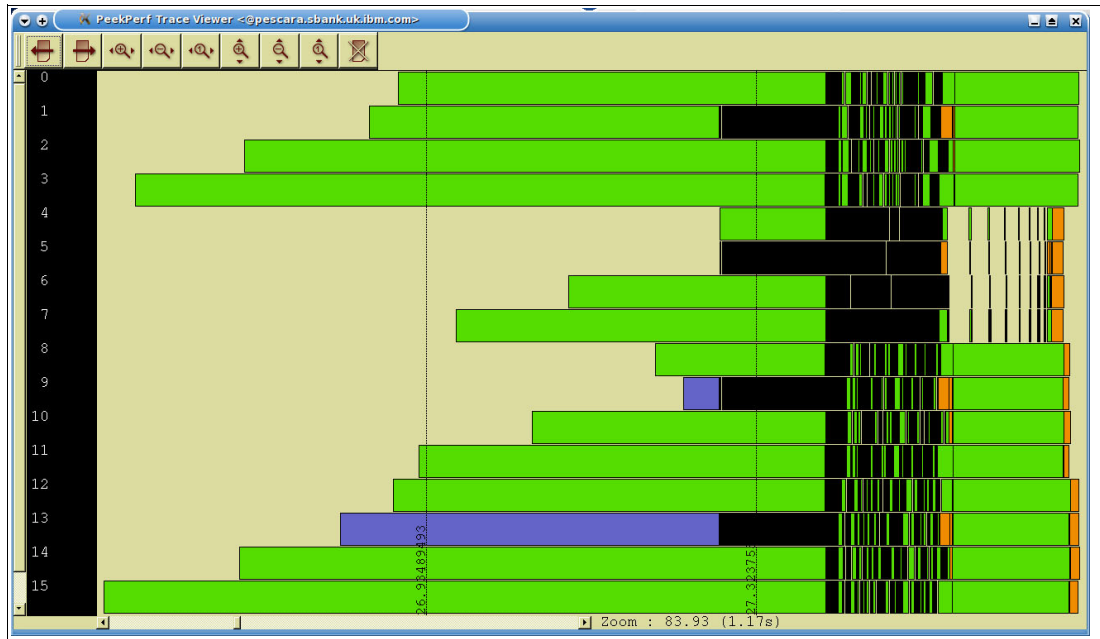


Figure 7-47 Communication phase

We see that the communications phase lasts nearly the full second in this case. The 0.2 second view, zooming in on the busy part, is shown in Figure 7-47.

Figure 7-48 shows the peekperf trace view.

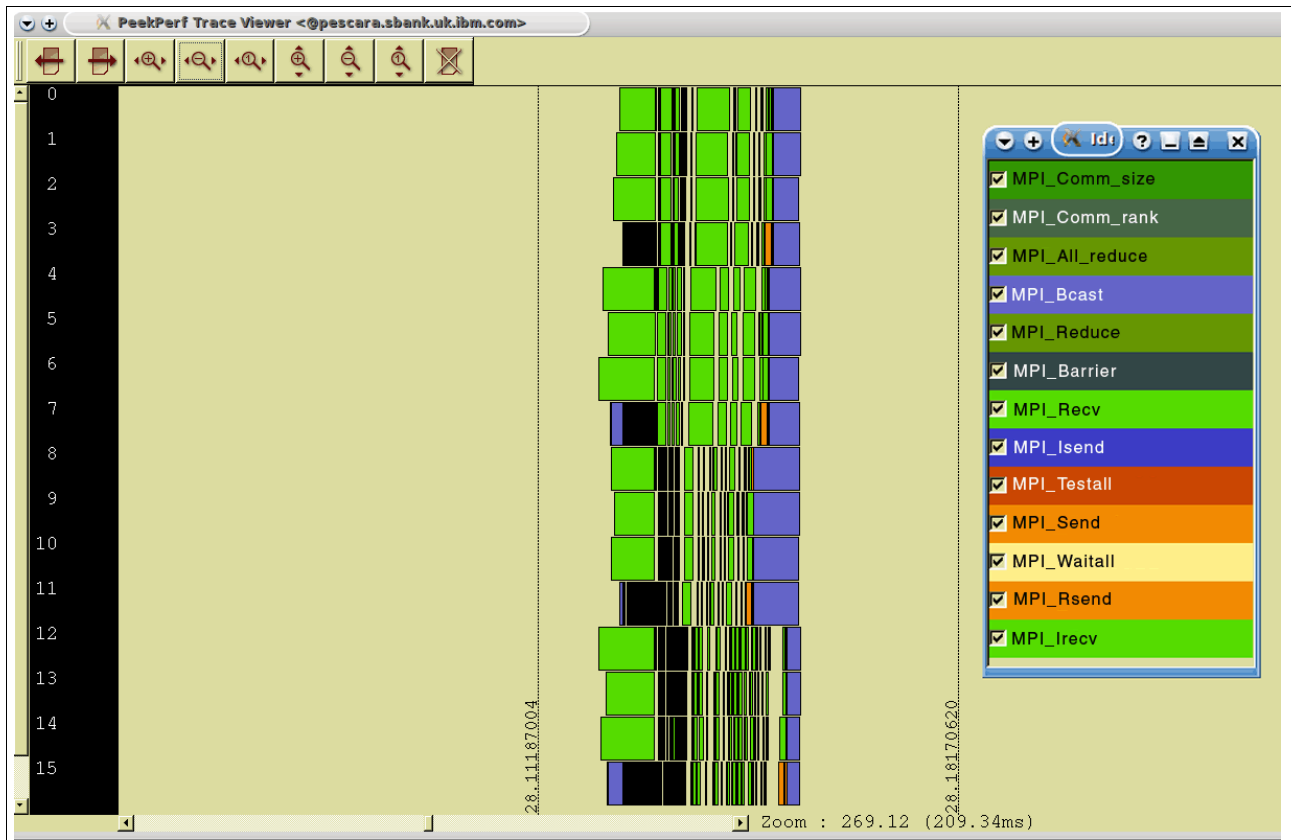
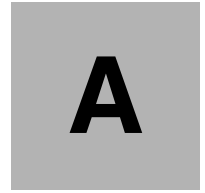


Figure 7-48 Peekperf trace view



HPC Toolkit environment variables

This appendix provides a list of environment variables, in tabular form, used by the HPC Toolkit.

General environment variables

Table A-1 shows the general environment variables.

Table A-1 General environment variables

Variable	Description	Default value
IHPCT_BASE	Specifies the path name of the directory in which the IBM HPC Toolkit is installed (usually /opt/ibmhpc/ppdev.hpct).	none
GPROF	Sets options for generating the gmon.out file. See the man page for gprof.	none

Variables related to hardware performance counters

Table A-2 shows the variables for the hardware performance counters.

Table A-2 Variables related to hardware performance counters

Variable	Description	Default value
HPC_EXCEPTION_COUNT	Is the number of tasks to be collected for minimum and maximum value of the metric specified by --exmetric.	10
HPC_EXCEPTION_METRIC	Is the metric to be used to determine which task's data to collect.	ELAPSED_TIME
HPC_OUTPUT_NAME	Specifies the name prefix of the output files name.hpm.txt and name.hpm.viz.	hpct
HPC_TRACE_MAX_BUFFERS	Set this to a positive integer value that defines the number of in-memory buffers that are used to store trace events.	1
HPC_TRACE_STORE	Specifies the mode for trace data collection.	memory
HPC_UNIQUE_FILE_NAME	Set to yes to generate unique file names for generated ASCII and XML output files. Set to no to generate the file name exactly as specified by HPC_OUTPUT_NAME.	
HPC_USE_HPCRUN	Use hpcrun.	
HPM_AGGREGATE	Specifies the name of a plug-in that defines the HPM data aggregation strategy.	

Variable	Description	Default value
HPM_ASC_OUTPUT	Set to yes to generate an ASCII output file with the name name.hpm.txt.	
HPM_COUNTING_MODE	Specifies the CPU mode where counting will occur.	user
HPM_DIV_WEIGHT	FLOPS weighting factor.	
HPM_EVENT_SET	A single value that specifies the hardware counter group to be used, or a comma-delimited list of hardware counter groups to be multiplexed.	POWER6:127 POWER7:147 X86:1
HPM_EXCLUSIVE_VALUES	Set to yes if exclusive counter values in nested counter regions are to be computed.	
HPM_PRINT_FORMULA	Set to yes to print the definitions of the derived metrics. Set to no to suppress this output.	no
HPM_PRINT_TASK	Specifies the MPI task that has its results displayed.	0
HPM_ROUND_ROBIN_CLUSTER	Allows setting the number of groups distributed per task.	
HPM_SLICE_DURATION	Specifies the interval, in milliseconds, to be used when hardware counter groups are multiplexed.	100 milliseconds
HPM_STDOUT	Set to yes to write ASCII output to stdout.	yes
HPM_VIZ_OUTPUT	Set to yes to generate an XML output file with the name name.hpm.viz.	

Variables related to MPI profiling

Table A-3 shows the variables for MPI profiling.

Table A-3 Variables related to MPI profiling

Variable	Description	Default value
MAX_TRACE_EVENTS	Specifies the maximum number of trace events that can be collected per task.	30,000
MAX_TRACE_RANK	Specifies the MPI task rank of the highest rank process that has MPI trace events collected.	256

Variable	Description	Default value
MT_BASIC_TRACE	Specifies whether the MAX_TRACE_RANK environment variable is checked.	
OUTPUT_ALL_RANKS	Set to yes to generate trace files for all MPI tasks.	Only for task 0 and the tasks that have the minimum, maximum, and median total MPI communication time.
TRACE_ALL_EVENTS	Set to yes to generate a trace containing trace events for all MPI calls after MPI_Init().	yes
TRACE_ALL_TASKS	Set to yes to generate MPI trace files for all MPI tasks in the application.	no
TRACEBACK_LEVEL	Specifies the number of levels to walk back in the function call stack when recording the address of an MPI call.	0

Variables related to I/O profiling

Table A-4 shows the variables used for I/O profiling.

Table A-4 Variables related to I/O profiling

Variable	Description	Default value
MIO_FILES	Specifies one or more sets of file name and the profiling library options to be applied to that file.	
MIO_DEFAULTS	Specifies the I/O profiling options to be applied to any file whose file name does not match any of the file name patterns specified in the MIO_FILES environment variable.	
MIO_STATS		
TKIO_ALTLIB	The path name of an interface module used by the I/O profiling library.	

Variables relating to OpenMP profiling

Table A-5 on page 183 shows the variables for OpenMP profiling.

Table A-5 Variables for OpenMP profiling

Variable	Description	Default value
POMP_LOOP	Specifies the level of OpenMP profiling instrumentation for OpenMP parallel regions.	all (check)
POMP_PARALLEL	Specifies the level of OpenMP profiling instrumentation for OpenMP parallel regions.	all (check)
POMP_USER	Specifies the level of OpenMP profiling for user functions.	

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Online resources

These websites are also relevant as further information sources:

- ▶ OpenMP compilers
<http://openmp.org/wp/openmp-compilers/>
- ▶ IBM XL compilers
<http://www-01.ibm.com/software/awdtools/fortran>
<http://www-01.ibm.com/software/awdtools/xlcppp/>
- ▶ information about the UPC language can be obtained at:
<https://upc.gwu.edu/>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



IBM Parallel Environment (PE) Developer Edition

(0.2"spine)
0.17"->0.473"
90->249 pages



IBM Parallel Environment (PE) Developer Edition



Provides installation instructions

This publication helps strengthen the position of IBM software solutions and enables for High Performance Computing (hardware, software, and tools) with a well-defined and documented deployment model within an IBM environment. As a result, customers receive a planned foundation for dynamic infrastructure for parallel High Performance Computing (HPC) applications.

Includes case scenarios

Helps improve performance

This IBM Redbooks publication addresses topics to take advantage of the strengths of IBM PE Developers Edition for HPC applications. The objective is to solve customer's challenges and maximize systems' throughput, performance, and management. This publication examines the tools, utilities, documentation, and other resources available to help the IBM technical teams provide solutions and support for IBM HPC solutions in an IBM hardware environment.

This IBM Redbooks is targeted toward technical professionals (consultants, technical support staff, IT Architects, and IT Specialists) responsible for providing HPC solutions and support.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-8075-00

ISBN 0738437689