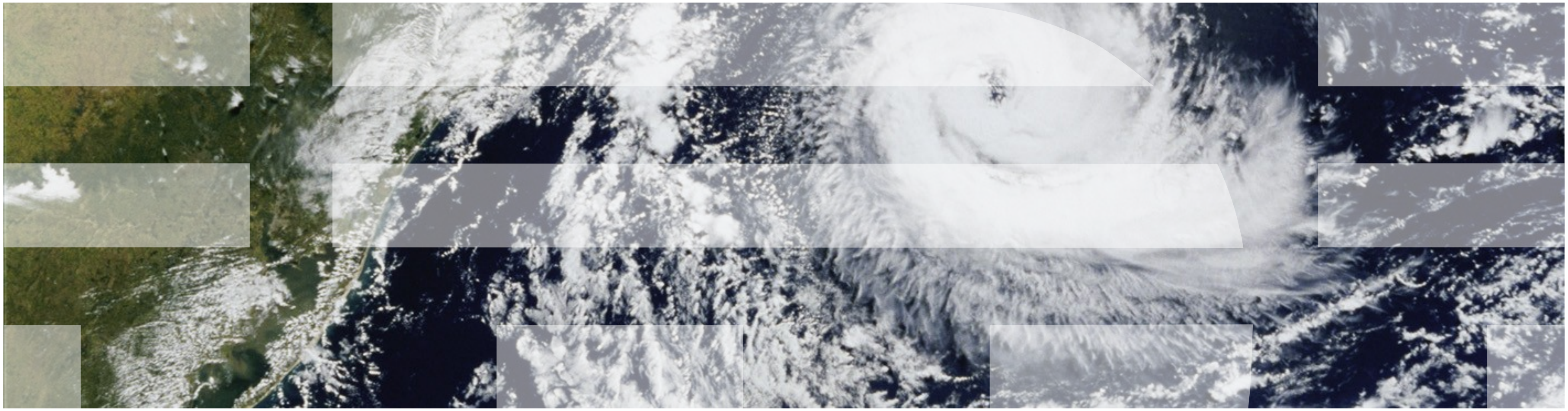
Kit Barton kbarton@ca.ibm.com
IBM Toronto Lab

Performance Tuning with the IBM XL Compilers
SciNet Tutorial



IBM Rational Disclaimer

Please Note:

- IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.
- Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision.
- The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract. The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Agenda

- **Part 1 – Overview of XLC/C++ V12.1 and XLF V14.1**
  - New features
  - Compile-time improvements
  - C++ Optimization Improvements
  - Debugging optimized code
  - OpenMP 3.1

- **Part 2 – Performance Tuning using the XL Compilers**
  - Overview of compiler options and frequently used pragmas/directives
  - C++ Optimization Tuning
  - Debugging Optimized Code
  - Tuning parallel codes
  - Data prefetch and reorganization
  - Vectorization
  - SIMD Tuning

Part 1 – Overview of XLC/C++ V12.1 and XLF V14.1

Major features of the XLC and XLF compilers

- Target AIX, Linux on Power
  - Common technology for Blue Gene/Q, and zOS (XLC/C++ only for zOS)
- Language standard compliance
  - C99 Standard compliance
  - C++98 and subsequent TRs, Selected C++0x features
  - Fortran 2003 Standard compliance
  - OpenMP implementation for parallel programming
- Fully backward compatible with objects compiled with older compilers
  - Supports mix-and-match of objects generated with different compilers and optimization levels
  - Backward compatibility through option control in some rare situations:
    - C++ name mangling, OpenMP TLS, etc
- GCC affinity
  - Partial source and full binary compatibility with gcc

Optimization capabilities of the XL Compilers

- Platform exploitation
  - qarch: ISA exploitation
  - qtune: skew performance tuning for specific processor, including tune=balanced
  - Large portfolio of compiler builtins and performance annotations

- Mature compiler optimization technology
  - Five distinct optimization packages
  - Debug support and annotated assembly listings at all optimization levels
    - Debug experience is affected by aggressive optimization
  - Aggressive loop restructuring
  - Whole program optimization
  - Profile-directed optimization

MASS and MASSV

- Libraries of mathematical routines tuned for optimal performance on various POWER architectures
  - General implementation tuned for POWER
  - Specific implementations tuned for specific POWER processors (pwr5, pwr6, pwr7)

- Compiler will automatically insert calls to MASS/MASSV routines at higher optimization levels
  - Users can add explicit calls to the library

- References
  - "*Improve the performance of programs calling mathematical functions*" by Robert F. Enenkel and Daniel M. Zabawa, http://www.ibm.com/developerworks/rational/library/10/improveperformanceprogramsmathfunctions/index.html
  - Autovectorization sandbox: http://www.ibm.com/developerworks/downloads/emsandbox/power_infrastructure.html
  - MASS Webpage: http://www.ibm.com/software/awdtools/mass

Major Features in the V12.1 XL C/C++ Release

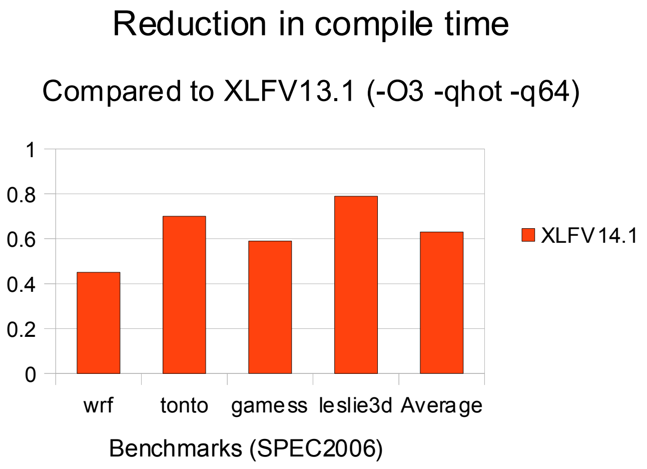
- Customer Requirements
  - GCC style atomic operation support
  - More aggressive restrict pointer implementation
  - SIMD level pragmas
  - Loop iteration pragmas
  - Functrace enhancement (optfile)
  - Inline ASM enhancements
  - Boost and GCC compatibility enhancements
- Performance Improvements
  - Improve performance of applications using object-oriented language features
  - Non-loop SIMDization for more VSX vector exploitation
- Language Standards
  - Continue C++0X phased feature release: constexpr, rvalue ref, strong enum…
  - Start C1X phased feature release: static_assert ..
  - OpenMP 3.1 conformance
- Productivity and Usability
  - **Reduced memory usage** for whole-program optimization
  - **Faster compilation** of complex codes, e.g, C++ template code
  - **Debugging enhancements**, e.g, better support for debugging optimized code, support for C++0x subset features
  - **Portability enhancements**
  - **XML Transformation report** content extension and usability enhancement
  - **Tooling integration** (PTP, HPCS toolkit)

Major Features in the V14.1 XL Fortran Release

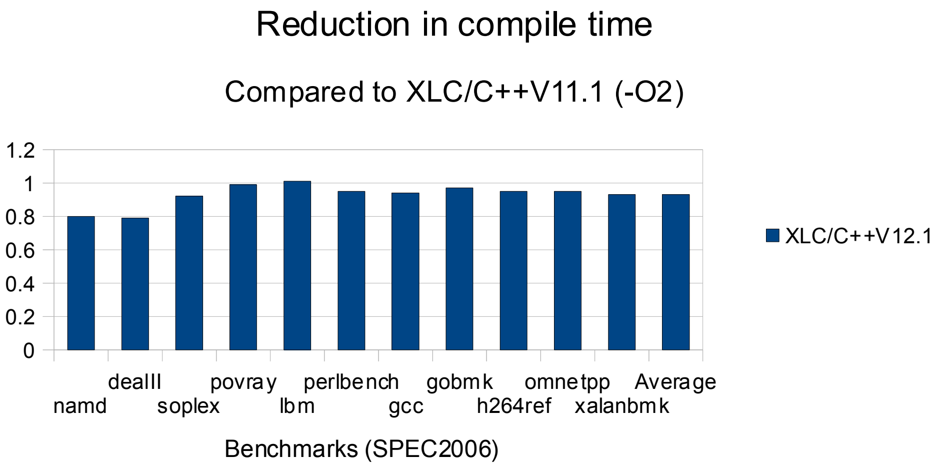
- Customer Requirements from HPC, ISV and other clients
  - Alignment control (alignment directive)
  - SIMDization control (SIMD level directives)
  - Loop iteration directives
  - Functrace enhancement (module name, optfile)
  - Initialization of malloc storage (-qinitalloc option)
  - Traceback enhancements
  - WORKSHARE improvements in FORALL
- Performance Improvements
  - Improved handling of F90 array language
  - More precise aliasing analysis for Fortran dummy arguments
  - Loop transformation enhancements at -O3 –qhot for SIMDization/Vectorization and data locality
- Language Standards
  - OpenMP 3.1 conformance
  - Subset of Fortran 2008 (non coarray part): CONTIGUOUS, BLOCK, Internal procedures as actual arguments and procedure pointer targets, compiler_version, compiler_options
- Productivity and Usability
  - **Compile-time performance improvement** for F90 array language and F90 modules
  - **Compilation memory footprint improvements**
  - **Compilation scalability improvements** including 64-bit components
  - **Improved XML compiler transformation reports**
  - **Improved diagnostics control** (-qhaltonmsg, -qmaxerr options)
  - **Debugging enhancements** including support for Fortran 2003 features

Infrastructure Improvements

- Compilation time
  - – Significant effort placed to improve compilation speed for large applications
  - – Caching of directory lookups to speed up processing of header files (C/C++)
- Scalability
  - – Improved memory utilization for IPA process
  - – More compiler components running on 64-bit mode
    - Eliminates compiler limitations for optimizing large source files


Reduction in compile time

Compared to XLFV13.1 (-O3 -qhot -q64)


Reduction in compile time

Compared to XLC/C++V11.1 (-O2)

C++ Optimization improvements

- General performance improvements
  - More aggressive dead code elimination
  - More precise side-effect analysis for IPA process (level=1 and up)

- Significant effort placed to improve runtime performance for C++ applications
  - More effective inlining to support object-oriented C++
  - Better support for always_inline attribute
  - Improved management for temporary objects
  - Improved management of aggregate returns
  - Improved management of aggregate value parameters
  - More effective implementation of the C99 “restrict” keyword (both C and C++)

- Expect runtime performance improvements for object-oriented C++ codes

Large TOC access model

- AIX uses a global data structure to access statically-allocated variables
  - Compiler generates a load off a reserved address, to be fixed up by the linker
    - Maximum 64k offset, providing up to 8k 64-bit entries
  - If this table overflows, current linker mechanism introduces a branch to fixup code
  - This can be addressed through IPA, but non-IPA compilations pay heavy runtime cost

- New mechanism in collaboration with the AIX linker
  - Compiler generates a two instruction sequence: an add-immediate-shifted and a load
    - Low latency between these two instructions
  - Provides up to 2G of TOC data, up to 256M entries
  - Available under the option -qpic=large

- Combination with data local mechanism provides faster access to global data

- Similar mechanism implemented on Linux on Power

Debug improvements

- Debug levels
  - There is an intrinsic tradeoff between compiler optimization and debug transparency
  - Compiler optimizations hide program state from the debugger
    - Users have to choose between full debug at no-opt, or marginal debug at full opt

- Compiler to provide control over tradeoffs between optimization and debug
  - Debug levels: -g0 to -g9
    - -g1 minimal debug, maintain full performance
    - -g9 will provide full debug capability, at runtime performance cost
  - Expect better runtime performance from -g9 -O2 than -g -O0
  - Intermediate levels provide other levels of tradeoff
    - -O2 -g8 provide full debug, except no modification to user variables from debugger

XML Compiler Transformation Reports

- Generate compilation reports consumable by other tools
  - Enable better visualization and analysis of compiler information
  - Help users do manual performance tuning
  - Help automatic performance tuning through performance tool integration
- Unified report from all compiler subcomponents and analysis
  - Compiler options
  - Pseudo-sources
  - Compiler transformations, including missed opportunities
- Consistent support among Fortran, C/C++
- Controlled under option

-qlistfmt=[xml | html]=inlines	**generates inlining information**
-qlistfmt=[xml | html]=transform	**generates loop transformation information**
-qlistfmt=[xml | html]=data	**generates data reorganization information**
-qlistfmt=[xml | html]=pdf	**generates dynamic profiling information**
-qlistfmt=[xml | html]=all	**turns on all optimization content**
-qlistfmt=[xml | html]=none	**turns off all optimization content**

new

OpenMP Features

- Full OpenMP 3.1 compliance for XL C/C++ V12.1 and XLF V14.1 on AIX/Linux

- Major OpenMP 3.1 features include
  - New omp atomic extensions
    - update / read / write / capture
  - Support for min/max reductions on C/C++
    - Aligns C/C++ reductions with Fortran
  - Mergeable and final clauses on tasks
    - Provides fine-grain control over task creation to improve performance
  - OMP_PROC_BIND
    - Generic mechanism to bind threads to processors
    - Existing thread binding mechanisms in XL Compilers still supported

- Performance enhancements to reduce the overhead of parallel work initialization

Improved GCC affinity

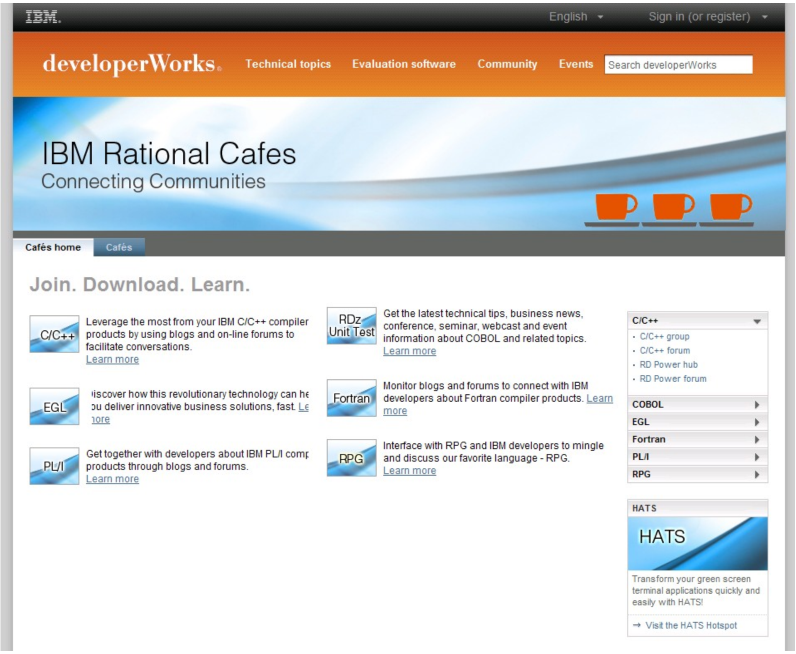
- Inline ASM
  - ASM is a gcc extension that allows mingling of machine assembly on C++ code
  - Not portable, not recommended unless there is no other option
  - New implementation is more robust and better mimics GCC behavior

- GCC builtins
  - Implement more gcc builtin functions
    - Particularly atomic access builtins

IBM Rational Cafes – Connecting Communities

- Accelerate your enterprise modernization efforts by becoming a member of the Cafe communities
- Ask questions, get free distance learning, browse the resources, attend user group webcasts, read the blogs, download trials, and share with others
- Cafes have forums, blogs, wikis, and more

- **Languages covered:** C/C++, COBOL, Fortran, EGL, PL/I, and RPG
- **Products covered:**
  - COBOL for AIX®
  - Enterprise COBOL for z/OS®
  - Enterprise PL/I for z/OS
  - Host Access Transformation Services
  - PL/I for AIX
  - Rational® Business Developer
  - Rational Developer for Power Systems Software™
  - Rational Developer for i for SOA Construction
  - Rational Developer for System z
  - Rational Developer for System z Unit Test
  - Rational Team Concert™
  - XL C for AIX
  - XL C/C++ for AIX/Linux ®
  - XL Fortran for AIX/Linux
  - XL C/C++ for z/VM
  - z/OS XL C/C++



Join. Download. Learn.

Leverage the most from your IBM C/C++ compiler products by using blogs and on-line forums to facilitate conversations. Learn more

Get the latest technical tips, business news, conference, seminar, webcast and event information about COBOL and related topics. Learn more

iscover how this revolutionary technology can he ou deliver innovative business solutions, fast. Le 1ore

Monitor blogs and forums to connect with IBM developers about Fortran compiler products. Learn more

Get together with developers about IBM PL/I comp products through blogs and forums. Learn more

Interface with RPG and IBM developers to mingle and discuss our favorite language - RPG. Learn more

Transform your green screen terminal applications quickly and easily with HATS!

→ Visit the HATS Hotspot

Become a member and join the conversation!

ibm.com/rational/café

facebook.com/IBMcompilers

@IBM_Compilers

compinfo@ca.ibm.com

Part 2 – Performance Tuning with XL Compilers

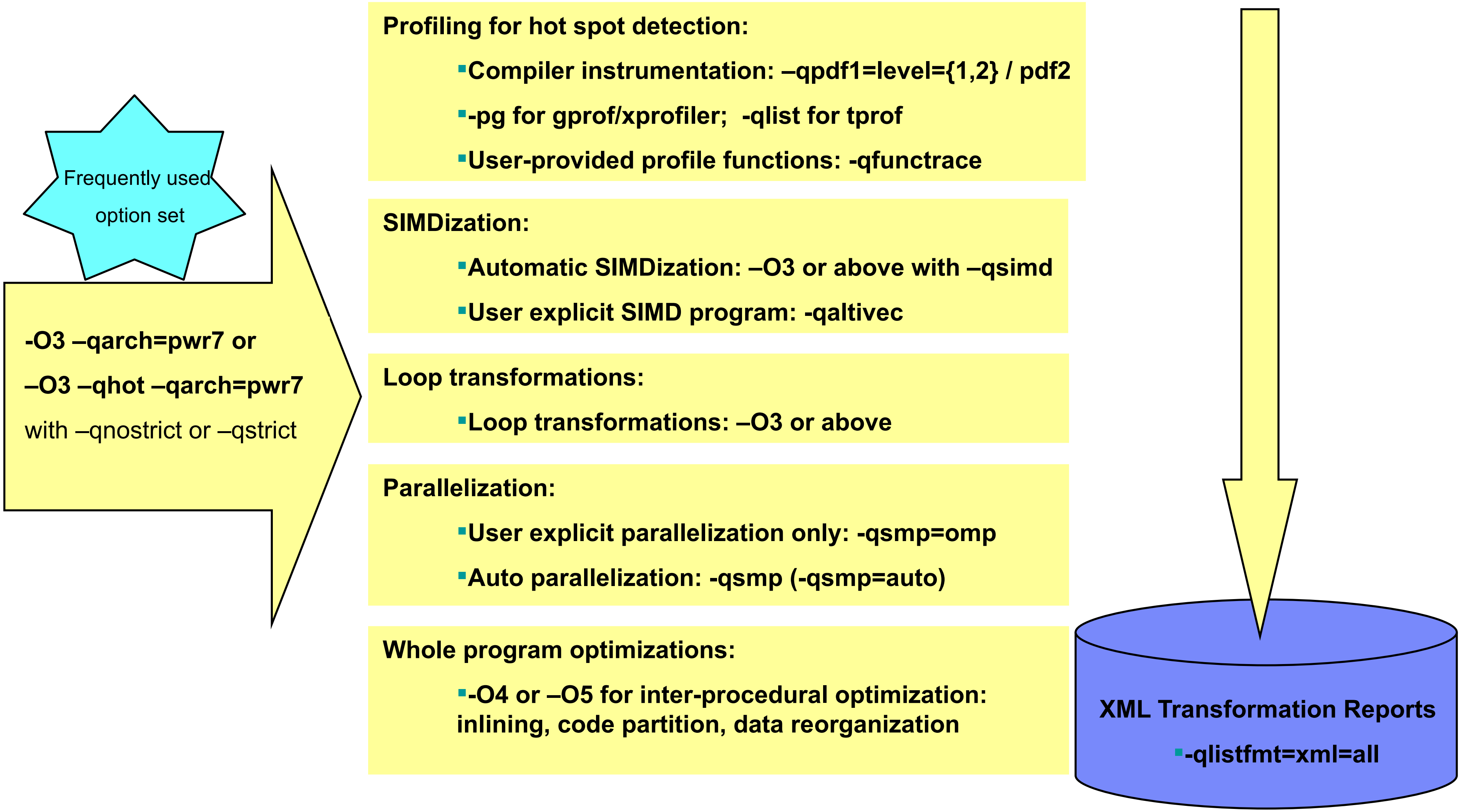
Performance Compiler Options

- Optimization levels:
  - -O0 to -O5
- High Order Transformations:
  - -qhot
- Interprocedural analysis:
  - -qipa or –O4 or –O5
- Profile directed feedback optimization:
  - -qpdf1/-qpdf2
- Target machine specification:
  - -qarch=pwr7 –qtune=pwr7 –qcache=auto
- Floating point options:
  - -qstrict=subopt, -qfloat=subopt
- Program behavior options:
  - -qassert=subopt, etc.
- Diagnostic options:
  - -qlist, -qreport, -qlistfmt, etc

Summary of Optimization Levels

- Noopt,-O0
  - Quick local optimizations
  - Keep the semantics of a program (-qstrict)
- -O2
  - Optimizations for the best combination of compile speed and runtime performance
  - Keep the semantics of a program (-qstrict)
- -O3
  - Equivalent to –O3 –qhot=level=0 –qnostrict
  - Focus on runtime performance at the expense of compilation time: loop transformations, dataflow analysis
  - May alter the semantics of a program (-qnostrict)
- -O3 –qhot
  - Equivalent to –O3 –qhot=level=1 –qnostrict
  - Perform aggressive loop transformations and dataflow analysis at the expense of compilation time
- -O4
  - Equivalent to –O3 –qhot=level=1 –qipa=level=1 -qnostrict
  - Aggressive optimization: whole program optimization; aggressive dataflow analysis and loop transformations
- -O5
  - Equivalent to –O3 –qhot=level=1 –qipa=level=2 -qnostrict
  - More aggressive optimization: more aggressive whole program optimization, more precise dataflow analysis and loop transformations

HPC Performance Tuning with XL Compilers



Frequently used option set

-O3 –qarch=pwr7 or

–O3 –qhot –qarch=pwr7

with –qnostrict or –qstrict

Profiling for hot spot detection:

- **Compiler instrumentation: –qpdf1=level={1,2} / pdf2**
- **-pg for gprof/xprofiler; -qlist for tprof**
- **User-provided profile functions: -qfunctrace**

SIMDization:

- **Automatic SIMDization: –O3 or above with –qsimd**
- **User explicit SIMD program: -qaltivec**

Loop transformations:

- **Loop transformations: –O3 or above**

Parallelization:

- **User explicit parallelization only: -qsmp=omp**
- **Auto parallelization: -qsmp (-qsmp=auto)**

Whole program optimizations:

- **-O4 or –O5 for inter-procedural optimization: inlining, code partition, data reorganization**

XML Transformation Reports

- **-qlistfmt=xml=all**

Pragmas (C/ C++)

- #pragma align ★
- #pragma alloca (C only)
- #pragma block_loop
- #pragma chars
- #pragma comment
- #pragma define, #pragma instantiate (C++ only)
- #pragma disjoint ★
- #pragma do_not_instantiate (C++ only)
- #pragma enum
- #pragma execution_frequency
- #pragma expected_value
- #pragma fini (C only)
- #pragma hashome (C++ only)
- #pragma ibm snapshot
- #pragma implementation (C++ only)
- #pragma info
- #pragma init (C only)
- #pragma ishome (C++ only)
- #pragma isolated_call
- #pragma langlvl (C only)
- #pragma leaves
- #pragma loopid
- #pragma map

- #pragma mc_func
- #pragma namemangling (C++ only)
- #pragma namemanglingrule (C++ only)
- #pragma nosimd ★
- #pragma novector ★
- #pragma object_model (C++ only)
- #pragma operator_new (C++ only)
- #pragma options
- #pragma option_override
- #pragma pack
- #pragma pass_by_value (C++ only)
- #pragma priority (C++ only)
- #pragma reachable
- #pragma reg_killed_by
- #pragma report (C++ only)
- #pragma simd_level ★
- #pragma STDC cx_limited_range
- #pragma stream_unroll
- #pragma strings
- #pragma unroll ★
- #pragma unrollandfuse
- #pragma weak
- #pragma ibm independent_loop

Parallel processing

- #pragma ibm critical (C only)
- #pragma ibm independent_calls (C only)
- #pragma ibm iterations (C only)
- #pragma ibm parallel_loop (C only)
- #pragma ibm permutation (C only)
- #pragma ibm schedule (C only)
- #pragma ibm sequential_loop (C only)
- #pragma omp atomic
- #pragma omp parallel
- #pragma omp for
- #pragma omp ordered
- #pragma omp parallel for
- #pragma omp section, #pragma omp sections
- #pragma omp parallel sections
- #pragma omp single
- #pragma omp master
- #pragma omp critical
- #pragma omp barrier
- #pragma omp flush
- #pragma omp threadprivate
- #pragma omp task
- #pragma omp taskwait

★ Frequently used

Frequently Used Pragmas/Directives/Attributes

- Dependency
  - #pragma ibm independent_loop
  - #pragma disjoint
- Frequency
  - #pragma execution_frequency
  - #pragma expected_value
  - #pragma ibm min_iterations
  - #pragma ibm max_iterations
  - #pragma ibm iterations
- Alignment
  - __alignx
  - __attribute__ {(aligned(16))}
- SIMDization
  - #pragma nosimd
  - #pragma simd_level
- Unroll
  - #pragma unroll

Summary

- Frequently used compiler option sets
  - -O3 –qarch=pwr7 –qtune=pwr7
  - -O3 –qhot –qarch=pwer7 –qtune=pwr7
- Frequently used compiler directives/pragmas
  - Dependency and alias analysis
  - Alignment
  - Frequency
  - Program behavior
  - Transformations

C++ Example using Eigen

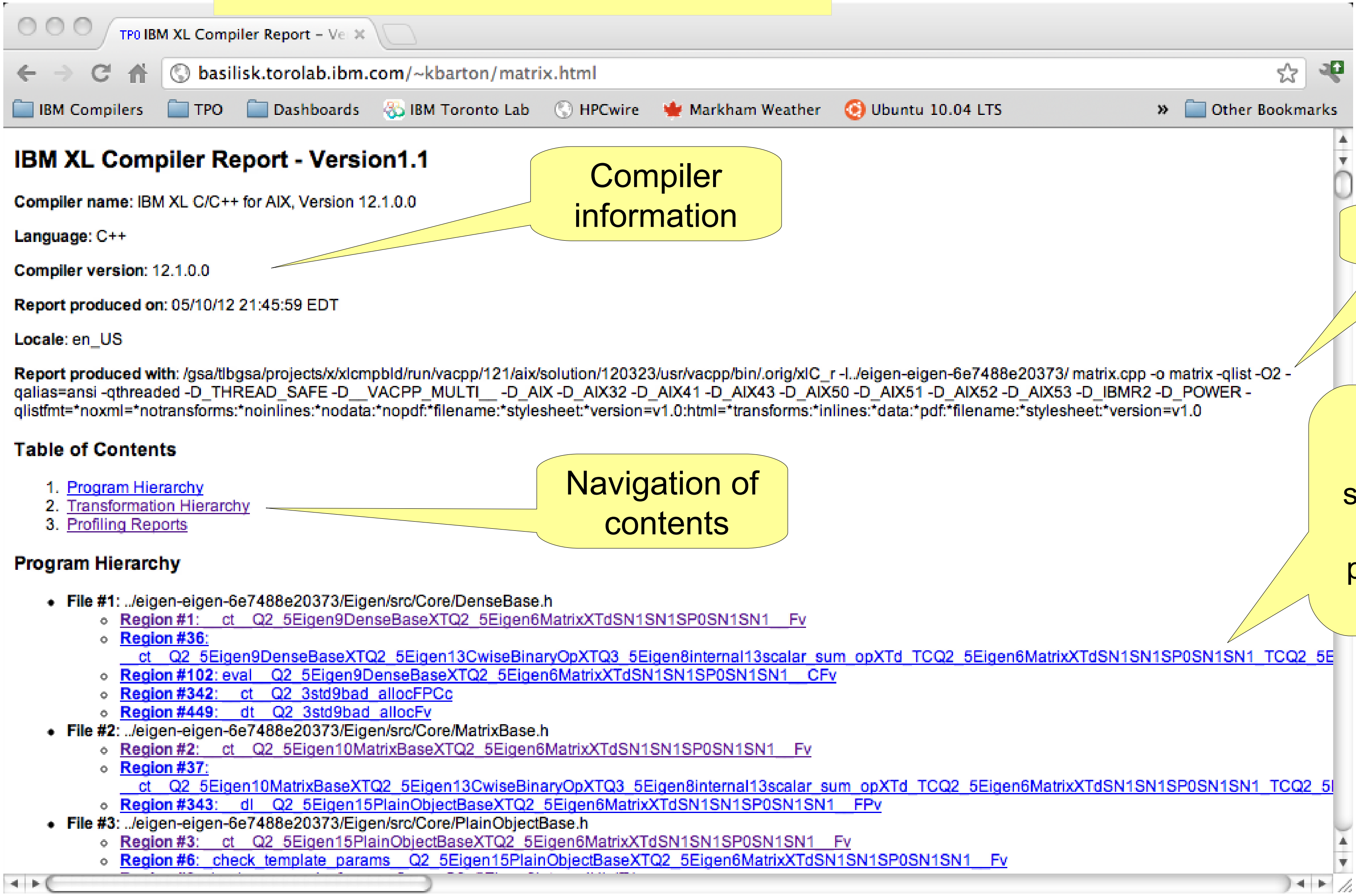
matrix.cpp

```
#include <iostream>
#include <Eigen/Dense>
#define ROWS 3
#define COLUMNS 3
using namespace Eigen;
int main() {
  MatrixXd a(ROWS,COLUMNS),
                 b(ROWS,COLUMNS),
                 res(ROWS,COLUMNS);
  for (int i=0; i<ROWS; i++) {
    for (int j=0; j<COLUMNS; j++) {
      a(i,j) = j+(i*COLUMNS);
      b(i,j) = a(i,j)*2;
    }
  }
  std::cout << "a: " << a << std::endl
               << "b: " << b << std::endl;
  res = a+b;
  std::cout << "res: " << res << std::endl;
  return 0;
}
```

- Create 3x3 matrices, *a* and *b*
- Initialize *a* and *b*
- Add *a* and *b* and put result in *res*

Transformation Reports

xlC_r matrix.cpp -o matrix -qlist -O2 -qlistfmt=html



IBM XL Compiler Report - Version1.1

Compiler name: IBM XL C/C++ for AIX, Version 12.1.0.0

Language: C++

Compiler version: 12.1.0.0

Report produced on: 05/10/12 21:45:59 EDT

Locale: en_US

Report produced with: /gsa/tlbgsa/projects/x/xlcmpbld/run/vacpp/121/aix/solution/120323/usr/vacpp/bin/.orig/xlC_r -I../eigen-eigen-6e7488e20373/ matrix.cpp -o matrix -qlist -O2 -qalias=ansi -qthreaded -D_THREAD_SAFE -D__VACPP_MULTI__ -D_AIX -D_AIX32 -D_AIX41 -D_AIX43 -D_AIX50 -D_AIX51 -D_AIX52 -D_AIX53 -D_IBMR2 -D_POWER -qlistfmt=*noxml=*notransforms:*noinlines:*nodata:*nopdf:*filename:*stylesheet:*version=v1.0:html=*transforms:*inlines:*data:*pdf:*filename:*stylesheet:*version=v1.0

Table of Contents

Program Hierarchy

- **File #1**: ../eigen-eigen-6e7488e20373/Eigen/src/Core/DenseBase.h
  - **Region #1**: __ct__Q2_5Eigen9DenseBaseXTQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1__Fv
  - **Region #36**: __ct__Q2_5Eigen9DenseBaseXTQ2_5Eigen13CwiseBinaryOpXTQ3_5Eigen8internal13scalar_sum_opXTd_TCQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1_TCQ2_5E
  - **Region #102**: eval__Q2_5Eigen9DenseBaseXTQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1__CFv
  - **Region #342**: __ct__Q2_3std9bad_allocFPCc
  - **Region #449**: __dt__Q2_3std9bad_allocFv
- **File #2**: ../eigen-eigen-6e7488e20373/Eigen/src/Core/MatrixBase.h
  - **Region #2**: __ct__Q2_5Eigen10MatrixBaseXTQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1__Fv
  - **Region #37**: __ct__Q2_5Eigen10MatrixBaseXTQ2_5Eigen13CwiseBinaryOpXTQ3_5Eigen8internal13scalar_sum_opXTd_TCQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1_TCQ2_5E
  - **Region #343**: __dl__Q2_5Eigen15PlainObjectBaseXTQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1__FPv
- **File #3**: ../eigen-eigen-6e7488e20373/Eigen/src/Core/PlainObjectBase.h
  - **Region #3**: __ct__Q2_5Eigen15PlainObjectBaseXTQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1__Fv
  - **Region #6**: _check_template_params__Q2_5Eigen15PlainObjectBaseXTQ2_5Eigen6MatrixXTdSN1SN1SP0SN1SN1__Fv

Compiler information

Options

Navigation of contents

Code from different source files – hyperlinks provide easy navigation

Transformation Hierarchy

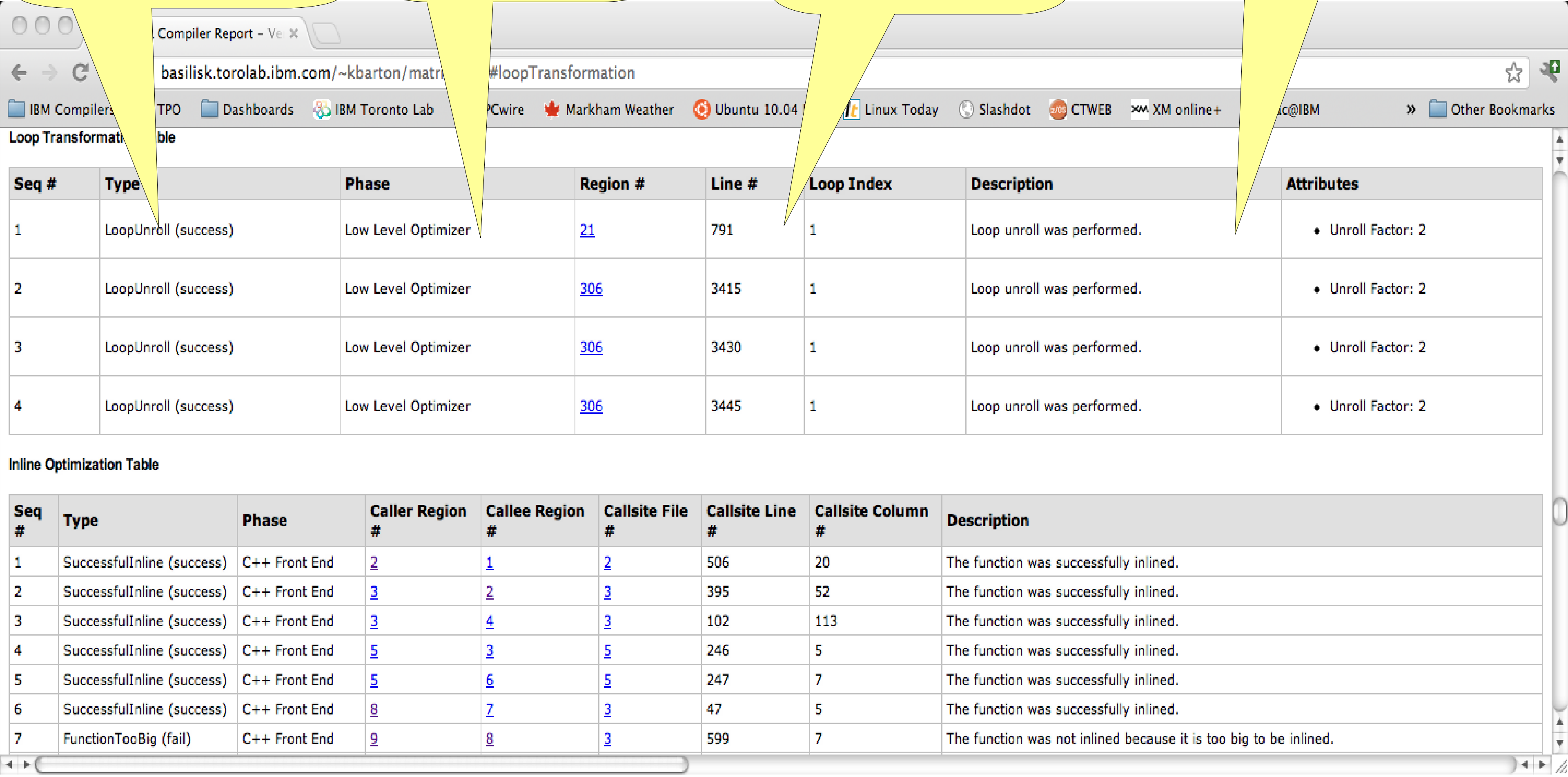
```
xlC_r matrix.cpp -o matrix -qlist -O2 -qlistfmt=html
```

Loop optimization

Component

Relevant source information

Success/failure and relevant information



Loop Transformati ble

Seq #	Type	Phase	Region #	Line #	Loop Index	Description	Attributes
1	LoopUnroll (success)	Low Level Optimizer	21	791	1	Loop unroll was performed.	• Unroll Factor: 2
2	LoopUnroll (success)	Low Level Optimizer	306	3415	1	Loop unroll was performed.	• Unroll Factor: 2
3	LoopUnroll (success)	Low Level Optimizer	306	3430	1	Loop unroll was performed.	• Unroll Factor: 2
4	LoopUnroll (success)	Low Level Optimizer	306	3445	1	Loop unroll was performed.	• Unroll Factor: 2

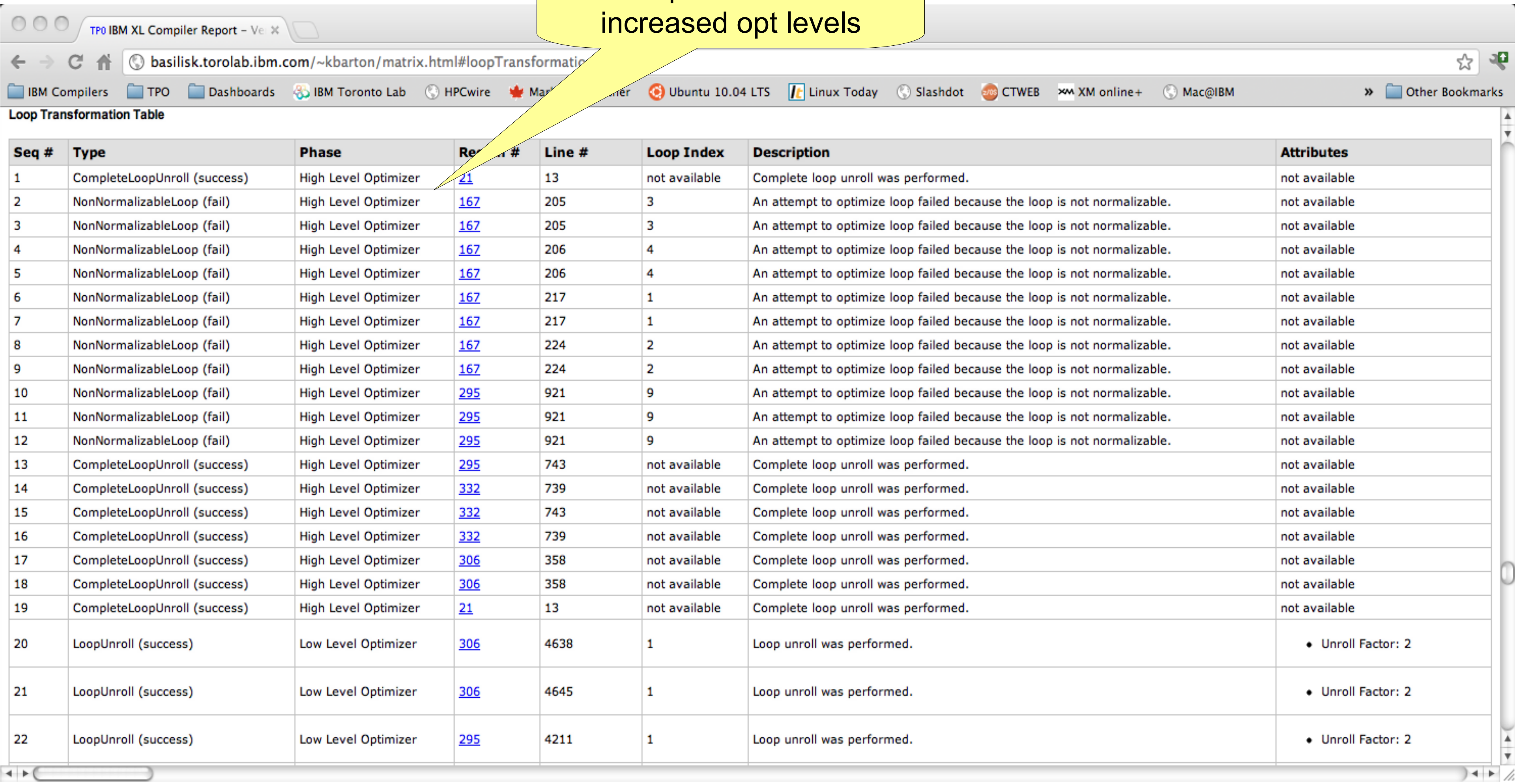
Inline Optimization Table

Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Callsite Column #	Description
1	SuccessfulInline (success)	C++ Front End	2	1	2	506	20	The function was successfully inlined.
2	SuccessfulInline (success)	C++ Front End	3	2	3	395	52	The function was successfully inlined.
3	SuccessfulInline (success)	C++ Front End	3	4	3	102	113	The function was successfully inlined.
4	SuccessfulInline (success)	C++ Front End	5	3	5	246	5	The function was successfully inlined.
5	SuccessfulInline (success)	C++ Front End	5	6	5	247	7	The function was successfully inlined.
6	SuccessfulInline (success)	C++ Front End	8	7	3	47	5	The function was successfully inlined.
7	FunctionTooBig (fail)	C++ Front End	9	8	3	599	7	The function was not inlined because it is too big to be inlined.

Transformation Hierarchy

```
xlC_r matrix.cpp -o matrix -qlist -O3 -qhot -qlistfmt=html
```

More optimizations with increased opt levels



Loop Transformation Table

Seq #	Type	Phase	Re #	Line #	Loop Index	Description	Attributes
1	CompleteLoopUnroll (success)	High Level Optimizer	21	13	not available	Complete loop unroll was performed.	not available
2	NonNormalizableLoop (fail)	High Level Optimizer	167	205	3	An attempt to optimize loop failed because the loop is not normalizable.	not available
3	NonNormalizableLoop (fail)	High Level Optimizer	167	205	3	An attempt to optimize loop failed because the loop is not normalizable.	not available
4	NonNormalizableLoop (fail)	High Level Optimizer	167	206	4	An attempt to optimize loop failed because the loop is not normalizable.	not available
5	NonNormalizableLoop (fail)	High Level Optimizer	167	206	4	An attempt to optimize loop failed because the loop is not normalizable.	not available
6	NonNormalizableLoop (fail)	High Level Optimizer	167	217	1	An attempt to optimize loop failed because the loop is not normalizable.	not available
7	NonNormalizableLoop (fail)	High Level Optimizer	167	217	1	An attempt to optimize loop failed because the loop is not normalizable.	not available
8	NonNormalizableLoop (fail)	High Level Optimizer	167	224	2	An attempt to optimize loop failed because the loop is not normalizable.	not available
9	NonNormalizableLoop (fail)	High Level Optimizer	167	224	2	An attempt to optimize loop failed because the loop is not normalizable.	not available
10	NonNormalizableLoop (fail)	High Level Optimizer	295	921	9	An attempt to optimize loop failed because the loop is not normalizable.	not available
11	NonNormalizableLoop (fail)	High Level Optimizer	295	921	9	An attempt to optimize loop failed because the loop is not normalizable.	not available
12	NonNormalizableLoop (fail)	High Level Optimizer	295	921	9	An attempt to optimize loop failed because the loop is not normalizable.	not available
13	CompleteLoopUnroll (success)	High Level Optimizer	295	743	not available	Complete loop unroll was performed.	not available
14	CompleteLoopUnroll (success)	High Level Optimizer	332	739	not available	Complete loop unroll was performed.	not available
15	CompleteLoopUnroll (success)	High Level Optimizer	332	743	not available	Complete loop unroll was performed.	not available
16	CompleteLoopUnroll (success)	High Level Optimizer	332	739	not available	Complete loop unroll was performed.	not available
17	CompleteLoopUnroll (success)	High Level Optimizer	306	358	not available	Complete loop unroll was performed.	not available
18	CompleteLoopUnroll (success)	High Level Optimizer	306	358	not available	Complete loop unroll was performed.	not available
19	CompleteLoopUnroll (success)	High Level Optimizer	21	13	not available	Complete loop unroll was performed.	not available
20	LoopUnroll (success)	Low Level Optimizer	306	4638	1	Loop unroll was performed.	• Unroll Factor: 2
21	LoopUnroll (success)	Low Level Optimizer	306	4645	1	Loop unroll was performed.	• Unroll Factor: 2
22	LoopUnroll (success)	Low Level Optimizer	295	4211	1	Loop unroll was performed.	• Unroll Factor: 2

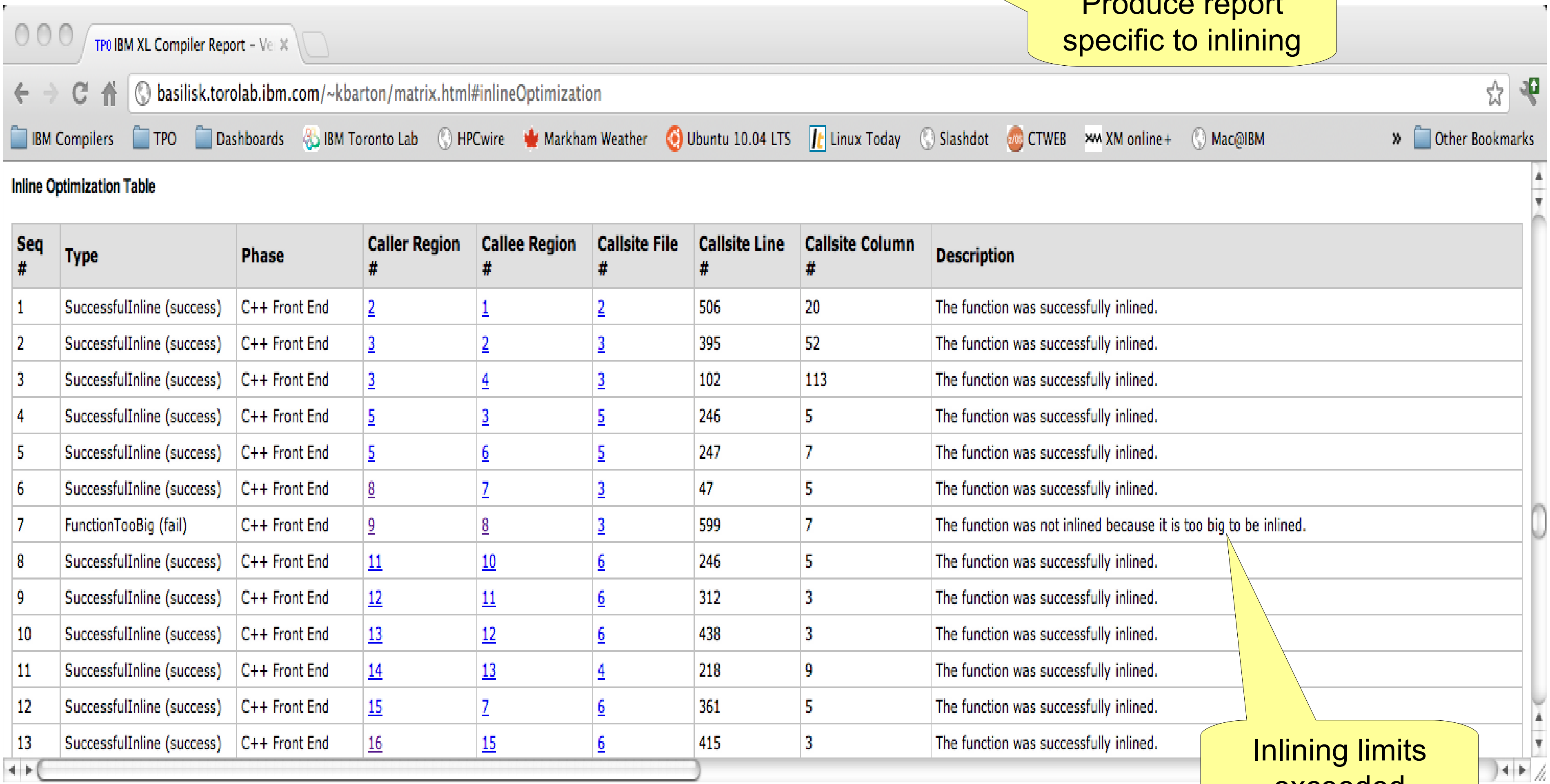
Inlining

- The XL Compilers perform inlining in many places
  - C++ FE
    - Small methods
    - inline and always_inline
    - Some exception-handling capabilities
    - Within a single source file
  - High-level Optimizer
    - Larger methods
    - Will inline within a source file (-O3, -qhot) and across source files (-O4, -O5, -qipa)
    - Inline across languages (-O4, -O5, -qipa)
  - Low-level optimizer
    - Larger methods
    - Within a source file
    - Enabled at -O2 and higher
- Transformation reports will show inlining results for each component

Inlining

```
xlC_r matrix.cpp -o matrix -qlist -O2 -qlistfmt=html=inlines
```

Produce report specific to inlining



Inline Optimization Table

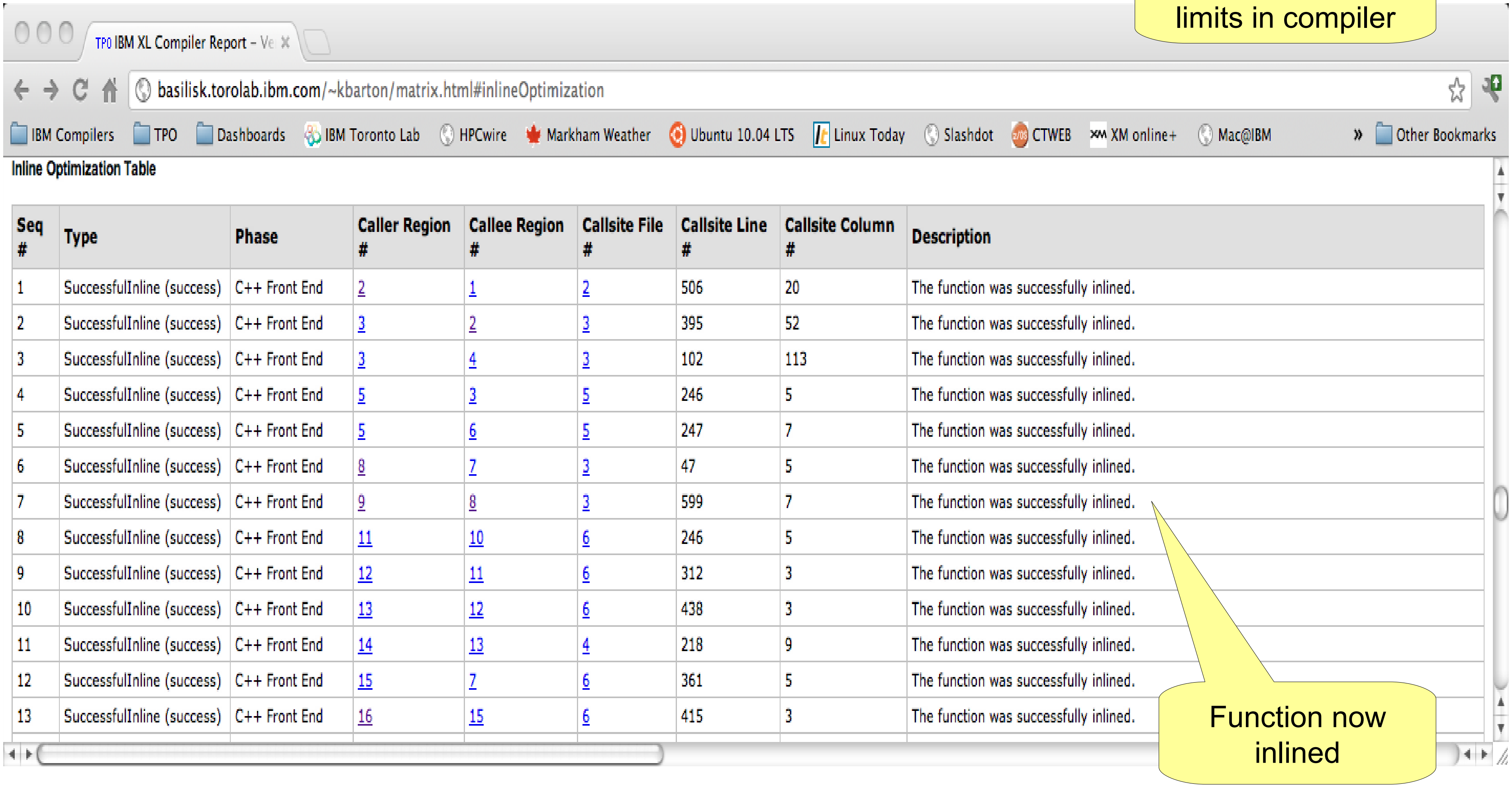
Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Callsite Column #	Description
1	SuccessfulInline (success)	C++ Front End	2	1	2	506	20	The function was successfully inlined.
2	SuccessfulInline (success)	C++ Front End	3	2	3	395	52	The function was successfully inlined.
3	SuccessfulInline (success)	C++ Front End	3	4	3	102	113	The function was successfully inlined.
4	SuccessfulInline (success)	C++ Front End	5	3	5	246	5	The function was successfully inlined.
5	SuccessfulInline (success)	C++ Front End	5	6	5	247	7	The function was successfully inlined.
6	SuccessfulInline (success)	C++ Front End	8	7	3	47	5	The function was successfully inlined.
7	FunctionTooBig (fail)	C++ Front End	9	8	3	599	7	The function was not inlined because it is too big to be inlined.
8	SuccessfulInline (success)	C++ Front End	11	10	6	246	5	The function was successfully inlined.
9	SuccessfulInline (success)	C++ Front End	12	11	6	312	3	The function was successfully inlined.
10	SuccessfulInline (success)	C++ Front End	13	12	6	438	3	The function was successfully inlined.
11	SuccessfulInline (success)	C++ Front End	14	13	4	218	9	The function was successfully inlined.
12	SuccessfulInline (success)	C++ Front End	15	7	6	361	5	The function was successfully inlined.
13	SuccessfulInline (success)	C++ Front End	16	15	6	415	3	The function was successfully inlined.

Inlining limits exceeded

Increasing inline thresholds

```
xlC_r matrix.cpp -o matrix -qlist -O2 -qlistfmt=html=inlines -qinline=level=10
```

Increase inlining limits in compiler



Inline Optimization Table

Seq #	Type	Phase	Caller Region #	Callee Region #	Callsite File #	Callsite Line #	Callsite Column #	Description
1	SuccessfulInline (success)	C++ Front End	2	1	2	506	20	The function was successfully inlined.
2	SuccessfulInline (success)	C++ Front End	3	2	3	395	52	The function was successfully inlined.
3	SuccessfulInline (success)	C++ Front End	3	4	3	102	113	The function was successfully inlined.
4	SuccessfulInline (success)	C++ Front End	5	3	5	246	5	The function was successfully inlined.
5	SuccessfulInline (success)	C++ Front End	5	6	5	247	7	The function was successfully inlined.
6	SuccessfulInline (success)	C++ Front End	8	7	3	47	5	The function was successfully inlined.
7	SuccessfulInline (success)	C++ Front End	9	8	3	599	7	The function was successfully inlined.
8	SuccessfulInline (success)	C++ Front End	11	10	6	246	5	The function was successfully inlined.
9	SuccessfulInline (success)	C++ Front End	12	11	6	312	3	The function was successfully inlined.
10	SuccessfulInline (success)	C++ Front End	13	12	6	438	3	The function was successfully inlined.
11	SuccessfulInline (success)	C++ Front End	14	13	4	218	9	The function was successfully inlined.
12	SuccessfulInline (success)	C++ Front End	15	7	6	361	5	The function was successfully inlined.
13	SuccessfulInline (success)	C++ Front End	16	15	6	415	3	The function was successfully inlined.

Function now inlined

Adjustments to inline heuristics

- -qinline=level=#
  - Compiler option to adjust internal thresholds used by inlining heuristics
  - Default level is 5
  - 6-10 increase inlining
  - 1-4 decrease inlining
- -qinline+ (C and Fortran only)
  - Compiler option to specify functions to be inlined
  - Compiler still uses internal thresholds, so inlining is not guaranteed
  - Equivalent -qinline- option to prevent functions from being inlined
- inline keyword (C99/C++ only)
  - Modify source code to indicate preferences for inlining
  - Compiler still uses internal thresholds, so inlining is not guaranteed
- __attribute__((always_inline)) pragma (IBM,GCC, C/C++ only)
  - Indicate a method should always be inlined
  - Compiler will always inline these methods – they are treated independently of internal thresholds

Pointer Aliasing

- Pointer aliasing can be a major impediment for compiler optimizations
  - If the compiler cannot prove that two pointers do not represent the same storage, it must assume they do
  - This can hinder optimizations

alias.c

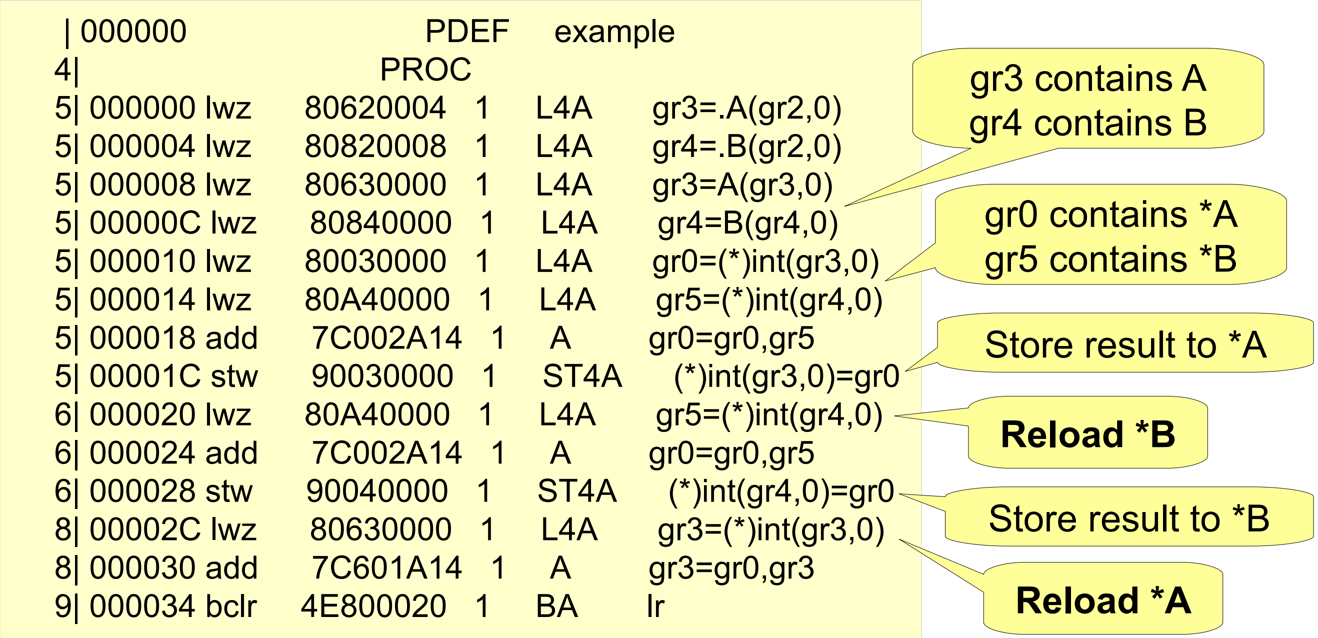
```
int *A;
int *B;

int example() {
  *A += *B;
  *B += *A;

  return *A + *B;
}
```

```
xlc_r -c -O2 -qlist alias.c
```

alias.lst



```
 | 000000                        PDEF    example
4|                            PROC
5| 000000 lwz     80620004   1    L4A      gr3=.A(gr2,0)
5| 000004 lwz     80820008   1    L4A      gr4=.B(gr2,0)
5| 000008 lwz     80630000   1    L4A      gr3=A(gr3,0)
5| 00000C lwz     80840000   1    L4A      gr4=B(gr4,0)
5| 000010 lwz     80030000   1    L4A      gr0=(*)int(gr3,0)
5| 000014 lwz     80A40000   1    L4A      gr5=(*)int(gr4,0)
5| 000018 add     7C002A14   1    A        gr0=gr0,gr5
5| 00001C stw     90030000   1    ST4A     (*)int(gr3,0)=gr0
6| 000020 lwz     80A40000   1    L4A      gr5=(*)int(gr4,0)
6| 000024 add     7C002A14   1    A        gr0=gr0,gr5
6| 000028 stw     90040000   1    ST4A     (*)int(gr4,0)=gr0
8| 00002C lwz     80630000   1    L4A      gr3=(*)int(gr3,0)
8| 000030 add     7C601A14   1    A        gr3=gr0,gr3
9| 000034 bclr    4E800020   1    BA       lr
```

gr3 contains A
gr4 contains B

gr0 contains *A
gr5 contains *B

Store result to *A

Reload *B

Store result to *B

Reload *A

Restricted Pointer Support

- The *restrict* keyword tells the compiler that a secific pointer is the only one that points to this data

restrict.c

```
int * restrict A;
int * restrict B;

int example() {
  *A += *B;
  *B += *A;

  return *A +
*B;
}
```

```
xlc_r -c -O2 -qlist restrict.c
```

restrict.c

```
 | 000000                    PDEF    example
4|                       PROC
5| 000000 lwz     80620004   1    L4A      gr3=.A(gr2,0)
5| 000004 lwz     80820008   1    L4A      gr4=.B(gr2,0)
5| 000008 lwz     80630000   1    L4A      gr3=A(gr3,0)
5| 00000C lwz     80840000   1    L4A      gr4=B(gr4,0)
5| 000010 lwz     80030000   1    L4A      gr0=(*)A{int}(gr3,0)
5| 000014 lwz     80A40000   1    L4A      gr5=(*)B{int}(gr4,0)
5| 000018 add     7C002A14   1    A        gr0=gr0,gr5
5| 00001C stw     90030000   1    ST4A     (*)A{int}(gr3,0)=gr0
6| 000020 add     7CA50214   1    A        gr5=gr5,gr0
6| 000024 stw     90A40000   1    ST4A     (*)B{int}(gr4,0)=gr5
8| 000028 add     7C602A14   1    A        gr3=gr0,gr5
9| 00002C bclr    4E800020   1    BA       lr
```

gr3 contains A
gr4 contains B

gr0 contains *A
gr5 contains *B

Store result to *A

Store result to *B

Restricted Pointers

Restricted parameter pointer:

```
void  function(float * restrict a1. float *restrict a2} {
 for  ( int  i=0; i<n;  i++)  {
     a1[i] = a2[i];
  }
}
```

Block scope restricted pointer:

```
  float * restrict a1 = A1; float * restrict a2 = A2;
  for  ( int  i=0; i<n;  i++)  {
     a1[i] = a2[i];
  }
```

Multiple level restricted pointer:

```
  float * restrict *restrict * restrict aa1 = AA1;
  float  * restrict *restrict * restrict bb1 = BB1;
  for (int k=0; k<n3; k++) {
    for (int j=0; j<n2; j++) {
       for (int i=0; i< n1; i++) {
           aa1[i][j][k] = bb1[i][j][k];
  } } }
```

- Determine if two different pointers are being used to reference different objects

- Refine aliasing to expose optimization opportunities;

Other Tuning Options for C++ codes

- Exception handling
  - If you are not using exception handling, use the -qnoeh option
    - Assertion that no exceptions will be thrown at runtime
  - Can improve optimization opportunities

- Malloc tuning
  - On AIX, there are several different algorithms for memory allocation
  - For C++, MALLOCOPTIONS=pool will frequently improve performance

  http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.genprogc/doc/genprogc/sys_mem_alloc.htm

- Data page size
  - Increasing data page size can also improve performance
  - -bdatapsize:64k

Debugging Optimized Code

- Debug levels
  - There is an intrinsic tradeoff between compiler optimization and debug transparency
  - Compiler optimizations hide program state from the debugger
    - Users have to choose between full debug at no-opt, or marginal debug at full opt

- Compiler to provide control over tradeoffs between optimization and debug
  - Debug levels: -g0 to -g9
    - -g1 minimal debug, maintain full performance
    - -g9 will provide full debug capability, at runtime performance cost
  - Expect better runtime performance from -g9 -O2 than -g -O0
  - Intermediate levels provide other levels of tradeoff
    - -O2 -g8 provide full debug, except no modification to user variables from debugger

Debugging Optimized Code

```
int example(int x, int y) {
  int t1, t2;
  int result;

  t1 = x*y-1;
  t2 = x*y*3;

  result = t1+t2;
  return result;
}


int main() {
  int res = example(4,5);

  printf("res=%d\n", res);

  return 0;
}
```

```
xlC_r -O2 -g -qlist debug.c
```

```
(dbx) listi example
0x10000800 (example)      7c0321d6      mullw   r0,r3,r4
0x10000804 (example+0x4)  5404103a        sli   r4,r0,0x2
0x10000808 (example+0x8)  3864ffff       addi   r3,-1(r4)
0x1000080c (example+0xc)  4e800020        blr
```

All locals optimized away

```
(dbx) stop in example
(dbx) run
[1] stopped in example at line 6 ($t1)
    6     t1 = x*y-1;
(dbx) print t1
reference through nil pointer
(dbx) step
stopped in example at line 7 ($t1)
    7     t2 = x*y*3;
(dbx) print t1
reference through nil pointer
(dbx) step
stopped in example at line 9 ($t1)
    9     result = t1+t2;
(dbx) print t2
reference through nil pointer
(dbx) step
stopped in example at line 11 ($t1)
   11   }
(dbx) print result
reference through nil pointer
```

Unable to print local variables

Debugging Optimized Code

```
int example(int x, int y) {
  int t1, t2;
  int result;

  t1 = x*y-1;
  t2 = x*y*3;

  result = t1+t2;
  return result;
}


int main() {
  int res = example(4,5);

  printf("res=%d\n", res);

  return 0;
}
```

```
xlC_r -O2 -g8 -qlist debug.c
```

```
(dbx) stop in example
[1] stop in example
(dbx) run
[1] stopped in example at line 6 ($t1)
    6     t1 = x*y-1;
(dbx) print t1
804398288
(dbx) step
stopped in example at line 7 ($t1)
    7     t2 = x*y*3;
(dbx) print t1
19
(dbx) step
stopped in example at line 9 ($t1)
    9     result = t1+t2;
(dbx) print t2
60
(dbx) step
stopped in example at line 10 ($t1)
   10     return result;
(dbx) print result
79
(dbx) step
stopped in example at line 11 ($t1)
   11   }
```

Local variables correctly displayed

Debugging Optimized Code

```
int example(int x, int y) {
  int t1, t2;
  int result;

  t1 = x*y-1;
  t2 = x*y*3;

  result = t1+t2;
  return result;
}


int main() {
  int res = example(4,5);

  printf("res=%d\n", res);

  return 0;
}
```

```
xlC_r -O2 -g9 -qlist debug.c
```

```
(dbx) stop in example
[1] stop in example
(dbx) run
[1] stopped in example at line 6 ($t1)
    6     t1 = x*y-1;
(dbx) step
stopped in example at line 7 ($t1)
    7     t2 = x*y*3;
(dbx) print t1
19
(dbx) assign t1=10
(dbx) step
stopped in example at line 9 ($t1)
    9     result = t1+t2;
(dbx) assign t2=20
(dbx) step
stopped in example at line 10 ($t1)
   10     return result;
(dbx) print result
30
(dbx) step
stopped in example at line 11 ($t1)
   11   }
```

Local variables modified in debugger

Debugging Optimized Code

- Performance
  - Varies across benchmarks tested

- -g8 Performance
  - noopt -g vs -O2 -g8: 1.42x to 8.14x improvement (average 3.14x improvement)
  - O2 -g8 vs -O2: 53% to 95% of -O2 performance (average 80% of -O2 performance)

- -g9 Performance
  - noopt -g vs -O2 -g9: 1.1x to 3.54x improvement
  - O2 -g9 vs -O2: 15% to 77% of -O2 performance (average 40% of -O2 performance)

XLSMPOPTS Environment Variable

- XLSMPOPTS environment variable allows you to tune runtime behaviour of OpenMP and autoparallel programs

- Some suboptions of interest:
  - **spins** and **yields** to define the behaviour of idle threads
    - By setting spins=0:yields=0 idle threads will busy wait
  - Thread binding using **startproc** and **stride** suboptions, or new **bind** suboption
  - **schedule** to define the runtime scheduling algorithm used for parallel loops (static, dynamic, guided)
    - Note that the default schedule has changed from *runtime* to *auto* in V11/V13

http://pic.dhe.ibm.com/infocenter/comphelp/v121v141/index.jsp?topic=%2Fcom.ibm.xlc121.aix.doc%2Fcompiler_ref%2Fenv_var_xlsmpopts.html

Thread binding using resource sets (AIX)

- Alternative method of binding threads to CPUs

- Advantages over existing mechanism:
  1. Ability to adjust granularity of binding based on resource sets (proc, MCM, *etc.*)
  2. Allows applications to stop and then resume over a checkpoint without losing the thread binding configuration

Thread binding using resource sets

- System Detail Level (SDL)
  - MCM
  - L2CACHE
  - PROC_CORE
  - PROC

- New suboptions for XLSMPOPTS:

  bind=SDL=n1,n2,n3

  n1=start resource
  n2=number of resources
  n3=stride

  bindlist=SLD=*i0*,*i1*,…,*ix*

http://pic.dhe.ibm.com/infocenter/comphelp/v121v141/index.jsp?topic=%2Fcom.ibm.xlc121.aix.doc%2Fcompiler_ref%2Fenv_var_xlsmpopts.html

IBM align and iteration count directives

```
 module mod
     real :: x(500), y(500), z(500)
!ibm* align(16, x, y, z)
     end module
     subroutine partial_sum(m, n)
     use mod
     integer, intent(in) :: m, n
!ibm* assert(itercnt(40))
ibm* assert(itercnt(120))
     do i=m, n
       z(i) = x(i) + 1.37*y(i)
     enddo
     end subroutine
```

Guide the compiler to align arrays x, y, and z to 16 bytes

- Avoid cache conflicts and false sharing
- Expose SIMDization opportunity

The frequently used loop iteration counts are 40 and 120

- Guide the compilers during profitability analysis
- Expose loop optimization opportinities

Data prefetching

- POWER7 prefetch engine supports up to 12 data streams
- POWER7 provides fine grained software control to specify data stream type, stream length, stream stride, prefetch depth
- Automatic data prefetch insertion at optimization level -O3 -qhot or above
  - More aggressive exploitation under option -qprefetch=aggressive
  - Global analysis for coarse grained prefetch engine control at optimization level -O5
- -qlistfmt=xml=transformations (-qlistfmt=html=transformations) generates data prefetching information in xml (html)

Data prefetch and Cache Control Built-in functions

- Transient cache line touch

```
void __dcbtt(void *address);
void __dcbtstt (void * address);
```

- Partial cache line touch

```
void __partial_dcbt(void *address);
```

- Stride-N stream prefetch

```
void __protected_stream_stride(offset, stride, stream_ID);
```

- Transient stream prefetch

```
void __transient_protected_stream_count_depth(unit_count,
                                               depth, stream_ID)
void __transient_unlimited_protected_stream_depth(prefetch_depth,
                                               stream_ID)
```

Example of POWER7 Data Prefetching Insertion

Store stream prefetch for array a;

transient stream prefetch for array b

```
__protected_store_stream_set(FORWARD, &a, 11);
__protected_stream_count_depth(n*sizeof(double)/128, DEEPER, 11);
__protected_stream_set(FORWARD,  &b, 0);
__transient_protected_stream_count_depth(n*sizeof(double)/128, DEEPER, 0);
__eieio();
__protected_stream_go();
 for (i=0; i< n; i++) {
   a[i] = b[i] + ...;
 }
```

Stream direction

Stream id

Stream length

Prefetch depth

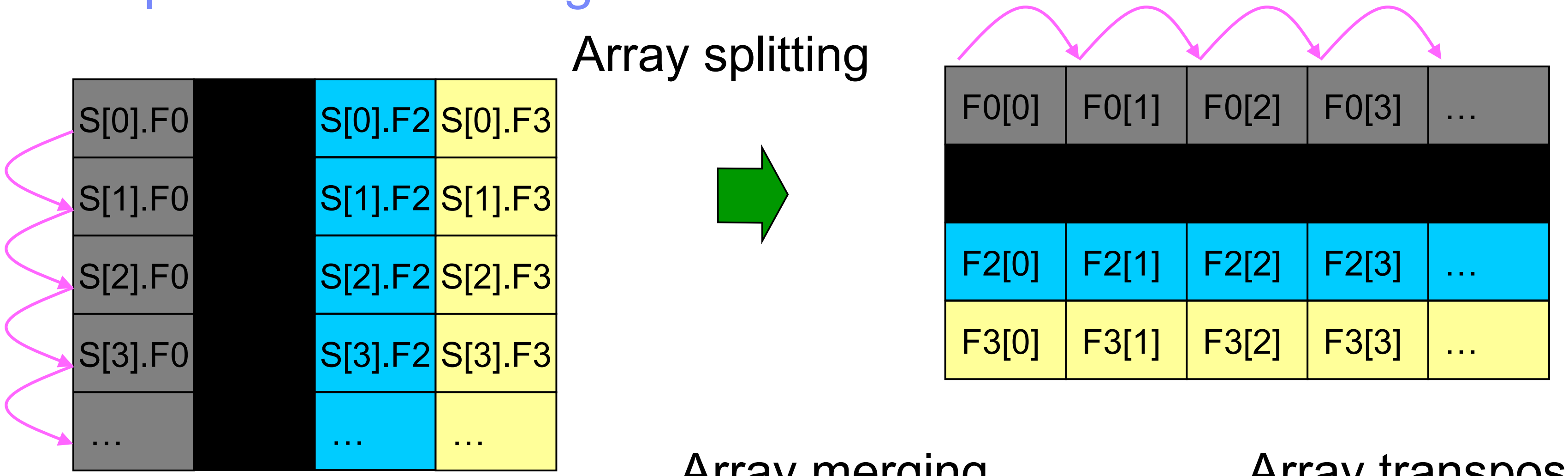
Start stream prefetch

Data Reorganization

- Reshape data layout to reduce memory latency, enhance cache utilization and memory bandwidth.
- Data reorganization transformations enabled at O5
  - Data splitting
  - Data interleaving
  - Data transposing
  - Data merging
  - Data grouping
  - Data compressing
  - Data padding
- -qlistfmt=xml=data (-qlistfmt=html=data) generates data reorganization transformation information in xml (html)

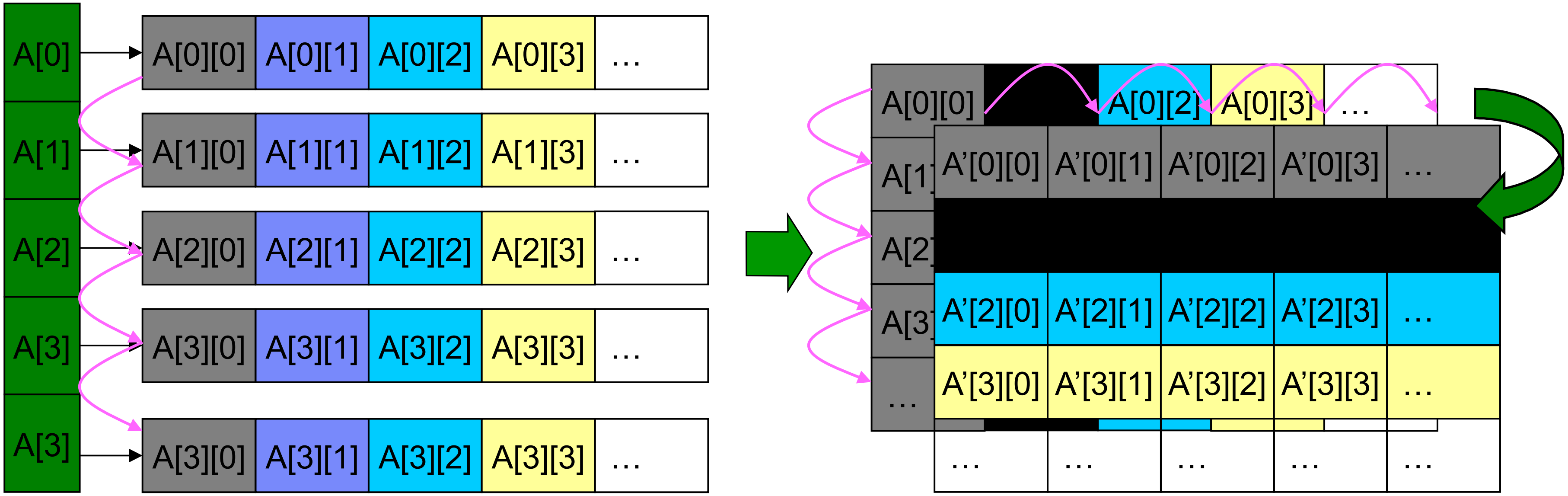
Examples of Data Reorganization

Array splitting

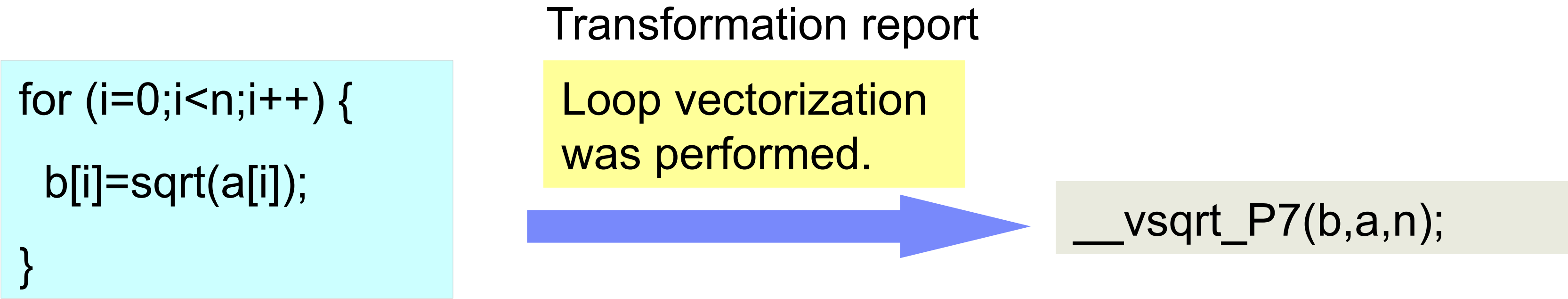


Array merging

Array transposing



MASS enhancements and Auto-vectorization



Transformation report

```
for (i=0;i<n;i++) {
  b[i]=sqrt(a[i]);
}
```

Loop vectorization was performed.

```
__vsqrt_P7(b,a,n);
```

- MASS enhancements for POWER7
  - **POWER7 vector MASS library** (libmassvp7.a)
    - Internally exploit VSX instructions

      SP: average speedup of 1.99 vs Power5 MASSV

      DP: average speedup of 1.27 vs Power5 MASSV
  - **POWER7 SIMD MASS library** (libmass_simdp7.a)
    - Tuned math routines operating on vector data types
    - Over 35 frequently used mathematical functions
    - Both simple and double precision
    - To be used in conjunction with explicit SIMD programming
- Auto-vectorization at optimization level –O3 or above
- -qstrict=vectorprecision to maintain precision over all loop iterations

Explicit SIMD programming for POWER7 Enabled under -qaltivec

- Successor to altivec programming extensions on POWER6/PPC970

– **Altivec data types**	**16-byte vectors**
vector char	16 elements
vector short	8 elements
vector pixel	8 elements
vector int	4 elements
vector float	4 elements
– **VSX Altivec extensions**	**16-byte vectors**
vector double	2 elements
vector long long	2 elements

- Altivec built-in functions extended to new data types

  vec_add(vector double, vector double),
  vec_sub(vector long long, vector long long),

- New vector operations: vec_mul, vec_div, …

- Unaligned load and store operations
  - Altivec truncating loads/stores still available: vec_ld, vec_st
  - New non-truncating loads/stores: vec_xld2, vec_xstd2

VSX Example

```
#include <math.h>
#include <altivec.h>
extern double x[1000],y[1000],z[1000];
void sub(){
  int i;
  vector double x2,y2,z2;
  for(i=0;i<1000;i+=2) {
     x2=vec_xld2(0,&x[i]);
     y2=vec_xld2(0,&y[i]);
     z2=vec_sqrt(vec_add(
        vec_mul(x2,x2),vec_mul(y2,y2)));
     vec_xstd2(z2,0,&z[i]);
}}
```

```
subroutine sub(x,y,z)
   real*8 x(*),y(*),z(*)
   vector(real(8)) x2,y2,z2
    do i=1,1000,2
      x2=vec_xld2(0,x(i))
      y2=vec_xld2(0,y(i))
      z2=vec_sqrt(vec_add(
           vec_mul(x2,x2),vec_mul(y2,y2)));
      call vec_xstd2(z2,0,z(i))
    enddo
   return
end
```

xlc –O3 –qarch=pwr7 –qvecnvol –qaltivec py.c

xlf90 –O3 –qarch=pwr7 –qvecnvol py.f

Compiler Listing:

```
 4|                                 CL.5:
 0| 000050 addi     38840010   1      AI        gr4=gr4,16
 7| 000054 xvsqrtdp F060032C   1      VDFSQRT   vs3=vs0,fcr
 7| 000058 xvcpsgnd F0021780   1      LRVS      vs0=vs2
 7| 00005C xvmuldp  F0442380   1      VDFM      vs2=vs4,vs4,fcr
 6| 000060 lxvd2x   7C840698   1      VLQD      vs4=y[]@gr612->.y(gr4,gr0,0)
 7| 000064 xvmaddad F0010B08   1      VDFMA     vs0=vs0,vs1,vs1,fcr
 5| 000068 lxvd2x   7C250698   1      VLQD      vs1=x[]@gr609->.x(gr5,gr0,0)
 0| 00006C addi     38A50010   1      AI        gr5=gr5,16
 8| 000070 stxvd2x  7C630798   1      VSTQD     z[]@gr621->.z(gr3,gr0,0)=vs3
 0| 000074 addi     38630010   1      AI        gr3=gr3,16
 0| 000078 bc       4320FFD8   1      BCT       ctr=CL.5,,100,0
```

Automatic SIMDization

- Automatic SIMDization for VMX and VSX
  - Supports data types of INTEGER, UNSIGNED, REAL and COMPLEX

- Features:
  - Basic block level SIMDizaton
  - Loop level aggregation
  - Data conversion, reduction
  - Loop with limited control flow
  - Automatic SIMDization with -qstrict (VSX) and -qnostrict
  - Support of unaligned vector memory accesses (VSX)
  - Automatic SIMDization enabled at -O3 -qsimd

AutoSIMD: VSX example

xlf90 –O3 –qhot **–qstrict** –qarch=pwr7 –qsimd –qvecnvol -qlist py.f

py.f:

```
subroutine sub(x,y,z)
    integer i
    real*8 x(*),y(*),z(*)
    CALL ALIGNX(16, x(1))
    CALL ALIGNX(16, y(1))
    CALL ALIGNX(16, z(1))
    do i=1,1000
      z(i)=sqrt(x(i)*x(i)+y(i)
*y(i))
    enddo
    return
end
```

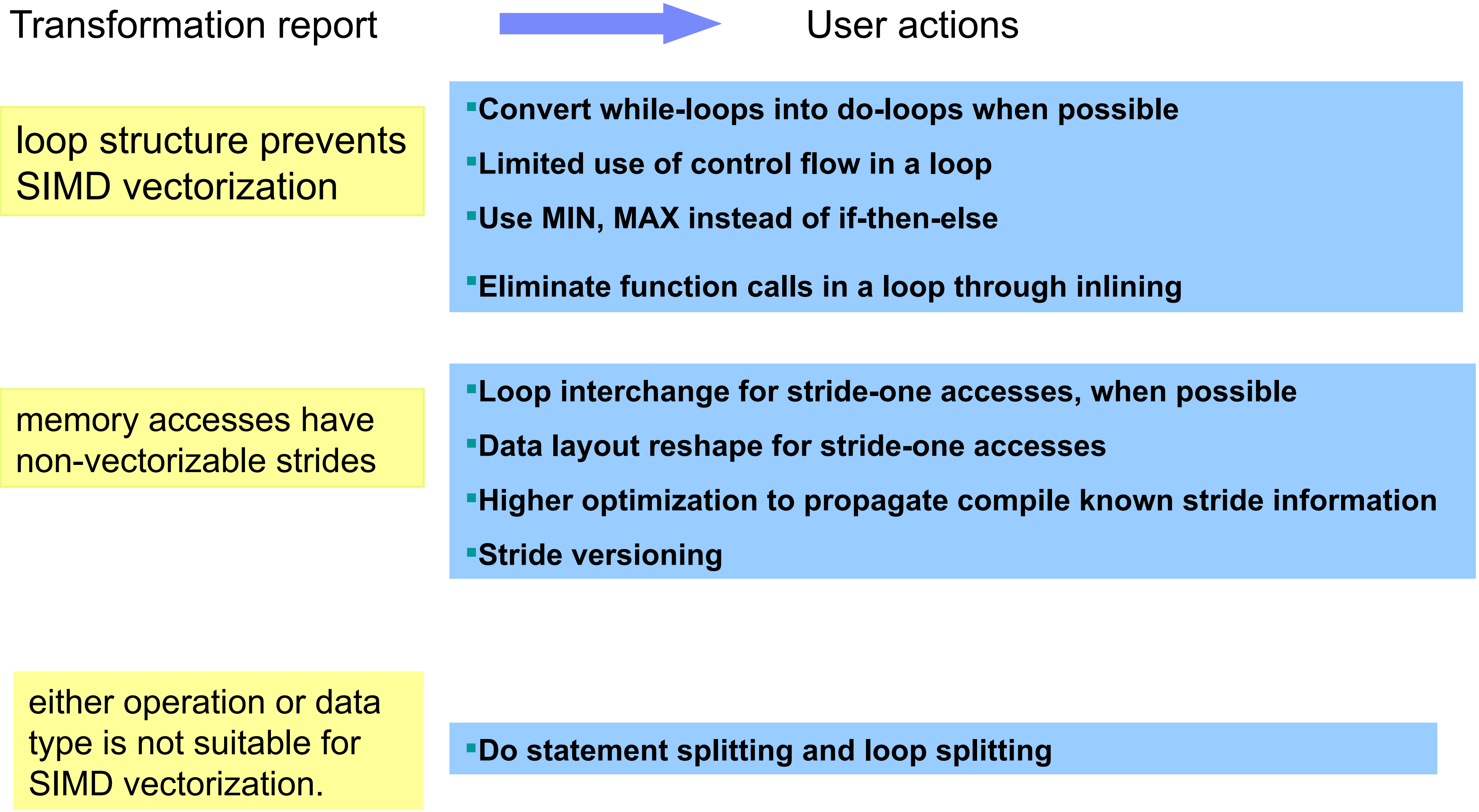
Compiler Listing:

```
 0|  CL.115:
 9|      VLQD        vs4=@V.y[].rns2.0(gr11,gr12,0)
 9|      VLQD        vs5=@V.y[].rns2.0(gr11,gr31,0)
 9|      VLQD        vs9=@V.x[].rns3.1(gr8,gr0,0)
 9|      VLQD        vs10=@V.x[].rns3.1(gr8,gr23,0)
 9|      VDFM        vs4=vs4,vs4,fcr
 9|      VDFM        vs5=vs5,vs5,fcr
 9|      LRVS        vs8=vs2
 9|      VDFMA       vs8=vs8,vs9,vs9,fcr
 9|      LRVS        vs9=vs3
 9|      VDFMA       vs9=vs9,vs10,vs10,fcr
 9|      VDFSQRT     vs10=vs0,fcr
 9|      VDFSQRT     vs11=vs1,fcr
 9|      VSTQD       @V.z[].rns1.2(gr9,gr26,0)=vs7
 9|      VSTQD       @V.z[].rns1.2(gr9,gr25,0)=vs6
 0|      AI          gr9=gr9,64
   ...
 0|      BCT         ctr=CL.115,,100,0
```

Tips for SIMDization Tuning

Transformation report	User actions
Loop was SIMD vectorized	
It is not profitable to vectorize	▪Use #pragma simd_level(10) to force the compiler to do SIMDization
data dependence prevents SIMD vectorization	▪Use fewer pointers when possible ▪Use #pragma independent if it has no loop carried dependency ▪Use #pragma disjoint (*a, *b) if a and b are disjoint ▪Use restrict keyword or compiler option –qrestrict
memory accesses have non-vectorizable alignment.	▪Use __attribute__((aligned(n)) to set data alignment ▪Use __alignx(16, a) to indicate the data alignment to the compiler ▪Use -qassert=refalign if all references are naturally aligned ▪Use array references instead of pointers where possible

Tips for SIMDization Tuning



Transformation report → User actions

loop structure prevents SIMD vectorization

- Convert while-loops into do-loops when possible
- Limited use of control flow in a loop
- Use MIN, MAX instead of if-then-else
- Eliminate function calls in a loop through inlining

memory accesses have non-vectorizable strides

- Loop interchange for stride-one accesses, when possible
- Data layout reshape for stride-one accesses
- Higher optimization to propagate compile known stride information
- Stride versioning

either operation or data type is not suitable for SIMD vectorization.

- Do statement splitting and loop splitting

Tips for Compiler Friendly Programming

- Obey all language aliasing rules (avoid –qalias=noansi in C/C++)
- Avoid unnecessary use of globals and pointers; use restrict keyword or compiler directives/pragmas to help the compiler do dependence and alias analysis
- Use “const” for globals, parameters and functions whenever possible
- Group frequently used functions into the same file (compilation unit) to expose compiler optimization opportunity (e.g., intra compilation unit inlining, instruction cache utilization)
- Excessive hand-optimization such as unrolling and inlining may impede the compiler
- Keep array index expressions as simple as possible for easy dependency analysis
- Consider using the highly tuned MASS and ESSL libraries rather than custom implementations or generic libraries

Tips for POWER7 Optimizations

- POWER7 exploitation
  - POWER7 specific ISA exploitation under –qarch=pwr7
    - Extended FP register file
    - 64-bit population count, bit permutation, fixed point pipelined multiply, fix point select, divide check for software divide assistance
    - VMS/VSX
    - Stride-N stream prefetch, partial cache line touch
  - Scheduling and instruction selection under –qtune=pwr7
- Automatic SIMDization
  - Use simd_level(0..10) pragma to exploit aggressive SIMDization
  - Use align attribute to force the compiler to align static data by 16-byte; use MALLOCALIGN=16 to force OS to align malloced data by 16-byte; use alignx directive to tell the compile the alignment.
  - Limited use of control flow
  - Limited use of pointers. Use independent_loop directive to tell the compiler a loop has no loop carried dependency; use either restrict keyword or disjoint pragma to tell the compiler the references do not share the same physical storage whenever possible
  - Limited use of stride accesses. Expose stride-one accesses whenever possible

Tips for POWER7 Optimizations

- POWER7 aware loop transformations
  - Loop distribution, unroll-and-jam, stream unrolling controlled by 12 streams on each core, shared by SMTs
  - Loop blocking controlled by L2 cache size
  - Selection of SIMDization and vectorization controlled by the threshold; use loop iteration directives to guide the compiler
- Memory hierarchy optimization
  - Data prefetch
    - Automatic data prefetch at O3 –qhot or above.
    - -qprefetch=aggressive to enable aggressive data prefetch;
    - DSCR setting for the default data prefetching;
    - Enable DCBZ insertion on POWER7 IH
    - Partial cache line touch
  - Data reorganization enabled at O5

XL Compiler Documentation

- An information centre containing the documentation for the XL Fortran V14.1 and XL C/C++ V12.1 versions is available at:
  - AIX Compilers:http://pic.dhe.ibm.com/infocenter/comphelp/v121v141/index.jsp
  - Linux Compilers: http://pic.dhe.ibm.com/infocenter/lnxpcomp/v121v141/index.jsp
    - Installation Guide
    - Getting Started with XL C/C++
    - Compiler Reference
    - Language Reference
- Whitepaper “Code optimization with the IBM XL Compilers”
  - http://www-01.ibm.com/support/docview.wss?uid=swg27005174
- Whitepaper “Overview of the IBM XL C/C++ and XL Fortran Compiler Family” available at:
  - http://www.ibm.com/support/docview.wss?uid=swg27005175
- Please send any comments or suggestions on this information center or about the existing C, C++ or Fortran documentation shipped with the products to compinfo@ca.ibm.com.

Go to IBM

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.