Compiler Usage Guidelines for AMD64 Platforms

Application Note

Publication # **32035** Revision: **3.22**
Issue Date: **November 2007**

Advanced Micro Devices 

Contents

Tables

Revision History

Date	Rev.	Description
November 2007	3.22	Made minor corrections. Seventh public release.
September 2007	3.21	Sixth public release.
August 2006	3.19	Fifth public release.
June 2005	3.18	Fourth public release. Updated generic performance switches for Sun Solaris in Section 3.8, Section 3.16, and Section 4.16.
June 2005	3.16	Third public release.
February 2005	3.09	Second public release.
October 2004	3.00	Initial public release.

Chapter 1 Introduction

Independent software vendors (ISVs) and end-users of platforms for the AMD Athlon™ 64, AMD Opteron™, and AMD Family 10h processors have a significant interest in porting and tuning their applications for the AMD64 architecture. Because several compilers are available for AMD64 architecture, evaluating them to choose the best-suited compiler for an application is a non-trivial task. This document provides a quick reference for optimization and portability switches for some commonly used compilers. The intent is to provide starting guidelines for porting and performance tuning applications and for increased performance of compiled code. The user should refer to the user's guides for specific compilers for further tuning help or for troubleshooting problems that are beyond the simple diagnostic steps listed here.

New compilers of interest are always on the horizon. This document may be updated when new compilers arrive or when the current compiler switches change significantly in their newer versions.

1.1 Audience

Theoretically, benchmarks should provide clear, unequivocal information that guides end-users in making choices about software and hardware. Reality is somewhat less than ideal; therefore, benchmarks can be quite subjective and prone to interpretation. Benchmarks are guidelines, not absolute answers and benchmarking can be a tricky business, especially when it comes to compilers. Developers can gain insight about the relative performance of different tools, by comparing results in a controlled environment. To be valid, benchmark source code must be available, and the testing conditions clearly stated. It is not methodologically sound to use a limited data set generated by a circumscribed suite of benchmarks demonstrating specific aspects of code generation to predict general compiler performance.

This document is intended for ISVs, SIs and end-users of the AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processor-based platforms who wish to port and tune their applications for the AMD64 architecture.

1.2 Intent of Document

This document provides a quick reference for optimization and portability switches for some commonly used compilers for AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processor-based platforms.

Performance models of applications enable high-performance computing (HPC) system designers and IT to gain insight into the optimal hardware for end-user applications, giving valuable information into the components of hardware, improving applications performance, and inform machine procurement and design. Real-world applications are currently the preferred method for measuring performance, whereas benchmarks are required for the discovery of interest or "door-

openers". Standard Performance Evaluation Corporation (SPEC) designed CPU2006 to provide a comparative measure of computation-intense performance across the widest range of hardware using workloads developed from real user applications. SPECcpu2006 is CPU-intensive—stressing a system's processor, memory subsystem and compiler.

This document provides a quick reference for optimization and portability switches commonly used when invoking compilers for AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processor-based platforms. SPECcpu2006 provides some insight into which command line options to utilize for certain applications. Chapter 5, Peak Options for SPEC®-CPU Benchmark Programs, on page 55 documents the compiler switches utilized on the compact application components representing the SPECcpu2006 benchmark suite.

1.3 Definitions, Abbreviations, and Notation

Switches and invocation commands are highlighted in **bold** text.

1.4 Additional Documents

Other resources for developers working with 64-bit operating systems include the following.

- *Software Optimization Guide for AMD Athlon™ 64 and AMD Opteron™ Processors,* order# 25112
- *System V Application Binary Interface (AMD64 Architecture Processor Supplement)* http://www.amd64.org/documentation
- PGI Compiler User's Guides: *http://www.pgroup.com/resources/docs.htm*
- Intel Compiler Manuals:
  http://www.intel.com/software/products/compilers/clin/docs/manuals.htm
  http://www.intel.com/software/products/compilers/flin/docs/manuals.htm
  http://www.intel.com/software/products/compilers/cwin/docs/manuals.htm
  http://www.intel.com/software/products/compilers/fwin/docs/manuals.htm
- Microsoft® Windows® AMD64 Application Binary Interface
- MSDN: *http://msdn.microsoft.com/*
- GNU Compiler Collection: *http://gcc.gnu.org*
- GCC Online Documentation: http://gcc.gnu.org/onlinedocs/
- Sun Studio Documentation: *http://developers.sun.com/prodtech/cc/reference/docs/index.html*

Chapter 2 List of Compiler Vendors for AMD Processors

The compiler vendors listed in this chapter are discussed in detail in subsequent chapters of this application note. This is not a comprehensive list of all compiler vendors for AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processors.

Table 1. Summary of Compilers1 lists the compiler vendors discussed in this document and shows whether a vendor provides 64-bit compilers, 32-bit compilers, or both for the Linux®, Microsoft® Windows®, or Sun Solaris platforms.

Table 1. Summary of Compilers

Compiler Vendor	Compiler Platform		
	Linux®	Microsoft® Windows®	Sun Solaris
PGI	64-bit and 32-bit	64-bit and 32-bit	–
Sun	64-bit and 32-bit	–	64-bit and 32-bit
GCC	64-bit and 32-bit	–	64-bit and 32-bit
Intel	64-bit and 32-bit	64-bit and 32-bit	–
PathScale	64-bit and 32-bit	–	–
Microsoft®	–	64-bit and 32-bit	–

2.1 Compilers (64-Bit) for Linux®

The following companies provide 64-bit compilers for Linux.

2.1.1 GCC

GCC provides C, C++, and Fortran compilers for AMD64 architecture-based systems running the Linux or the Sun Solaris operating systems. This application note, however, does not discuss GCC compilers for Sun Solaris; this discussion is limited to the discussion of GCC compilers for Linux. Different Linux distributions offer different versions of the GCC compilers. This application note focuses on the recommended compilers for the following major Linux distributions:

- SuSE Linux Enterprise Server 8
- SuSE Linux Enterprise Server 9
- SuSE Linux Enterprise Server 10
- SuSE Linux 10.1
- SuSE Linux 10.2

- Red Hat Enterprise Linux 3
- Red Hat Enterprise Linux 4

This application note also briefly discusses the GCC 4.2 compiler, which is the current GCC compiler from the Free Software Foundation (FSF).

2.1.2 Intel

Intel provides C, C++, and Fortran compilers for EM64T and compatible architecture-based systems running the Linux operating systems. The current version (as of August 2007) is 10.0.

2.1.3 PathScale

PathScale provides C, C++, and Fortran compilers for AMD64 architecture-based systems running the Linux operating system. The current version (as of August 2007) is 3.0.

2.1.4 PGI

The Portland Group (PGI) Toolkits are composed of high performance C, C++, and/or Fortran Compiler(s), a debugger, and a performance profiler for 32-bit and 64-bit AMD64 and EM64T processor-based Linux. The latest PGI Edition 7 provides leading-edge application performance on AMD64 next-generation systems and supports features like auto-parallelization, OS-native multithreading, OpenMP multithreading models, and MPI programming for AMD64 architecture-based multicore shared-memory and distributed-memory cluster-based systems. The current version (as of Sept 2007) is PGI Release 7.1.

2.2 Compilers (64-Bit) for Microsoft® Windows®

The following companies provide 64-bit compilers for Microsoft Windows.

2.2.1 Intel

Intel provides C/C++ and Fortran compilers for EM64T and compatible systems running the Microsoft Windows operating system. The current version (as of August 2007) is 10.0.

2.2.2 Microsoft®

Microsoft provides C/C++ compilers for AMD64 architecture-based systems running the Microsoft Windows operating system. The current version is Visual Studio 2008.

2.2.3 PGI

The Portland Group (PGI) Toolkits are composed of high performance C, C++ and/or Fortran Compiler(s), a debugger and a performance profiler for 32-bit and 64-bit AMD64 and EM64T processor-based Windows platforms. The latest PGI Edition 7 provides leading-edge application

performance on AMD64 next-generation systems and supports features like auto-parallelization, OS native multithreading, OpenMP multithreading models, and MPI programming for AMD64 architecture-based multicore shared-memory and distributed-memory cluster-based systems. The current version (as of Sept 2007) is PGI Release 7.1.

2.3 Compilers (64-bit) for Solaris

The following companies provide 64-bit compilers for x86 Solaris.

2.3.1 Sun

Sun provides C, C++, and Fortran compilers for the AMD64 architecture-based systems running the Sun Solaris operating system. The current version (as of August 2007) is 5.9 and comes in the Sun Studio 12 developer tool suite.

2.4 Compilers (32-Bit) for Linux®

The following companies provide 32-bit compilers for x86 Linux. These compilers also run on 64-bit Linux Operating systems, running on AMD Athlon™ 64 processor-based platforms, AMD Opteron™ processor-based platforms, or AMD Family 10h processor-based platforms.

2.4.1 GCC

The GNU Compiler Collection (GCC) provides C, C++, and Fortran compilers for x86 Linux and Sun Solaris. This application note, however, does not discuss the GCC compilers for Sun Solaris; it discusses only GCC compilers for Linux. Different Linux distributions offer different versions of the GCC compiler. This application note focuses on the recommended compilers for the following major Linux distributions for workstations and servers—SuSE Linux Enterprise Server 8, SuSE Linux Enterprise Server 9, SuSE Linux Enterprise Server 10, SuSE Linux 10.1, SuSE Linux 10.2, Red Hat Enterprise Linux 3 and Red Hat Enterprise Linux 4. This application note also briefly discusses the GCC 4.2 compiler, which is the current GCC version from the Free Software Foundation (FSF).

2.4.2 Intel

Intel provides C, C++, and Fortran compilers for x86 Linux. The current version (as of August 2007) is 10.0. This document also talks about two previous versions of the compiler, 9.1 and 8.1, because they are comparable in performance to the current version (when run on AMD platforms) and are still in use.

2.4.3 PathScale

PathScale provides C, C++, and Fortran compilers for x86 Linux. The current version (as of August 2007) is 3.0.

2.4.4 PGI

The Portland Group (PGI) Toolkits are composed of high performance C, C++, and/or Fortran Compiler(s), a debugger, and a performance profiler for 32-bit and 64-bit AMD64 and EM64T processor-based Linux. The latest PGI Edition 7 provides leading-edge application performance on AMD64 next-generation systems and supports features like auto-parallelization, OS native multithreading, OpenMP multithreading models, and MPI programming for AMD64 architecture-based multicore shared-memory and distributed-memory cluster-based systems. The current version (as of September 2007) is PGI Release 7.1.

2.5 Compilers (32-Bit) for Microsoft® Windows®

The following companies provide 32-bit compilers for Microsoft Windows.

2.5.1 Intel

Intel provides C, C++ and Fortran compilers for x86 Microsoft Windows. The current version (as of August 2007) is 10.0 This document also talks about two previous versions of the compiler, 9.1 and 8.1, because they are comparable in performance to the current version and are still in use.

2.5.2 Microsoft®

Microsoft provides C/C++ compilers for x86 Microsoft Windows. The current version is Microsoft Visual Studio 2008.

2.5.3 PGI

The Portland Group (PGI) Toolkits are composed of high performance C, C++, and/or Fortran Compiler(s), a debugger, and a performance profiler for 32-bit and 64-bit AMD64 and EM64T processor-based Windows platforms. The latest PGI Edition 7 provides leading-edge application performance on AMD64 next-generation systems and supports features like auto-parallelization, OS native multithreading, OpenMP multithreading models, and MPI programming for AMD64 architecture-based multicore shared-memory and distributed-memory cluster-based systems. The current version (as of Sept 2007) is PGI Release 7.1.

2.6 Compilers (32-bit) for Sun Solaris

The following companies provide 32-bit compilers for Sun Solaris.

2.6.1 Sun

Sun provides C, C++, and Fortran compilers for x86 Solaris operating system. The current version (as of August, 2007) is 5.9 and comes in the Sun Studio 12 developer tool suite.

Chapter 3 Performance-Centric Compiler Switches

This chapter describes the various switches that can be useful for individual compilers. For each compiler, a list of generally recommended performance switches is provided. This list is further augmented by other switches that could prove beneficial for certain code bases.

3.1 PGI Compilers (32- and 64-Bit) for Linux® and Microsoft® Windows®

The Portland Group (PGI) high performance C, C++, and Fortran compilers (PGCC, PGC++, PGHPF, PGF95, PGF77) and program development tools (PGDBG debugger and PGPROF profiler) optimize code for 32-bit and 64-bit AMD64 and EM64T processor-based Linux® and Microsoft® Windows® platforms. PGI Edition 7 provides local and global optimizations, loop optimization (unrolling, vectorization, and parallelization), inter-procedural analysis and optimization, and function inlining on AMD64 single-, dual- and quad-core systems. PGI Tools support parallel programming features like auto-parallelization, OS native multithreading, OpenMP multithreading models, and MPI programming for AMD64 architecture-based multicore shared-memory and distributed-memory cluster-based systems. The current version (as of September 2007) is PGI Release 7.1. All the options described in this section apply to PGI Release 7.1.

3.1.1 Invocation Commands

The following commands invoke specific compilers and tools:

- **pgcc** invokes the PGI C compiler.
- **pgcpp** (**pgCC**) invokes the PGI C++ compiler.
- **pgf77** invokes the PGI Fortran 77 compiler.
- **pgf95** invokes the PGI Fortran 90/95 compiler.
- **Pghpf** invokes the PGI High-performance Fortran Compiler
- **pgdbg** invokes the PGDBG source code debugger
- **pgfrof** invokes the PGPROF performance profiler

Note: *Invoking PGI compilers within BASH on Windows platforms is case insensitive, therefore using pgCC will invoke the PGI C compiler (i.e. pgCC is equivalent to pgcc).*

3.1.2 General Performance Switches

To get a program running, start by compiling and linking without optimization. Use the optimization level **-O0** or select **-g** to perform minimal optimization. At this level, you can debug a program easily and isolate any coding errors exposed during porting to x86 or AMD64 platforms. Use option **-tp** (i.e. target processor) to specify the target architecture. Options **-tp k8-64** and **-tp k8-64e** result in the generation of code supported on and optimized for AMD64 processors. Edition 7 supports AMD Opteron quad-core processor with options **-tp barcelona-64** to generate 64-bit code and **-tp barcelona** to generate 32-bit code.

Note: *The 64-bit PGI compiler can generate 32-bit binaries.*

To get started quickly with optimization, with any PGI compiler use options **-fast** and **-Mipa=fast**. For C++ programs, add **-Minline=levels:10 --no_exceptions** (C++ program compiled with **--no_exceptions** will fail if the program uses exception handling). Beginning in Edition 7 the **-fast** option became synonymous with the **-fastsse** option, and the optimizations performed by **-fast** in previous releases were placed under the **-nfast** option.

Note: *The **-fastsse** option is still necessary to compile 32 bit code.*

Generally, further significant performance gains can be realized. However, individual optimizations can sometimes cause slowdowns depending on coding style. Optimization flags most likely to further improve performance are**-O3**, **-Mpfi/-Mpfo**, **-Minline**, and on targets with multiple processors **-Mconcur**,

The --zc_eh option allows zero-cost exception handling for C++.

For C++ BASE optimization, use --zc_eh with **-Mipa=fast,inline** and **-Msmartalloc=huge**. The **huge** flag enables the use of huge pages if the OS is configured to provide them.

3.1.3 Optimization Switches

In addition to the **-tp** (i.e., target processor) switch, the following list of switches may improve the performance of the program. It is worth experimenting with these switches, but care must be used to ensure performance improvements.

Local and Global Optimization using -O. Specify any of the following optimization level (**-O***level*) options.

-O0—(level-0) specifies no optimization. This optimization level generates a basic block for each language statement. This is useful for debugging since there is a direct correlation between the program text and the code generated.

-O1 (level-1) specifies local optimization. This optimization level performs scheduling of basic blocks and allocates registers.

-O2 (level-2) specifies global optimization. This optimization level performs all level-one local optimization as well as level-two global optimization.

-O3 (level-3) specifies aggressive global optimization. This optimization level performs all level-one and level-two optimizations and enables more aggressive hoisting and scalar replacement optimizations that may or may not be profitable.

-O4 (level-4) performs all level-1, level-2, and level-3 optimizations and enables hoisting of guarded invariant floating point expressions.

Loop Optimization using -Munroll, -Mvect, and -Mconcur. Loop performance may be improved through vectorization or unrolling options, and, on systems with multiple processors, by using parallelization options.

-Munroll unrolls loops. Executing multiple instances during each loop iteration reduces branch overhead, improving execution speed by creating better opportunities for instruction scheduling. Using **-Munroll** sub-options **c:number** and **n:number**, or using **-Mnounroll** can control whether and how loops are unrolled.

-Mvect option triggers the vectorizer to scan code searching for loops that are candidates for high-level transformations such as loop distribution, loop exchange, cache tiling, and idiom recognition (replacement of a recognizable code sequence, such as a reduction loop, with optimized code sequences or function calls). The vectorizer transformation can be controlled by arguments to the **-Mvect** option. By default, **-Mvect** without sub-options is equivalent to **-Mvect=assoc, cachesize:262144**. Vectorization sub-options are **assoc**, **cachesize:number**, **sse**, and **prefetch**.

-Mconcur option instructs the compiler to scan code searching for loops that are candidates for auto-parallelization. **-Mconcur** must be used at compile-time and link-time. The parallelizer performs various operations that are controlled by arguments to the **-Mconcur** option. By default, **-Mconcur** without sub-options is equivalent to **-Mconcur=dist:block**. Auto-Parallelization sub-options are **altcode:number**, **dist:block, dist:cycle**, **cncall**, **noassoc,** and **innermost.**

Interprocedural Analysis and Optimization using -Mipa. Interprocedural analysis (IPA) can improve performance for many programs. To compile programs with IPA use an aggregated suboption such as **-Mipa=fast**. Refer to the *PGI Compiler User's Guide* for available sub-options.

Function Inlining using -Minline. Inlining allows a call to a function or subroutine to be replaced by a copy of the body of that function or subroutine. Several **-Minline** sub-options determine the selection criteria for functions to be inlined. Available sub-options are **except:func**, **name:func**, **size:number**, **levels:number**, and **lib:filename.ext**. Note that in C++ releases prior to 6.2, function inlining does not occur unless the **-Minline** switch is used. Beginning with release 6.2 inlining will occur automatically for C++ functions specified by means of the `inline` keyword or methods defined in the body of the class. Also, if C++ exceptions are not used, the **--no_exceptions** flag improves performance.

3.1.4 Linking with ACML

Due to the strategic importance of the AMD multi-core processor architecture, libraries are in place to assist developers in porting software to AMD processors. AMD Core Math Library (ACML) is designed to "squeeze" the greatest possible performance from AMD multi-core platforms and is integrated in all PGI Toolkits. As the number of cores increases over time, future processor

innovations are automatically incorporated into applications through the use of ACML. The AMD Core Math Library (ACML) revision 4.0, built with PGI Edition 7, includes BLAS, LAPACK, FFT and RNG routines that are optimized for AMD Athlon™ 64 and AMD Opteron™ processors. If the program uses these routines, using ACML in place of generic C/Fortran implementation may greatly improve the performance. For additional details on how to install this library and use it, please refer to the ACML User Guide available at *http://developer.amd.com/assets/acml_userguide.pdf*.

3.2 GCC Compilers (64-Bit) for Linux®

The 64-bit GCC compilers can be installed and run on 64-bit Linux®, AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processors. GCC compilers vary slightly, depending on the Linux distribution. This section discusses the following GCC compilers.

- gcc 4.2.0 from Free Software Foundation (FSF)
- gcc 4.2.0 from SuSE Linux Enterprise Server 10
- gcc 4.2.0 supplied with Red Hat Enterprise Linux 4

3.2.1 Recommended Compiler Versions

The Linux distributions from SuSE and Red Hat include a default 64-bit GCC compiler and optional GCC compilers. From a performance standpoint, the optional compilers are recommended. Table 2, below, shows the recommended (optional) compiler versions for the current SuSE and Red Hat distributions. These optional compilers are included on product CDs and DVDs.

Table 2. GCC Versions Included with Linux® Distributions

Linux® Distribution	Default GCC Compiler Version	Recommended (Optional) Compiler Version
Red Hat Enterprise Linux 4	4.1.0	gcc-ssa
SuSE Linux Enterprise Server 10	4.1.0	4.2.0
Red Hat Enterprise Linux 4	3.4.1	No optional compiler available with the distribution. The default compiler is the recommended compiler.
SuSE Linux 10.1	4.1.0	4.2.0
SuSE Linux Enterprise Server 10	3.3.3	4.2.0

Table 2, "GCC Versions Included with Linux® Distributions," identifies the recommended optional compilers by their package names. The Red Hat distribution media include the Red Hat Package Managers. The gcc-ssa package is installed in /usr/bin by default, while gcc-33 is installed in /opt/gcc33/.

In addition to the supplied compilers, the user can also experiment with the latest GCC compilers (version 3.4, 4.0, and 4.2.0) from the Free Software Foundation (FSF). Users probably cannot expect, however, the same level of support for FSF GCC compilers as they can expect for supplied compilers.

3.2.2 Invocation Commands

The following commands invoke specific compilers:

- **gcc** invokes the C compilers for gcc 4.1, 3.4.1, 3.4, 3.3.4 and gcc 3.3.3.
- **gcc-ssa** invokes the gcc-ssa C compiler.
- **gfortran** invokes the Fortran 90/95 compiler for gcc 4.1.
- **g++** invokes the C++ compilers for gcc 4.1, 3.4.1, 3.4, 3.3.4 and gcc 3.3.3.
- **g++-ssa** invokes the gcc-ssa C++ compiler.
- **g77** invokes the Fortran 77 compiler for gcc 3.4.1, 3.4, 3.3.4 and gcc 3.3.3.
- **g77-ssa** invokes the gcc-ssa Fortran 77 compiler.

The user may have to specify the full path of the invocation command for using the optional GCC compilers. For example, the optional SLES8 GCC compiler will be invoked by **/opt/gcc33/bin/gcc**.

3.2.3 Generic Performance Switches

Different optimization switches are recommended for 64-bit SuSE GCC 3.3.3, Red Hat gcc-ssa, and the FSF gcc 4.1 compilers. Table 3 shows the recommended switches for these compilers.

Table 3. Recommended Option Switches for 64-Bit GCC Compilers for Linux®

Compiler Version	Recommended Optimization Switches
SuSE GCC 3.3.3 (for C/C++ and Fortran) and Red Hat gcc-ssa (C/C++ and Fortran)	-O3 -ffast-math -funroll-all-loops
FSF GCC 3.4 (for C/C++ and Fortran) and Red Hat GCC 3.4.1	-O3 -ffast-math -funroll-all-loops -fpeel-loops -ftracer -funswitch-loops -funit-at-a-time
SuSE GCC 4.2	-O3 -ffast-math -funroll-all-loops -fpeel-loops
FSF GCC 4.2 (for C/C++ and Fortran)	-O3 -ffast-math -funroll-all-loops -ftree-vectorize

The **-O3** switch turns on several general optimizations.

Using the **-ffast-math** switch allows the compiler to use a fast floating point model.

The **-funroll-all-loops** causes all loops to be unrolled and makes code larger and could bring improvement in speed.

Some of the options implied by **-O3** in SuSE GCC 3.3.3 and the gcc-ssa compilers are not implied by the GCC 3.4 compiler and should be added for additional performance improvement. These are **-funit-at-a-time**, **-fpeel-loops**, **-ftracer**, and **-funswitch-loops**.

The GCC 4.0 and later version compilers can perform loop vectorization by using the **-ftree-vectorize** flag.

3.2.4 Other Switches

In addition to the switches mentioned in Table 3 on page 23, the following list of switches may also improve the program performance. It is worth experimenting with these switches.

-march=k8. For the FSF GCC 4.2.0 SuSE 4.2.0and Red Hat 4.2.0 compilers, using this switch may give you a performance advantage in some cases.

-march=amdfam10. For applications to be executed on AMD Family 10h processor-based platforms, this switch results in better performance.

*Note: The **amdfam10** option is not available on all GCC compiler releases. See your compiler documentation for further information.*

Profile Guided Optimization. The 64-bit GCC compiler also allows profile guided optimization. Table 4 shows the profile guided optimization switches for the different GCC compilers.

Table 4. Profile Guided Optimization for 64-Bit GCC Compilers for Linux®

Compiler Version	Optimization Switches
SuSE GCC 4.2.0 and Red Hat gcc-ssa (for C/C++ and Fortran)	Step 1.Compile the program with **-fprofile-arcs**. Step 2.Run the executable produced in Step 1. Running the executable generates several files with profile information (*.da). Step 3.Recompile the program with **-fbranch-probabilities**.
FSF GCC 4.2.0 and Red Hat GCC 4.2.0	Step 1.Compile the program with **-fprofile-generate**. Step 2.Run the executable produced in Step 1. Running the executable generates several files with profile information (*.da). Step 3.Recompile the program with **-fprofile-use**.

-Bsymbolic. Sarting from GCC 4.1, gcc compiler no longer requires the **-Bsymbolic** switch. GCC 4.1 and later versions offer **-combine -fwhole -program**, which should be used together, but require that makefiles be changed to use a single command to compile and link all files of an application, slowing down builds. So it should only be used for non-debug builds. Unfortunately, these options may fail compiling some files.

-minline-all-stringops. When using the GCC 3.4 compiler on Red Hat Enterprise Linux 4, experiment with the switch **-minline-all-stringops**. This switch is *not* recommended for GCC 3.4 on SuSE Linux Enterprise Server.

Linking with ACML. The AMD Core Math Library (ACML) includes BLAS, LAPACK and FFT routines that are optimized for AMD Athlon™ 64, AMD Opteron™, and AMD Family 10h processors. If the program uses these routines, using ACML in place of generic C/Fortran

implementation may greatly improve the performance. For additional details on how to install this library and use it, see *http://developer.amd.com/assets/acml_userguide.pdf*.

-fno-rtti. This switch disables generation of information about every class, with virtual functions, for use by the C++ runtime type identification features (**dynamic_cast** and **typeid**). If the user does not use those parts of the language, some space can be conserved by using this switch.

Generate 32-Bit Binaries with -m32. The user can also use 64-bit GCC compilers to generate 32-bit binaries by using the **-m32** switch. This can improve performance if the program has lots of variables of the type long and/or pointers. As these data-types are 32-bit in x86, this switch will reduce the memory footprint of the program. Also, the user should use the recommended switches for the 32-bit GCC compiler (section 3.8.4 on page 32) when **-m32** is used.

Users can obtain more details on these switches by trying **info gcc** on their Linux systems.

3.3 Intel Compilers (64-Bit) for Linux®

Intel provides 64-bit compilers for Linux that can be used for AMD64 systems. The current version (as of Augus 2007) is 10.0. All options described in this section apply to this version.

3.3.1 Invocation Commands

The following commands invoke specific compilers:

- **icpc** invokes the C++ compiler.
- **icc** invokes the C compiler.
- **ifort** invokes the Fortran compiler.

3.3.2 Generic Performance Switches

The switches **-xW -ipo -O3 -static** are generally recommended.

3.3.3 Other Switches

In addition to the generic performance switches, it is worth experimenting with the following switches.

Profile Guided Optimization. Intel compilers allow profile guided optimization. Use the following steps for profile guided optimization with Intel compilers.

1. Compile the program with the **-prof_gen** switch. The **-ipo** or **-ip** switch is ignored by the compiler if used with **-prof_gen**.
2. Run the executable produced in Step 1. Running this executable generates several files with profile information (*.dyn and *.dpi).
3. Recompile the program with the **-prof_use** switch. It is recommended to also use the **-ipo** switch in this stage.

-fno-rtti. Using this switch instructs the C++ compiler to discard C++ run-time type information (RTTI). This may improve performance. However, C++ features requiring RTTI (exceptions, dynamic cast, etc.) will not be supported.

-ansi-alias. Try this switch if the program strictly conforms to the ISO C99 standard. If the program adheres to the standard, this switch allows the compiler to perform aggressive optimizations.

3.4 PathScale Compilers (64-Bit) for Linux®

PathScale provides C, C++, and Fortran compilers for AMD64 architecture-based systems running the Linux operating system. The current version (as of August 2007) is 3.0 All options described in this section apply to this version.

3.4.1 Invocation Commands

The following commands invoke specific compilers:

- **pathcc** invokes the QLogic PathScale C compiler.
- **pathCC** invokes the QLogic PathScale C++ compiler.
- **pathf95** invokes the QLogic PathScale Fortran compiler.

3.4.2 Generic Performance Switches

The **-O3** and **-OPT:Ofast** switches are recommended as the first step of optimization. For further tuning, experiment with the switches in the next section.

3.4.3 Other Switches

In addition to the **-O3** and **-OPT:Ofast** switches, the following list of switches may improve the performance of the program. It is worth experimenting with these switches.

Profile Guided Optimization. The 64-bit QLogic PathScale compiler allows profile guided optimization. Use the following steps for profile guided optimization with 64-bit PathScale compilers for Linux.

1. Compile the program with the **-fb_create fbdata** switch.
2. Run the executable produced in Step 1. It will generate several files with profile information.
3. Recompile the program with the **-fb_opt fbdata** switch.

Inter-Procedure Optimization. Use the switch **-ipa** to enable inter-procedure optimization.

-Ofast. For aggressive optimization, use the **-Ofast** switch. This is the shorthand for the switches **-O3**, **-OPT:Ofast**, **-ipa -ffast-math**, and **-fno-math-errno** .

Linking with ACML. The AMD Core Math Library (ACML) includes BLAS, LAPACK and FFT routines that are optimized for AMD Athlon™ 64 and AMD Opteron™ processors. If the program

uses these routines, using ACML in place generic C/Fortran implementation may greatly improve the performance. Use the GNU64 libraries of ACML for the 64-bit PathScale compiler. For additional details on how to install this library and use it, see *http://developer.amd.com/assets/acml_userguide.pdf*.

Refer to the *PathScale EKOPath Compiler Suite User Guide*, Version 2.1, for more options and suggestions for tuning your application performance.

3.5 Intel Compilers (64-Bit) for Microsoft® Windows®

Intel provides C, C++ and Fortran compilers for EM64T and compatible architecture-based systems running 64-bit Microsoft® Windows® operating systems. The current version (as of August 2007) is 10.0. All options described here apply to this version.

3.5.1 Invocation Commands

The following commands invoke specific compilers:

- **icl** invokes the Intel C and C++ compiler.
- **ifort** invokes the Intel Fortran compiler.

3.5.2 Generic Performance Switches

The switches **-QxW -Qipo -O3** are generally recommended.

3.5.3 Other Switches

In addition to the generic performance switches above, it is worth experimenting with the following switches.

Profile Guided Optimization. Intel compilers allow profile guided optimization. Use the following steps for profile guided optimization with Intel compilers.

1. Compile the program with the **-Qprof_gen** switch. The **-Qipo** or **-Qip** switch is ignored by the compiler if used with **-Qprof_gen**.
2. Run the executable produced in Step 1. Running this executable generates several files with profile information (*.dyn and *.dpi).
3. Recompile the program with the **-Qprof_use** switch. It is recommended to also use **-Qipo** in this stage.

-Qansi-alias. Try this switch if the program strictly conforms to the ISO C99 standard. If the program adheres to the standard, this switch allows the compiler to perform more aggressive optimizations.

3.6 Microsoft® Compilers (64-Bit) for Microsoft® Windows®

Microsoft provides C/C++ compilers for AMD64 architecture-based systems running the Microsoft Windows operating system. The current version is Visual Studio 2008. This document contains the latest C/C++ compiler recommendations for Visual Studio 2008. All the options described below apply to this version of the compiler.

3.6.1 Invocation Commands

The **cl** command invokes the Microsoft C/C++ compiler.

3.6.2 Generic Performance Switches

The **/O2**, **/GL** and **/fp:fast** switches almost always result in improved performance. The **/O2** switch turns on several general optimizations. The **/GL** enables interprocedural optimizations. Using **/fp:fast** allows the compiler to use a fast floating-point model. However, for applications that requires high precision this switch should be avoided. For code that may be sensitive to cache size, consider using the **/O1** compiler switch. **/O1** will generate smaller code at the possible expense of instruction execution speed. However, the potential performance improvement due to smaller code footprint may be of more benefit than any loss due to slower instructions.

Profile-Guided Optimization. The 64-bit Microsoft compiler allows profile-guided optimization. Use the following steps for profile-guided optimization with 64-bit Microsoft compilers for Microsoft Windows.

1. Compile the program with the **/GL** switch and link with the **/LTCG:PGI** switch.
2. Run the executable produced in Step 1. Running the executable generates several files with profile information.
3. Relink the program with the **/LTCG:PGO** switch.

/D_SECURE_SCL=0. To turn off linking with secure C++ libraries, use the **/D_SECURE_SCL=0** switch. This switch can improve the performance of iterator-heavy C++ code , but can sacrifice security as buffer-overrun checks are disabled.

/OPT:ref,icf. This linker option removes redundant symbols and unused functions, resulting in a smaller binary.

3.6.3 /favor Performance Switch

When targeting AMD Family 10h processors, use the **/favor:blend** switch for best performance. If no favor flag is specified, **/favor:blend** is the default. When targeting AMD processors prior to AMD family 10h, use the **/favor:AMD64** switch. It will typically result in improved performance on those platforms.

3.7 Sun Compilers (64-bit) for Solaris

Sun provides C, C++, and Fortran compilers for AMD64 architecture-based systems running the Solaris operating system. The current version (as of August, 2007) is version 5.9 available in the Sun Studio 12 developer tools suite. All the options described below apply to this version of the compiler.

3.7.1 Invocation Commands

The following commands invoke specific compilers:

- **cc** invokes the Sun Studio C compiler.
- **CC** invokes the Sun Studio C++ compiler.
- **f77** invokes the Sun Studio Fortran 77 compiler.
- **f90** invokes the Sun Studio Fortran 90 compiler.

3.7.2 Generic Performance Switches

Use the following switches to enable generation of 64-bit binaries**, -xarch=sse3a -m64**, which includes prefetch to help tune better for the AMD instruction set architecture; **-m64**, which is the same as
-xarch=generic64 (which we otherwise recommend because it helps for SPARC as well as Xeon and Opteron processors) produces binaries meant to run on both ISA, Xeons and AMD processors.

Different optimization switches are recommended for different platforms. The **-fast** switch enables a number of optimizations that optimize the execution time on the compilation platform. If the program is run on a different machine **-fast** can be combined with **-xtarget** to optimize for a different platform. If performance on a wide variety of systems is desired, combine
-xtarget=generic with **-fast**. If a switch implied by **-fast** (e.g., **-xarch=isa**) is overridden, that switch must follow **-fast** on the command line, or it will be ignored. For AMD Family 10h, we recommend using **-xtarget=barcelona** to take better advantage of AMD Family 10h ISA. Note that patch01 for Sun Studio 12 must be installed before one can use this switch.

3.7.3 Other Switches

In addition to the generic switches, the following switches may improve the performance of the program. It is worth experimenting with these switches.

Use the **-xO[1|2|3|4|5]** switch to enable various levels of general optimization algorithms. Usually using a higher number results in faster execution, but in some cases **-xO2** or **-xO3** is faster than **-xO4** or **-xO5**.

Note: *The **-fast** switch implies the **-xO5** switch.*

The **-xprofile=collect:[name]** and **-xprofile=use:[name]** flags enable profile-guided optimization. The flags must be specified both when compiling and linking. After compiling with the

-xprofile=collect:[name] flag, run the program on a typical dataset. Then compile with **-xprofile=use:[name]** to utilize the resulting profile data to tune the program.

The **-xcrossfile** flag enables optimization across all source files. This flag must be combined with **-xO4** or **-xO5** to be effective.

The **-xipo=2** flag enable interprocedural optimization (this option is preferred over **-xcrossfile**, which was pre-ipo).

The **-xprefetch** and **-xprefetch_level=1,2,3**. causes prefetch with various metric triggers **-xalias_level**, which communicates that a given program is known to adhere to certain aliasing restrictions **-xvector=simd,lib** which causes generation of SIMD instructions for chips that support SIMD(all, for Opteron).

Note: *The **-fast** switch implies the **-xO5** switch.*

Additional performance improvements can be gained in floating point programs using the **-fsimple[=n]** switch. The **-fsimple=2** flag enables aggressive floating point optimizations, but sacrifices numeric accuracy. This flag is implied by **-fast**.

3.8 GCC Compilers (32-Bit) for Linux®

The 32-bit GNU Compiler Collection (GCC) compilers can be installed and run on 32-bit Linux and 64-bit Linux on AMD Athlon™ 64 and AMD Opteron™ processors. The GCC compilers come in a number of different varieties. This section discusses the following different GCC compilers:

- gcc 4.2.0 from Free Software Foundation (FSF)
- gcc 4.2.0 compiler from SuSE Linux Enterprise Server 10
- gcc 4.2.0 compiler from SuSE Linux 10.1

3.8.1 Recommended Compiler Versions

The Linux distributions from SuSE and Red Hat include a default 32-bit GCC compiler and optional compilers. From a performance standpoint, the optional compilers are recommended. Table 5 shows the recommended (optional) compiler versions for the current SuSE and Red Hat distributions.

Table 5. GCC Versions Included with Linux® Distributions

Linux® Distribution	Default GCC Compiler Version	Recommended (Optional) Compiler Version
Red Hat Enterprise Linux® 3	3.2	gcc-ssa
SuSE Linux Enterprise Server 10	4.1	gcc 4.2.0

Table 5. GCC Versions Included with Linux® Distributions

Red Hat Enterprise Linux 4	3.4.1	No optional compiler available with the distribution. The default compiler is the recommended compiler.
SuSE Linux 10.1	4.1.1	4.2.0
SuSE Linux Enterprise Server 10	4.1.0	4.2.0

Table 4, "Profile Guided Optimization for 64-Bit GCC Compilers for Linux®," on page 24 identifies the recommended optional compilers by their package names. The distribution media include the RPMs. The package gcc-ssa is installed in /usr/bin by default, while gcc-33 is installed in /opt/gcc33/.

In addition to the supplied compilers, the user can also experiment with the latest GCC compilers (versions 4.2.0) from the Free Software Foundation (FSF). Users probably cannot expect, however, the same level of support for FSF GCC compilers as they can expect for supplied compilers.

3.8.2 Invocation Commands

The following commands invoke specific compilers:

- **gcc** invokes the C compilers for 4.0, 3.4.1, 3.4, 3.3.4 and gcc 3.3.3.
- **gcc-ssa** invokes the gcc-ssa C compiler.
- **gfortran** invokes the Fortran 95 compiler for 4.0.
- **g++** invokes the C++ compilers for 4.0, 3.4.1, 3.4, 3.3.4 and gcc 3.3.3.
- **g++-ssa** invokes the gcc-ssa C++ compiler.
- **g77** invokes the Fortran 77 compiler for 3.4.1, 3.4, 3.3.4 and gcc 3.3.3.
- **g77-ssa** invokes the gcc-ssa Fortran 77 compiler.

The user may have to specify the full path of the invocation command for using the optional GCC compilers. For example, /opt/gcc33/bin/gcc invokes the optional SLES8 GCC compiler.

3.8.3 Generic Performance Switches

Different optimization switches are recommended for 32-bit SuSE GCC 3.3.3, Red Hat gcc-ssa and 3.4 compiler versions. Table 6 shows the recommended optimization switches for the listed compilers.

Table 6. Recommended Option Switches for 32-Bit GCC Compilers for Linux®

Compiler Version	Recommended Optimization Switches

Table 6. Recommended Option Switches for 32-Bit GCC Compilers for Linux®

SuSE GCC 4.2.0 (for C/C++ and Fortran) and Red Hat gcc-ssa (for C/C++ and Fortran)	**-O3 -march=k8 -ffast-math -fomit-frame-pointer -malign-double -mfpmath=sse**
FSF GCC 4.2.0-Red Hat GCC 3.4.1	**-O3 -march=k8 -ffast-math -fomit-frame-pointer -malign-double -mfpmath=sse -fpeel-loops -ftracer -funswitch-loops -funit-at-a-time**
SuSE GCC 4.2.0	**-O3 -march=k8 -ffast-math -fomit-frame-pointer -malign-double -mfpmath=sse -fpeel-loops**
FSF GCC 4.2.0 (for C/C++ and Fortran)	**-O3 -march=k8 -ffast-math -fomit-frame-pointer -malign-double -mfpmath=sse -fpeel-loops -ftracer -funswitch-loops -ftree-vectorize**

The **-O3** switch turns on several general optimizations.

Using the **-ffast-math** switch allows the compiler to use a significantly fast floating point model.

The **-fomit-frame-pointer** causes the frame pointer to be omitted resulting in a performance improvement. The user should not use this switch if they need to rewind the stack using the frame pointer.

Using **-malign-double** will result in better alignment and hence faster code on the AMD Athlon™ 64 and AMD Opteron™ processors.

Using **-mfpmath=sse** causes the compiler to generate SSE/SSE2 instructions in favor of the default x87 instructions.

Since the default for the 32-bit gcc compiler is **-march=i386**, using **-march=k8** causes it to generate high-performance code for the AMD Athlon™ 64 and AMD Opteron™ processors, while using **-march=amdfam10** causes it to generate high-performance code for AMD Family 10h processors.

The GCC 4.2.0 compiler can perform loop vectorization by using the **-ftree-vectorize** flag.

3.8.4 Other Switches

In addition to the switches mentioned in Table 6, "Recommended Option Switches for 32-Bit GCC Compilers for Linux®," on page 31 the following list of switches may also improve the performance of the program. It is worth experimenting with these switches.

Profile Guided Optimization. The 32-bit GCC compiler allows profile guided optimization. Table 7 shows the profile guided optimization switches for the three GCC compilers.

Table 7. Profile Guided Optimization for 32-Bit GCC Compilers for Linux®

Compiler Version	Optimization Switches

Table 7. Profile Guided Optimization for 32-Bit GCC Compilers for Linux®

SuSE GCC 4.2.0 (for C/C++ and Fortran) and Red Hat gcc-ssa (for C/C++ and Fortran) and SuSE GCC 4.2.0	Step 1.Compile the program with **-fprofile-arcs**. Step 2.Run the executable produced in Step 1. Running this executable generates several files with profile information (*.da). Step 3.Recompile the program with **-fbranch-probabilities**.
FSF GCC 4.2.0 (for C/C++ and Fortran) and Red Hat GCC 4.2.0	Step 1.Compile the program with **-fprofile-generate**. Step 2.Run the executable produced in Step 1. Running this executable generates several files with profile information (*.da). Step 3.Recompile the program with **-fprofile-use**.

-funroll-loops. This switch causes loops, the iterations of which can be determined at compile time or entry into the loop to be unrolled. Some loops are therefore unrolled.

This switch can be used with all three versions of the 32-bit GCC Compilers for Linux.

-Bsymbolic. GCC 4.2 no longer uses the **-Bsymbolic** compiler switch. Instead, GCC 4.2 now offers the **-combine -fwhole-program** switch combination, which should be used together. This switch combination requires that makefiles be changed to use a single command to compile and link all files of an application, slowing down builds. So the **-combine -fwhole-program** switch combination, should only be used for non-debug builds. Unfortunately, these options may fail when compiling some files**.**

minline-all-stringops. When using the GCC 4.2.0 compiler on Red Hat Enterprise Linux 4, experiment with the switch **-minline-all-stringops**. This switch is *not* recommended for GCC 4.2.0 on SuSE Linux Enterprise Server.

This switch can be used with all three versions of the 32-bit GCC Compilers for Linux.

Linking with ACML. The AMD Core Math Library (ACML) includes BLAS, LAPACK and FFT routines that are optimized for AMD Athlon™ 64 and AMD Opteron™ processors. If the program uses these routines, using ACML in place of generic C/Fortran implementation may greatly improve the performance. For additional details on how to install this library and use it, see *http://developer.amd.com/assets/acml_userguide.pdf*.

ACML can be used with all 3 versions of 32-bit GCC Compilers for Linux discussed in this application note.

Generate 32-bit binaries with -m32. The 32-bit GCC compilers generate 32-bit binaries by default. The user can also use 64-bit GCC compilers to generate 32-bit binaries by using the **-m32** switch. Use the switches recommended in this section along with the **-m32** switch.

This switch can be used with all three versions of the GCC Compilers for Linux talked about here.

-fno-rtti. This switch disables generation of information about every class with virtual functions for use by the C++ runtime type identification features (**dynamic_cast** and **typeid**). If the user does not use those parts of the language, some space can be conserved by using this switch.

Users can obtain more details on these switches by trying **info gcc** on their Linux systems.

3.9 Intel Compilers (32-Bit) for Linux®

The 32-bit Intel compilers can be installed and run on 32-bit and 64-bit Linux on AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processors. On 64-bit Linux, the 32-bit binaries will run in compatibility mode. To be able to do this, one has to tell the system linker on 64-bit Linux to link with 32-bit libraries, and to generate 32-bit executable. This can be done with the following command.

```
PROMPT$ icc -Wl,-m,elf_i386 <other compiler switches> <source files>
```

3.9.1 Invocation Commands

The following commands invoke specific compilers:

- **icpc** invokes the Intel C++ compiler for version 10.0.
- **icc** invokes the Intel C/C++ compilers for version 10.0.
- **ifort** invokes the Intel Fortran versions 10.0 compiler.

3.9.2 Generic Performance Switches

These flags are recommended for Intel 10.0 compiler: -xW -ipo -O3 -static.

The **-xW** switch instructs the compiler to optimize for a Pentium® 4 processor (including SSE2 instructions).

The **-ipo** switch enables inter-procedural (across source files) analysis.

The **-O3** switch optimizes for speed, including several aggressive optimizations.

3.9.3 Other Switches

In addition to the switches mentioned in the following list of switches may also improve the performance of the program. It is worth experimenting with these switches.

Profile Guided Optimization. Intel compilers allow profile guided optimization. Use the following steps for profile guided optimization with Intel compilers.

1. Compile the program with the **-prof_gen** switch. The **-ipo** or **-ip** switch is ignored by the compiler if used with **-prof_gen**.
2. Run the executable produced in Step 1. Running this executable generates several files with profile information (*.dyn and *.dpi).

3. Recompile the program with the **-prof_use** switch. It is recommended to also use the **-ipo** switch in this stage.

-nolib_inline. For programs with many calls to memory-related library routines (such as, **memmove** and **memcopy**), using the **-nolib_inline** switch may improve performance for Intel compiler versions 7.1 and 8.0. This switch is *not* recommended for version 9.1.

-unroll[n]. This switch sets the maximum number of times to unroll a loop. Experiment with values 1–4. For scientific programs, a particular value may slightly improve performance.

-fno-rtti. Using this switch will instruct the C++ compiler not to keep C++ run-time type information (RTTI). This may improve performance. However, C++ features requiring RTTI (exceptions, dynamic cast, etc.) will not be supported.

-ansi-alias. Try this switch if the program strictly conforms to the ISO C99 standard. If the program adheres to the standard, this switch allows the compiler to perform aggressive optimizations.

3.10 PathScale Compilers (32-Bit) for Linux®

PathScale provides C, C++, and Fortran compilers for x86 Linux. The current version (as of August 2007) is 3.0. All the options described in this section apply to this release. To generate 32-bit binaries, the -m32 switch must be used with the PathScale compiler.

3.10.1 Invocation Commands

The following commands invoke specific compilers:

- **pathcc** invokes the PathScale C compiler.
- **pathCC** invokes the PathScale C++ compiler.
- **pathf90** invokes the PathScale Fortran compiler.

3.10.2 Generic Performance Switches

Use the **-O3** and **-OPT:Ofast** switches as the first step of optimization. For further tuning, experiment with the switches in Section 3.10.3.

3.10.3 Other Switches

In addition to the **-O3** and **-OPT:Ofast** switches, the following list of switches may improve the performance of the program. It is worth experimenting with these switches.

Profile Guided Optimization. The 32-bit PathScale compiler also allows profile guided optimization. Use the following steps for profile guided optimization with PathScale compilers.

1. Compile the program with the **-fb_create fbdata** switch.
2. Run the executable produced in Step 1. Running this executable generates several files with profile information.

3. Recompile the program with the **-fb_opt fbdata** switch.

Inter-Procedure Optimization. Use the **-ipa** switch to enable inter-procedure optimization.

-Ofast. For aggressive optimization, use the **-Ofast** switch. This is the shorthand for the switches **-O3**, **-OPT:Ofast**, **-ipa**, and **-fno-math-errno**.

Linking with ACML.

The AMD Core Math Library (ACML) includes BLAS, LAPACK and FFT routines that are optimized for AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processors. If the program uses these routines, using ACML in place of generic C/Fortran implementation may greatly improve the performance. For additional details on how to install this library and use it, see *http://developer.amd.com/assets/acml_userguide.pdf*.

Refer to the *PathScale EKOPath Compiler Suite User Guide*, Version 2.1, for more options and suggestions for tuning your application performance.

3.11 Intel Compilers (32-Bit) for Microsoft® Windows®

The 32-bit Intel compilers can be installed and run on 32-bit Microsoft Windows on AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processors.

3.11.1 Invocation Commands

The following commands invoke specific compilers:

- **icl** invokes the 32-bit Intel C/C++ compilers.
- ifort invokes the 32-bit Intel Fortran versions 9.1 and 10.0compilers.

3.11.2 Generic Performance Switches

Use of the **-QxW -Qipo -O3** switches are recommended for Intel compiler version 10.0.

The **-QxW** switch instructs the compiler to optimize for Pentium 4 processor (including SSE2 instructions).

The **-Qipo** switch enables interprocedural (across multiple source files) analysis.

The **-O3** optimizes for speed and includes several aggressive optimizations.

3.11.3 Other Switches

In addition to the switches mentioned in the program. It is worth experimenting with these switches.

Profile Guided Optimization. Intel compilers allow profile guided optimization. Use the following steps for profile guided optimization with Intel compilers.

1. Compile the program with the **-Qprof_gen** switch. The **-Qipo** or **-Qip** switch is ignored by the compiler if used with **-Qprof_gen**.

2. Run the executable produced in Step 1. Running the executable generates several files with profile information (*.dyn and *.dpi).
3. Recompile the program with the **-Qprof_use** switch. It is recommended to also use the **-Qipo** switch in this stage.

-Oi-. For programs with many calls to memory-related library routines (such as, **memset** and **memcpy**), using the **-Oi-** switch may improve performance for Intel compiler versions 7.1 and 8.0. This switch is *not* recommended for version 9.1.

-Qunroll[n]. This switch sets the maximum number of times to unroll a loop. Experiment with values 1–4. For scientific programs, a particular value may slightly improve performance.

-Qansi-alias. Try this switch if the program strictly conforms to the ISO C99 standard. If the program adheres to the standard, this switch allows the compiler to perform aggressive optimizations.

3.12 Microsoft® Compilers (32-Bit) for Microsoft® Windows®

The 32-bit Microsoft compilers can be installed and run on 32-bit Microsoft Windows and 64-bit Microsoft Windows on AMD Athlon™ 64, AMD Opteron™, and AMD Family 10h processors. The current version is Visual Studio 2008. All the options below apply to this version.

3.12.1 Invocation Command

The **cl** command invokes the Microsoft C/C++ compiler.

3.12.2 Generic Performance Switches

The **/O2**, **/GL**, **/Oy**, and **/fp:fast** switches almost always result in improved performance. The **/O2** switch turns on several general optimizations. The **/GL** switch enables whole-program IPA and **/Oy** allows the compiler to use frame pointer register as a general register which usually result in better performance. Using **/fp:fast** allows the compiler to use fast math library routines with extensive error checking turned off. Using **/fp:fast** also allows the compiler to adhere to a fast but less predictable floating point model in general. However, applications that require high precision should avoid using this switch. For code that may be sensitive to cache size, consider using the **/O1** compiler switch. **/O1** will generate smaller code at the possible expense of instruction execution speed. However, the potential performance improvement due to smaller code footprint may be of more benefit than any loss due to slower instructions.

3.12.3 Other Switches

In addition to the **/O2**, **/GL**, **/Oy**, and **/fp:fast** switches, the following list of switches may improve the performance of the program. It is worth experimenting with these switches.

Profile Guided Optimization. The 32-bit Microsoft compiler allows profile guided optimization. Use the following steps for profile guided optimization with 32-bit Microsoft compilers for Microsoft Windows.

1. Compile the program with the **/GL** switch and link with the **/LTCG:PGI** switch.
2. Run the executable produced in Step 1. Running this executable generates several files with profile information.
3. Relink the program with the **/LTCG:PGO** switch.

The **-arch:SSE2** switch allows the compiler to use the SSE2 instructions, when it determines that it is faster than x87 for scalar, floating-point computations and will interleave the two as appropriate. As a result, the code uses a mixture of both x87 and SSE2. Using this switch almost always results in increased speed.

The compiler emits code that is thread-safe by default. Turning off this default by using **/D_ST_MODEL** can result in an additional performance improvement.

/OPT:ref,icf. This linker option removes redundant symbols and unused functions, resulting in a smaller binary.

3.13 Sun Studio Compilers (32-bit) for Solaris

Sun Microsystems provides C, C++, and Fortran compilers for the x86 Solaris operating system. The current version of each compiler (as of August 2007) is 5.9, and is available in the Sun Studio 12 developer tools suite. All options below apply to this version of the compilers.

3.13.1 Invocation Commands

The following commands invoke specific compilers:

- **cc** invokes the Sun Studio C compiler.
- **CC** invokes the Sun Studio C++ compiler.
- **f77** invokes the Sun Studio Fortran 77 compiler.
- **f90** invokes the Sun Studio Fortran 90 compiler.

3.13.2 Generic Performance Switches

Different optimization switches are recommended for different platforms. The **-fast** switch enables a number of optimizations that optimize the execution time on the compilation platform. If the program will be run on a different machine, **-fast** can be combined with **-xtarget** to optimize for a different platform. If performance on a wide variety of systems is desired, combine **-xtarget=generic** with **-fast**. If a switch implied by **-fast** (e.g., **-xarch=*isa***) is overridden, that switch must follow **-fast** on the command line, or it will be ignored.

3.13.3 Other Switches

In addition to the generic switches, the following switches may improve the performance of the program. It is worth experimenting with these switches.

Use the **-xO[1|2|3|4|5]** switch to enable various levels of general optimization algorithms. Usually using a higher number results in faster execution, but in some cases **-xO2** or **-xO3** may be faster than **-xO4** or **-xO5**.

Note: *The **-fast** switch implies the **-O5** switch.*

The **-xprofile=collect:[name]** and **-xprofile=use:[name]** flags enable profile guided optimization. The flags must be specified both when compiling and linking. After compiling with the **-xprofile=collect:[name]** flag, run the program on a typical dataset. Then compile with **-xprofile=use:[name]** to utilize the resulting profile data to tune the program.

The **-xcrossfile** flag enables optimization across all source files. This flag must be combined with **-xO4** or **-xO5** to be effective.

Note: *The **-fast** switch implies the **-xO5** switch.*

Additional performance improvements can be gained in floating point programs using the **-fsimple[=n]** switch. The **-fsimple=2** switch enables aggressive floating point optimizations, but sacrifices numeric accuracy. This flag is implied by **-fast**.

Chapter 4 Troubleshooting and Portability Issues

Tuning code for optimal performance presents a wide variety of challenges from compilation errors to unexpected results. This chapter presents the developer with a series of diagnostic steps for a given compiler to troubleshoot errors encountered when compiling or running code.

Troubleshooting issues fall into the following broad categories:

- Compilation errors
- Interoperability between languages
- Link-time errors
- Run-time errors
- Compiled and linked code generates unexpected results
- Other issues

4.1 PGI Compilers for Linux® and Microsoft® Windows®

This section addresses errors and unexpected results that may be encountered when using 32-bit and/or 64-bit PGI compilers for Linux and Microsoft® Windows®.

4.1.1 Interoperability Between Languages

Is your program composed of both C/C++ and Fortran modules?

This section discusses several issues that can arise when linking together Fortran and C/C++ modules.

Definition of `main()` in a C/C++ Module

When linking together C and Fortran object files using the **pgf90** invocation command, if the `main()` function is included in one of the C objects, use the **-Mnomain** switch. Using the **-Mnomain** switch instructs the PGI compiler not to include the Fortran `main` program module during linking.

Ensuring Cases and Underscores Match

By default Linux and Microsoft Windows convert all Fortran symbol names to lower-case. C and C++ are case sensitive, so upper-case function names stay upper-case. When using inter-language calling, either name the C/C++ functions with lower-case names, or invoke the Fortran compiler command

with the **-Mupcase** switch. This switch prevents the compiler from converting symbol names to lower-case.

To match the underscore appended by the compiler to global symbol names in Fortran, use the following function naming convention.

1. When calling a C/C++ function from Fortran, rename the C/C++ function by appending an underscore.
2. When calling a Fortran function from C/C++, append an underscore to the Fortran function name in the calling program.

Functions or Subroutines?

Fortran, C, and C++ define functions and subroutines differently. For a Fortran program calling a C or C++ function, observe the following return value convention.

1. When the C or C++ function returns a value, call the value from Fortran as a function; when the C or C++ function returns something other than a value, call the value as a subroutine.
2. When calling a Fortran function from C/C++, the call should return a similar type. For a list of compatible types between the C/C+ and Fortran modules, refer to the *PGI Compiler User's Guide*.

The32-bit PGI compiler for Windows supports several different calling conventions. The nature of the issues regarding interoperability of languages depends on the calling convention used. For additional details please refer to the *PGI Compiler User's Guide*.

Passing by Reference vs. Passing by Value

Fortran passes arguments by reference (i.e., the address of the argument is passed, rather than the argument itself). C/C++ passes arguments by value, except for strings and arrays, which are passed by reference. C/C++ provides the flexibility to work around these differences. Solving the parameter passing differences generally involves intelligent use of the & and * operators in argument passing when C/C++ calls Fortran and in argument declarations when Fortran calls C/C++.

For strings declared in Fortran as type CHARACTER, Fortran passes an argument representing the length of the string to a calling function. On Linux systems, the compiler places the length argument(s) at the end of the parameter list, following the other formal arguments and passes the length argument by value, not by reference.

Passing Arrays

C/C++ arrays and Fortran arrays use different default initial array index values. By default, C/C++ arrays start at 0 and Fortran arrays start at 1. Adjust your array comparisons so that the second element in a Fortran array is compared to the first element in a C/C++ array. Make similar adjustments for other elements. If adjusting initial array index values is not satisfactory, declare your Fortran arrays to start at zero.

Fortran and C/C++ arrays also use different storage methods. Fortran uses column-major order, and C/C++ uses row-major order. This poses no problems for one-dimensional arrays. For two-dimensional arrays, where there are an equal number of rows and columns, simply reverse the row and column indices. For arrays other than single dimensional arrays, and square two-dimensional arrays, inter-language function mixing is not recommended.

Linking Fortran Modules with C/C++ Main Programs

You must explicitly link in the PGI Fortran runtime support libraries when linking **pgf90**-compiled program units into C or C++ main programs (C/C++ calling Fortan) using the switches **-lpgf90**, **-lpgf90_rpm1**, **-lpgf902**, **-lpgf90rtl**, and **-lpgftnrtl**. When linking **pgf77**-compiled program units into C or C++ main programs, you need to use only the **-lpgftnrtl** switch.

4.1.2 Run-Time Errors

Does your program expect 64-bit integers?

By default, the Fortran INTEGER data-type is a 32-bit entity in AMD64. If a program expects INTEGER to be a 64-bit entity (e.g., programs ported from some 64-bit architecture, such as Alpha), use the **-i8** switch. The **-i8** switch makes all integers 64-bit entities. This switch is only available for the PGI Fortran compiler (**pgf90**).

Are you receiving a run-time error?

Check for array overruns. Run-time errors can be caused by accessing arrays out-of-bounds. Use the switch **-Mbounds** to generate code for checking array bounds.

4.1.3 Compiled and Linked Code Generates Unexpected Results

Are you generating vectorized code?

For some loops, vectorization can cause a slight difference in results due to the reordering of floating-point operations. Using the switch combination **-tp=k8-64** and **-fastsse** may cause vectorization. Try using the non-vectorizing switch combination **-tp=k8-64**, **-Mscalarsse**, and **-fast** as a diagnostic step instead. As an alternative to the vectorizing switches, use the non-vectorizing switches if their use causes your code to give the correct, expected behavior.

Does your program require floating-point divisions conforming to the IEEE 754 standard?

Use the **-Kieee=strict** switch to generate floating-point divisions that are strictly compliant with the IEEE 754 standard.

Does your program rely on x87 features?

The **-fastsse** switch instructs the compiler to use SSE2 registers and instructions. If the results of a program do not match your expectations when using SSE2 registers and instructions, the program may rely on some x87 features.

As a diagnostic step, try building the program using x87 operations for floating-point computations and see if the results are as expected. Use the **-tp=k8-32** and **-fast** switches instead of the switches recommended in the general performance guidelines.

Because not using those switches recommended in the general performance guidelines could lower performance, the user should investigate the precision requirements of the program. If the user has access to the source code, it may be possible to adapt the algorithm to SSE2.

4.1.4 Program Gives Unexpected Results or Terminates Unexpectedly

Are your binary input data files big-endian?

If your Fortran program is performing unformatted I/O, and the data files are big-endian, use the **-Mbyteswapio** switch for swapping endian formats.

4.2 GCC Compilers (64-Bit) for Linux®

This section addresses errors and unexpected results that may be encountered when using 64-bit GCC compilers for Linux.

4.2.1 Compilation Errors

Do you need ANSI-compliant code?

If a developer requires ANSI-compliant code in a program, GCC provides the **-ansi** switch to test the ANSI-compliance of the code in a program. To see gratuitous errors and warnings for the non-ANSI parts of the program, the user should use the **-pedantic** switch. The user can then modify the program to be ANSI-compliant. The user can also use the **-std** switch to specify the required version of ISO C.

*Does your code suffer from 64-bit portability issues, such as type casting pointers to **int**?*

GCC provides the **-Wall** switch to show all warnings. This switch enables the user to detect 64-bit portability issues, such as type-casting pointers to **int**.

On 64-bit Linux, **int** is 32 bits, and pointers and **long** are 64 bits (LP64). Do not use **int** for type-casting pointers. Use ISO C99 portable, scalable data-types such as **intptr_t** for this purpose. Additional information on this can be obtained in the ISO C99 Standard document.

Users should note that **-Wall** is not sufficient to get all warnings from gcc. The following switches are avalaible that turn GCC into an effective 'lint': **-Werror**, **-Wall**, **-W**, **-Wstrict-prototypes**, **-Wmissing-prototypes**, **-Wpointer-arith**, **-Wreturn-type**, **-Wcast-qual**, **-Wwrite-strings**, **-Wswitch**, **-Wshadow**, **-Wcast-align**, **-Wuninitialized**, **-Wbad-function-cast**, **-Wchar-subscripts**, **-Winline**, **-Wnested-externs**, **-Wredundant-decl**, **-ansi**, **-pedantic**. For further detail on these switches refer to the gcc manual.

4.2.2 Link-Time Errors

Are you trying to link C and Fortran code?

Turn on the **-fno-f2c** switch for compiling Fortran 77 modules with **g77**. Turning on the **-fno-f2c** switch prevents **g77** from generating code designed to be compatible with code generated by **f2c** and uses the GNU calling conventions instead.

4.2.3 Run-Time Errors

Is your code causing buffer overruns?

Turn on the **-fbounds-check** switch. When the **-fbounds-check** switch is turned on, the GCC compiler generates additional code to check whether the indices used to access arrays are or are not within the declared range. The **-fbounds-check** switch is currently supported only by the Fortran 77 front-end, in which this option defaults to false.

Are you building a shared library?

Turn on the **-fPIC** switch if you need position-independent code suitable for use in a shared library.

4.2.4 Compiled and Linked Code Generates Unexpected Results

Does your program depend on precise floating point behavior?

Do not use the **-ffast-math** switch. When the **-ffast-math** is used, the compiler relaxes the rules when optimizing floating-point operations. This mode allows the compiler to further optimize floating-point code for speed, sometimes at the expense of floating-point accuracy. Do not use the **-ffast-math** switch if precise floating-point behavior is required.

Does your program rely on x87 features?

The 64-bit GCC compiler emits SSE/SSE2 code with **-mfpmath=sse**, which can yield better performance. SSE2 offers 64-bit precision, which is sufficient for almost all programs. If the results do not match your expectations when using SSE2, the program may rely on some x87 features.

As a diagnostic step, try building the program using x87 operations for floating-point computations and see if the results are as expected. Do this by omitting the **-mfpmath=sse** switch recommended in the general performance guidelines. By default the compiler uses **-mfpmath=387**.

Because omitting the **-mfpmath=sse** switch could lower performance, the user should investigate the precision requirements of the program. If the user has access to the source code, it may be possible to adapt the algorithm to SSE2.

4.2.5 Program Gives Unexpected Results or Exception Behavior

Does your code depend on exact implementation of IEEE rules or specifications for floating-point behavior?

GCC provides switches, such as the **-mieee-fp** switch, to control whether or not the compiler uses IEEE floating-point comparisons.

The user should not use the **-ffast-math** optimization recommended in the general optimization guidelines in this case. Using the **-ffast-math** switch results in a fast but less predictable floating-point model. The user should also be careful to not use a switch that implies **-ffast-math**.

Does your code need C++ exception handling?

GCC generates the extra code needed to propagate exceptions with the **-fexceptions** switch. For some targets, propagating exceptions implies that GCC generates frame unwind information for all functions. Generating frame unwind information for all functions can produce significant data-size overhead, although it does not affect the execution of a program.

By default, GCC enables the **- fexceptions** option for languages like C++ that normally require exception handling. GCC disables the **-fexceptions** option for languages like C that do not normally require it. You may need, however, to enable this option when compiling C code that must interoperate properly with exception handlers written in C++. You may also wish to disable this option if you are compiling older C++ programs that do not use exception handling.

Do you need to unwind the stack using the frame pointer?

The frame pointer is omitted by default on 64-bit GCC compilers to improve performance. This default omission can be reversed by using **-fno-omit-frame-pointer**.

4.3 Intel Compilers (64-Bit) for Linux®

See section 4.9, “Intel Compilers (32-Bit) for Linux®”, on page 50 for the portability and troubleshooting issues with this compiler.

4.4 PathScale Compilers (64-Bit) for Linux®

For information on diagnosing problems with the PathScale compiler, refer to the tuning document distributed with the PathScale compiler suite.

4.5 Intel Compilers (64-Bit) for Microsoft® Windows®

See section 4.11, “Intel Compilers (32-Bit) for Microsoft® Windows®”, on page 51 for troubleshooting errors with this compiler.

4.6 Microsoft® Compilers for (64-Bit) Microsoft® Windows®

This section addresses errors and unexpected results that may be encountered when using 64-bit Microsoft® compilers for Microsoft Windows®.

4.6.1 Compilation Errors

*Does your code suffer from 64-bit portability issues such as type-casting pointers to **int** or **long**?*

Use the **/Wp64** switch to detect 64-bit porting problems. This switch can be used with both 32-bit and 64-bit Microsoft compilers. (This switch is on by default for the 64-bit compiler.)

On AMD64 architecture-based systems running the Microsoft Windows operating system, both **int** and **long** are 32 bits (P64), and pointers are 64 bits. Do not use **int** or **long** for type-casting pointers. Use portable, scalable data types like INT_PTR, UINT_PTR, LONG_PTR, and ULONG_PTR for type-casting pointers.

Note: *Data types INT_PTR, UINT_PTR, LONG_PTR, and ULONG_PTR are Microsoft specific data types.*

Issues such as these can be detected by using the **/Wp64** switch.

4.6.2 Run-Time Errors

Is your code causing buffer overruns and thus violating security?

Turn on the **/GS** switch. Turning on the **/GS** switch causes the Microsoft compiler to generate additional security code, such as bounds checking.

4.6.3 Compiled and Linked Code Generates Unexpected Results

Does your program depend on precise floating-point behavior?

Do not use the **/fp:fast** switch recommended in the general performance guidelines. When the **fp:fast** mode is enabled, the compiler relaxes the rules that **fp:precise** uses when optimizing floating-point operations. This mode allows the compiler to further optimize floating-point code for speed at the expense of floating-point accuracy.

4.6.4 Program Gives Unexpected Results or Exception Behavior

Does your code depend on the exact implementation of IEEE or ISO rules or specifications for floating-point behavior?

Do not use the **/fp:fast** switch recommended in the general performance guidelines. The compiler uses **/fp:precise** by default if no **/fp** switch is specified.

Does your code need C++ exception handling?

Enable exception handling with the appropriate /EH switch.

4.7 Sun Compilers (64-bit) for Solaris

See section 4.13, “Sun Compilers (32-bit) for Solaris”, on page 54 for the portability and troubleshooting issues with this compiler.

4.8 GCC Compilers (32-Bit) for Linux®

This section addresses errors and unexpected results that may be encountered when using 32-bit GNU Compiler Collection (GCC) compilers for Linux®.

4.8.1 Compilation Errors

Do you need ANSI-compliant code?

If a developer requires ANSI-compliant code in a program, the GCC compiler provides the **-ansi** switch to test the ANSI-compliance of the code in a program. To see gratuitous errors and warnings for the non-ANSI parts of the program, the user should use the **-pedantic** switch. The user can then modify the program to be ANSI-compliant. The user can also use the **-std** switch to specify the required version of ISO C.

GCC also provides the **-Wall** switch to show almost all warnings. This switch enables all the warnings about constructions that some users consider questionable.

Users should note that **-Wall** is not sufficient to get all warnings from gcc. Warning switches that turn GCC into an effective '**lint**' are: **-Werror**, **-Wall**, **-W**, **-Wstrict-prototypes**, **-Wmissing-prototypes**, **-Wpointer-arith**, **-Wreturn-type**, **-Wcast-qual**, **-Wwrite-strings**, **-Wswitch**, **-Wshadow**, **-Wcast-align**, **-Wuninitialized**, **-Wbad-function-cast**, **-Wchar-subscripts**, **-Winline**, **-Wnested-externs**, **-Wredundant-decl**, **-ansi**, **-pedantic**. For further details on these switches, refer to the GCC manual.

4.8.2 Link-Time Errors

Are you trying to link C and Fortran code?

Compile the Fortran 77 code with the **-fno-f2c** switch. The **-fno-f2c** switch prevents the **g77** command from generating code designed to be compatible with code generated by the **f2c** command and uses the GNU calling conventions instead.

4.8.3 Run-Time Errors

Is your code causing buffer overruns?

Turn on the **-fbounds-check** switch. When the **-fbounds-check** switch is turned on, the GCC compiler generates additional code that checks whether the indices used to access arrays are or are not

within the declared range. The **-fbounds-check** switch is currently supported only by the Fortran 77 front-end, in which this option defaults to false.

Are you building a shared library?

Turn on the **-fPIC** switch if you need position-independent code suitable for use in a shared library.

4.8.4 Compiled and Linked Code Generates Unexpected Results

Does your program depend on precise floating-point behavior?

Experiment without the **-ffast-math** switch. When the **-ffast-math** is used, the compiler relaxes the rules when optimizing floating-point operations. This mode allows the compiler to further optimize floating-point code for speed, sometimes at the expense of floating-point accuracy. Do not use the **-ffast-math** switch if precise floating-point behavior is required.

Does your program rely on x87 features?

The 32-bit GCC compiler emits SSE/SSE2 code with **-mfpmath=sse**, which can yield better performance. SSE2 offers 64-bit precision, which is sufficient for most programs. If the results do not match your expectations when using SSE2, the program may rely on some x87 features.

As a diagnostic step, try building the program using x87 operations for floating-point computations, and see if the results are as expected. By omitting the **-mfpmath=sse** switch recommended in the general performance guidelines, the compiler uses **-mfpmath=387** by default.

Because omitting the **-mfpmath=sse** switch could lower performance, the user should investigate the precision requirements of the program. If the user has access to the source code, it may be possible to adapt the algorithm to SSE2.

4.8.5 Program Gives Unexpected Results or Exception Behavior

Does your code depend on exact implementation of IEEE rules or specifications for floating-point behavior?

GCC provides switches, such as the **-mieee-fp** switch, to control whether or not the compiler uses IEEE floating point comparisons.

The user should not use the **-ffast-math** optimization recommended in the general optimization guidelines in this case. Using the **-ffast-math** switch results in a fast, but less predictable, floating-point model. The user should also be careful to not use a switch that implies **-ffast-math**.

Does your code need C++ exception handling?

GCC generates the extra code needed to propagate exceptions with the **-fexceptions** switch. For some targets, propagating exceptions implies that GCC generates frame unwind information for all functions. Generating frame unwind information for all functions can produce significant data-size overhead, although it does not affect the execution of a program.

By default, GCC enables the **-fexceptions** option for languages like C++ that normally require exception handling. GCC disables the **-fexceptions** option for languages like C that do not normally require it. You may need, however, to enable this option when compiling C code that must interoperate properly with exception handlers written in C++. You may also wish to disable this option if you are compiling older C++ programs that do not use exception handling.

4.9 Intel Compilers (32-Bit) for Linux®

This section addresses errors and unexpected results that may be encountered when using 32-bit Intel compilers for Linux®.

4.9.1 Compilation Errors

Are you using the right ANSI-compliant switch?

Use the **-ansi-alias-** switch to compile Fortran programs that do not adhere to the ANSI Fortran-type alias rules.

4.9.2 Link-Time Errors

Are you trying to link C and Fortran code?

If you are linking C and Fortran modules, and the link-time error is due to a mismatch of symbol names, use the **-us** switch with the Intel Fortran compiler. Using the **-us** switch appends an underscore to the symbol names derived from external variables or functions, causing them to match the C symbols.

4.9.3 Compiled and Linked Code Generates Unexpected Results

Are you generating vectorized floating-point code?

For some loops, vectorization can cause a slight difference in results due to the reordering of floating-point operations. The switches **-xK** and **-xW** vectorize loops where possible. As a diagnostic step, try compiling without these switches.

Does your program rely on some x87 features?

Some Intel compiler switches instruct the compiler to use SSE2 registers and instructions. If the results of a program do not match your expectations when using SSE2 registers and instructions, the program may rely on some x87 features.

As a diagnostic step, try building the program using x87 operations for floating-point computations, and see if the results are as expected. Not using the **-xK** and **-xW** switches recommended in the general performance guidelines causes the compiler to build the program using x87 operations for floating-point computations.

Because not using the **-xK** and **-xW** switches could lower performance, the user should investigate the precision requirements of the program. If the user has access to the source code, it may be possible to adapt the algorithm to SSE2.

4.9.4 Program Terminates Unexpectedly

Are you using an architecture switch that is unsafe for AMD Athlon™ 64and AMD Opteron™ processors?

Some architecture switches can cause programs compiled with Intel compiler versions 7.1, 8.0, and 8.1 to terminate unexpectedly when run on AMD Athlon™ 64 and AMD Opteron™ processors. Table 8 shows 32-bit Intel compiler architecture switches that are not safe for AMD Athlon™ 64 and AMD Opteron™ processors. Try building the program without these switches.

Table 8. Unsafe Architecture Switches in 32-Bit Intel Compilers for Linux®

Compiler Version	Unsafe Architecture Switches
Intel 7.1	**-xK** and **-xW**
Intel 8.0	**-xK -xW -xP -xB** and **-xN**
Intel 8.1	**xN** and **-xP**

4.10 PathScale Compilers (32-Bit) for Linux®

For information on diagnosing problems with the PathScale compiler, please refer to the tuning document distributed with the PathScale compiler suite.

4.11 Intel Compilers (32-Bit) for Microsoft® Windows®

This section addresses errors and unexpected results that may be encountered when using 32-bit Intel compilers for Microsoft Windows.

4.11.1 Compilation Errors

Are you using the right ANSI-compliant switch?

Use the **-Qansi-alias-** switch to compile Fortran programs that do not adhere to ANSI Fortran-type alias rules.

4.11.2 Compiled and Linked Code Generates Unexpected Results

Are you generating vectorized code?

For some loops, vectorization can cause a slight difference in results due to the reordering of floating-point operations. The switches **-QxK**, **-QxW**, **-arch:SSE**, and **-arch:SSE2** cause vectorization of loops where possible. As a diagnostic step, try compiling without these switches.

Does your program rely on x87 features?

Some Intel compiler switches instruct the compiler to use SSE2 registers and instructions. If the results do not match your expectations when using SSE2, the program may rely on some x87 features. As a diagnostic step, try building the program using x87 operations for floating-point computations and see if the results are as expected. Omitting the **-QxK**, **-QxW**, and **-arch:SSE2** switches recommended in the general performance guidelines causes the compiler to build the program using x87 operations for floating-point computations.

Because omitting the **-QxK**, **-QxW**, and **-arch:SSE2** switches could lower performance, the user could investigate the precision requirements of the program. If the user has access to the source code, it may be possible to adapt the algorithm to SSE2.

4.11.3 Program Terminates Unexpectedly

Are you using an architecture switch that is unsafe for AMD Opteron™ processors?

Some architecture switches can cause programs compiled with the Intel compiler versions 7.1 and 8.0 to terminate unexpectedly when run on AMD Opteron™ processors. Table 9 shows 32-bit Intel compiler architecture switches that are not safe for the AMD Opteron™ processor. If a program built with any of the switches shown in Table 9 produces errors, try building the program without those switches.

Table 9. Unsafe Architecture Switches in 32-Bit Intel Compilers for Microsoft® Windows®

Compiler Version	Unsafe Architecture Switches
Intel 7.1	**-QxK** and **-QxW**
Intel 8.0	**-QxK -QxW -QxP -QxB** and **-QxN**
Intel 8.1	**-QxN** and **-QxP**

4.11.4 Program Gives Unexpected Results or Exception Behavior

Does your program need C++ exception handling?

By default, the Intel C++ compiler for Microsoft Windows does not turn on C++ exception handling. To enable C++ exceptions, use the **-GX** and **-GR** switches with the C++ compiler.

4.12 Microsoft® Compilers (32-Bit) for Microsoft® Windows®

This section addresses errors and unexpected results that may be encountered when using 32-bit Microsoft compilers for Microsoft Windows.

4.12.1 Run-Time Errors

Is your code causing buffer overruns that violate security?

Turn on the **/GS** switch. Turning on the **/GS** switch causes the Microsoft compiler to generate additional security code, such as bounds checking.

4.12.2 Compiled and Linked Code Generates Unexpected Results

Does your program depend on precise floating-point behavior?

Do not use the **/fp:fast** switch recommended in the general performance guidelines. When the **fp:fast** mode is enabled, the compiler relaxes the rules that **fp:precise** uses when optimizing floating-point operations. This mode allows the compiler to further optimize floating-point code for speed at the expense of floating-point accuracy.

Does your program rely on some x87 features?

The **/arch:SSE2** switch instructs the compiler to use SSE2 registers and instructions. If the results do not match your expectations when using SSE2, the program may rely on some x87 features.

As a diagnostic step, try building the program using x87 operations for floating point computations and see if the results are as expected. Omitting the **/arch:SSE2** switch recommended in the general performance guidelines causes the compiler to build the program using x87 operations for floating-point computations.

Because omitting the **/arch:SSE2** switch could degrade performance, the user should investigate the precision requirements of the program. If the user has access to the source code, it may be possible to adapt the algorithm to SSE2.

4.12.3 Program Gives Unexpected Results or Exception Behavior

Does your code depend on exact implementation of IEEE or ISO rules or specifications for floating-point behavior?

Do not use **/fp:fast** optimization, as recommended in the general performance guidelines, in this case. The compiler uses **/fp:precise** by default if no **/fp** switch is specified.

Does your code need structured and/or C++ exception handling?

Enable C++ exception handling with the appropriate /EH switch.

Are you to developing 32-bit code that you may eventually port to 64-bit code, and you would like the code to remain compatible?

Use **/Wp64** to detect 64-bit porting problems. This switch can be used with both 32-bit and 64-bit Microsoft compilers.

On AMD64 architecture-based systems running the Microsoft Windows operating system, both **int** and **long** are 32-bit, and pointers are 64-bit (P64). Do not use **int** or **long** for type-casting pointers.

Use portable, scalable data types like INT_PTR, UINT_PTR, LONG_PTR, and ULONG_PTR for type-casting pointers.

Issues such as these can be detected by using the **/Wp64** switch.

4.13 Sun Compilers (32-bit) for Solaris

This section addresses errors and unexpected results that may be encountered when using 32-bit Sun compilers for Solaris.

4.13.1 Compilation Errors

Do you need ANSI-compliant code?

If a developer needs ANSI-compliant code, Sun Studio provides several switches to detect and print errors and warnings about non-conforming constructs. The **-Xc** switch specifies ISO C compliance without K&R extensions. A number of additional switches are available to check compliance with various combinations of standards and extensions.

4.13.2 Compiled and Linked Code Generates Unexpected Results

Does your program depend on precise floating-point behavior? Does your program depend on the exact implementation of the IEEE 754 floating-point standard?

The **-fsimple[=n]** switch (implied by the **-fast** switch) may cause the compiler to generate code that does not comply with the IEEE 754 floating-point standard. To guarantee compliance with the IEEE 754 floating-point standard, this switch must be set to value 0.

Do you need access to a frame pointer register?

The compilers by default do not use the stack frame pointer register to improve performance. If this register is needed for debugging or performance analysis tools, or for C++ exceptions, it can be enabled with the **-xregs=no%frameptr** switch.

Chapter 5 Peak Options for SPEC®-CPU Benchmark Programs

This chapter enumerates the best-known peak switches (as of September 2007) for SPEC®-CPU2006 benchmarks compiled for AMD Athlon™ 64, AMD Opteron™ and AMD Family 10h processor-based platforms by different compilers.

5.1 PGI Release 7.1 32- and 64-Bit Compilers for Linux®

To translate and link SPECcpu2006 benchmarks with PGI Fortran, C, or C++ compilers the following commands are used:

- **pgcc -w** invokes the PGI C compiler
- **pgcpp -w** invokes the PGI C++ compiler
- **pgf95 -w** invokes the PGI Fortran 90/95 compiler

5.1.1 Base Command-line Options

The best-known base switches for various benchmarks in SPEC-cpu2006 suite for 64-bit PGI Release 7.1 compilers for Linux on AMD Athlon™ 64 processor based platforms,
AMD Opteron™ processor-based platforms and AMD Family 10h processor-based platforms.

The following command-line options are used for base integer component of SPECcpu2006 (CINT2006).

- 400.perlbench

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:840 -tp barcelona-64 -DSPEC_CPU_LP64 -DSPEC_CPU_LINUX_X64

- 403.gcc and 429.mcf

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:840 -tp barcelona-64

- 462.libquantum

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:840 -tp barcelona-64 -DSPEC_CPU_LP64 -DSPEC_CPU_LINUX

- 483.xalancbmk

  pgcpp -w -fastsse -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 --zc_eh -tp barcelona -DSPEC_CPU_LINUX

- All remaining integer components of CINT2006

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:840 -tp barcelona-64 -DSPEC_CPU_LP64

  pgcpp -w -fastsse -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 --zc_eh -tp barcelona -DSPEC_CPU_LP64

The following command-line options are used for the base floating-point component of SPECcpu2006 (CFP2006):

- 435.gromacs, 436.cactusADM, and 454.calculix

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -Msmartalloc=huge:448 -tp barcelona-64 -DSPEC_CPU_LP64

  pgf95 -w -fast -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 -Mnomain -tp barcelona-64 -DSPEC_CPU_LP64

- 481.wrf

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -Msmartalloc=huge:448 -tp barcelona-64 -DSPEC_CPU_CASE_FLAG -DSPEC_CPU_LINUX

- All remaining integer components of CFP2006

  pgcc -w -fast Mipa=fast, inline -Mfprelaxed -Msmartalloc=huge:448 tp barcelona-64 -DSPEC_CPU_LP64

  pgcpp -w -fast -Mipa=fast, inline -Mfprelaxed -Msmartalloc=huge:448 -zc_eh -tp barcelona-64 -DSPEC_CPU_LP64

  pgf95 -w -fast -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 -tp barcelona-64 -DSPEC_CPU_LP64

5.1.2 Peak Command-line Options

The table below specifies the best-known peak switches for various benchmarks in the SPECcpu2006 suite for the 64-bit PGI Release 7.1 compilers for Linux® on AMD Athlon™ 64 processor based platforms and AMD Opteron™ processor-based platforms.

Table 10. Best-Known Peak Switches for the 64-Bit PGI Compilers for Linux®

Application Area	Benchmark	Language	Best Known Peak Switches
CINT2006[3]			
Programming Language	*400.perlbench*	ANSI C	pgcc -w -fast -O4 -Mfprelaxed -Msmartalloc=huge:448 -Mnounroll -Mpfi(pass 1) -Mpfo(pass 2) -Mipa=inline(pass 2) -tp barcelona-64 -DSPEC_CPU_LP64 -DSPEC_CPU_LINUX_X64
Compression	*401.bzip2*	ANSI C	pgcc -w -fast -O4 -Msmartalloc=huge:448 -Mpfi(pass 1) -Mpfo(pass 2) -tp barcelona-64 -Mpfo -DSPEC_CPU_LP64
GNU C compiler	*403.gcc*	C	pgcc -w -fastsse -Mfprelaxed -Msmartalloc=huge:448 -Mipa=fast, inline -tp barcelona
Combinational Optimization	*429.mcf*	ANSI C[1]	pgcc -w -fastsse -Mipa=fast, inline:1 -Msmartalloc=huge:420 -tp barcelona
Artificial Intelligence: Go	*445.gobmk*	C	pgcc -w -fast -O4 -Msmartalloc=huge:448 -Mfprelaxed -Mnovect -tp barcelona-64 -Mpfi(pass 1) -Mpfo(pass 2) -Mipa=fast(pass 2) -DSPEC_CPU_LP64
Search Gene Sequence	*456.hmmer*	C	pgcc -w -fast -Msmartalloc=huge:448 -Mfprelaxed -Msafeptr -Mipa=const, ptr, arg -tp barcelona-64 -DSPEC_CPU_LP64
Artificial Intelligence: Chess	*458.sjeng*	ANSI C	pgcc -w -fast -Msmartalloc=huge:448 -Mfprelaxed -tp barcelona-64 -Mpfi(pass 1) -Mpfo(pass 2) -Mipa=fast, inline:1, noarg(pass 2) -DSPEC_CPU_LP64
Physics / Quantum Computing	*462.libquantum*	"C99"	pgcc -w -fast -Mfprelaxed -Msmartalloc=huge:448 -Munroll=m:4 -Mipa=fast, inline, noarg -DSPEC_CPU_LP64 -DSPEC_CPU_LINUX
Video compression	*464.h264ref*	C	Use base binaries and/or base results for peak.
Discrete Event Simulation	*471.omnetpp*	C++	Use base binaries and/or base results for peak.
Path-finding Algorithms	*473.astar*	C++	Use base binaries and/or base results for peak.

Notes:

1. *Mathematical library (libm) required*
2. *Boost Library required*
3. *Smartheap libraries utilized. If the Smartheap libraries are not loaded,* xalancbmk *performs better with the* ***-Msmartalloc=huge:160*** option.

Table 10. Best-Known Peak Switches for the 64-Bit PGI Compilers for Linux®

Application Area	Benchmark	Language	Best Known Peak Switches
XML Processing	*483.xalancbmk*	C++[3]	pgcpp -w -fastsse -O4 -Mipa=fast, inline -Mfprelaxed -Msmartalloc --zc_eh -tp Barcelona -DSPEC_CPU_LINUX
CFP2006			
Fluid Dynamics	*410.bwaves*	Fortran 77	Use base binaries and/or base results for peak.
Quantum Chemistry	*416.gamess*	Fortran	pgf95 -w -fast -Mipa=fast, inline -Mfprelaxed -Mvect=noaltcode -Msmartalloc=huge:448 -tp barcelona-64 - DSPEC_CPU_LP64
Physics/Quantum Chromodynamics	*433.milc*	C	pgcc -w -fast -O4 -Mdse -Mfprelaxed -Msmartalloc=huge:448 -Mpfi(pass 1) -Mipa=fast, inline, noarg(pass 2) -Mpfo(pass 2) -tp barcelona-64 -DSPEC_CPU_LP64
Physics / CFD	*434.zeusmp*	Fortran 77	Use base binaries and/or base results for peak.
Biochemistry / Molecular Dynamics	*435.gromacs*	C	pgcc -w -fast -Mfpapprox=rsqrt -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 -tp barcelona-64 -DSPEC_CPU_LP64
		Fortran	pgf95 -w -fast -Mfpapprox=rsqrt -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 -tp barcelona-64 -Mnomain -DSPEC_CPU_LP64
Physics / General Relativity	*436.cactusADM*	ANSI C	Use base binaries and/or base results for peak.
		Fortran 90	Use base binaries and/or base results for peak.
Fluid Dynamics	*437.leslie3d*	Fortran 90	Use base binaries and/or base results for peak.
Biology / Molecular Dynamics	*444.namd*	C++	pgcpp -w -fast -O4 -Mfprelaxed -Msmartalloc=huge:448 -zc_eh -tp barcelona-64 -Mnodepchk -Mprefetch -Msafe_lastval -Msafeptr=static -Mstride0 -Munroll=n:4 -Mvect=noidiom -Mvect=prefetch -DSPEC_CPU_LP64
Finite Element Analysis	*447.dealII*	C++[2]	pgcpp -w -fast -Mfprelaxed -Msmartalloc=huge:448 --zc_eh -Mnovect -alias=ansi -Mipa=fast,inline -tp barcelona-64 -DSPEC_CPU_LP64
Linear Programming, Optimization	*450.soplex*	ANSI C++	Use base binaries and/or base results for peak.
Image Ray-Tracing	*453.povray*	ISO C++	Use base binaries and/or base results for peak.
Structural Mechanics	*454.calculix*	C	Use base binaries and/or base results for peak.
		Fortran90	Use base binaries and/or base results for peak.

Notes:

1. *Mathematical library (libm) required*
2. *Boost Library required*
3. *Smartheap libraries utilized. If the Smartheap libraries are not loaded,* xalancbmk *performs better with the* ***-Msmartalloc=huge:160*** option.

Table 10. Best-Known Peak Switches for the 64-Bit PGI Compilers for Linux®

Application Area	Benchmark	Language	Best Known Peak Switches
Computational Electromagnetics	*459.GemsFDTD*	Fortran 90	pgf95 -w -fast -O4 -Mdse -Mipa=fast,inline -Mfprelaxed -Msmartalloc=huge:448 -tp barcelona-64 -DSPEC_CPU_LP64
Quantum Chemistry	*465.tonto*	Fortran 95	pgf95 w -fast -O4 -Mfprelaxed -Msmartalloc=huge:448 -Mipa=fast,inline -Mvect=noaltcode tp barcelona 64 DSPEC_CPU_LP64
Fluid Dynamics	*470.lbm*	ANSI C	Use base binaries and/or base results for peak.
Weather	*481.wrf*	C	pgcc -w -fast -Mfprelaxed -Msmartalloc=huge:448 -Mvect=noaltcode -tp barcelona-64 -DSPEC_CPU_LP64
		Fortran 90	pgf95 -w -fast -Mfprelaxed -Msmartalloc=huge:448 -Mvect=noaltcode -tp barcelona-64 -DSPEC_CPU_LP64
Speech recognition	*482.sphinx3*	C	Use base binaries and/or base results for peak.

Notes:

1. *Mathematical library (libm) required*
2. *Boost Library required*
3. *Smartheap libraries utilized. If the Smartheap libraries are not loaded,* xalancbmk *performs better with the* ***-Msmartalloc=huge:160*** option.

5.2 PGI Release 7.1 Compilers (32- and 64-Bit) for Microsoft® Windows®

5.2.1 Invoking the Compilers

To translate and link SPECcpu2006 benchmarks with PGI Fortran, C, or C++ compilers the following commands are used:

- pgcc -w invokes the PGI C compiler
- pgcpp -w invokes the PGI C++ compiler
- pgf95 -w invokes the PGI Fortran 90/95 compiler

5.2.2 Base Command-line Options

The best-known base switches for various benchmarks in SPECcpu2006 suite for 64-bit PGI Release 7.1 compilers for Linux on AMD Athlon™ 64 , AMD Opteron™ and AMD Family 10h processor-based platforms. The following command-line options are used for base integer component of SPECcpu2006 (CINT2006).

By default all benchmark programs use the following option:

OPTIMIZE = -stack=nocheck,39000000,39000000

Note: *INT base C++ are compiled as 32-bit binaries. SmartHeap libraries are required for INT C++ base.*

- 400.perlbench

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:16 -tp barcelona-64 -DSPEC_CPU_WIN64_X64

- 403.gcc

  pgcc w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:16 -tp barcelona-64 -DSPEC_CPU_WIN64 -DSPEC_CPU_NEED_ALLOCA_H

- 462.libquantum

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:16 -tp barcelona-64 -DSPEC_CPU_COMPLEX_I

- 464.h264ref

  pgcc -w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:16 -tp barcelona-64 -DSPEC_CPU_NO_INTTYPES -DWIN32

- 471.omnetpp

  pgcpp -w -fastsse -Mipa=fast, inline -Mfprelaxed --zc_eh -tp barcelona -DSPEC_CPU_WIN64_X64

- 483.xalancbmk

  pgcpp -w -fastsse -Mipa=fast, inline -Mfprelaxed --zc_eh -tp barcelona -DSPEC_CPU_XMLCH_IS_NOT_UNSIGNED_SHORT

- All remaining integer components of CINT2006

  pgcc w -fast -Mipa=fast, inline, noarg -Mfprelaxed -Msmartalloc=huge:16 -tp barcelona-64 -DSPEC_CPU_WIN64_X64

  pgcpp w -fastsse -Mipa=fast, inline -Mfprelaxed --zc_eh -tp barcelona - DSPEC_CPU_WIN64_X64

The following command-line options are used for base floating point component of SPECcpu2006 (CFP2006).

- 435.gromacs

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed - tp barcelona-64 -DSPEC_CPU_APPEND_UNDERSCORE -DSPEC_CPU_HAVE_ERF

  pgf**95 -w -fast -Mipa=fast,inline -Mfprelaxed -Mnomain -tp-barcelona-64 -DSPEC_CPU_WIN64_X64**

- 436.cactusADM

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -tp barcelona-64 -DSPEC_CPU_WIN64_X64

  pgf95 -w -fast -Mipa=fast,inline -Mfprelaxed -Mnomain -tp barcelona-64 -DSPEC_CPU_WIN64_X64

- 453.povray

  pgcpp -w -fast -Mipa=fast, inline -Mfprelaxed -zc_eh -tp barcelona-64 -DSPEC_CPU_INVHYP -DNEED_INVHYP

- 454.calculix

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -tp barcelona-64 -DSPEC_CPU_APPEND_UNDERSCORE -DSPEC_CPU_NOZMODIFIER

  pgf95 -w -fast -Mipa=fast,inline -Mfprelaxed -Mnomain -tp barcelona-64 -DSPEC_CPU_WIN64_X64

- 481.wrf

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -tp barcelona-64 -DSPEC_CPU_CASE_FLAG -DSPEC_CPU_NEEDIO_H

- All remaining floating point components of CFP2006

  pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -tp barcelona-64 -DSPEC_CPU_P64

  pgcpp -w -fast -Mipa=fast, inline -Mfprelaxed -zc_eh -tp barcelona-64 -DSPEC_CPU_WIN64_X64

  pgf95 -w -fast -Mipa=fast,inline -Mfprelaxed -tp barcelona-64 -DSPEC_CPU_WIN64_X64

5.2.3 Peak Command-line Options

The table below delineates the best-known peak switches for various benchmarks in the SPECcpu2006 suite for the 64-bit PGI Release 7.1 compilers for Windows® on AMD Athlon™ 64, AMD Opteron™ and Amd Family 10h processor-based platforms.

Table 11. Best-Known Peak Switches for the 64-Bit PGI Compilers for Microsoft® Windows®

Application Area	Benchmark	Language	Best Known Peak Switches
CINT2006[3]			
Programming Language	*400.perlbench*	ANSI C	pgcc -w -fast -O4 -Mfprelaxed -Mnounroll -Mnodll -Mpfi(pass 1) -Mpfo(pass 2) -Mipa=inline(pass 2) -tp barcelona-64 -DSPEC_CPU_LP64 -DSPEC_CPU_WIN64_X64
Compression	*401.bzip2*	ANSI C	pgcc -w -fast -O4 -Msmartalloc=huge:8 -Mnodll -tp barcelona-64 -Mpfi(pass 1) -Mpfo(pass 2) -DSPEC_CPU_P64
GNU C compiler	*403.gcc*	C	pgcc -w -fastsse -Mfprelaxed -Mnodll -Mpfi(pass 1) -Mpfo(pass 2) -Mipa=fast, inline(pass 2) -tp barcelona -DSPEC_CPU_WIN32 -DSPEC_CPU_NEEDALLOCA_H
Combinational Optimization	*429.mcf*	ANSI C[1]	pgcc -w -fastsse -Mipa=fast, inline:1 Mnodll -tp barcelona[4]
Search Gene Sequence	*456.hmmer*	C	pgcc -w -fast -Msmartalloc=huge:8 -Mfprelaxed -Msafeptr -Mipa=const, ptr, arg Mnodll -tp barcelona-64 -DSPEC_CPU_P64
Artificial Intelligence: Chess	*458.sjeng*	ANSI C	pgcc -w -fast -Msmartalloc=huge:8 -Mfprelaxed -Mnodll -tp barcelona-64 -Mpfi(pass 1) -Mpfo(pass 2) -Mipa=fast, inline:1, noarg(pass 2) -DSPEC_CPU_P64
Physics / Quantum Computing	*462.libquantum*	"C99"	pgcc -w -fast -Mfprelaxed -Msmartalloc=huge:8 -Munroll=m:4 -Mipa=fast, inline, noarg -Mnodll -DSPEC_CPU_P64 -DSPEC_CPU_COMPLEX_I
Video compression	*464.h264ref*	C	Use base binaries and/or base results for peak.
Discrete Event Simulation	*471.omnetpp*	C++	Use base binaries and/or base results for peak.
Path-finding Algorithms	*473.astar*	C++	Use base binaries and/or base results for peak.

Notes:

1. *Mathematical library (libm) required.*
2. *Boost Library required.*
3. *SmartHeap libraries utilized.*
4. *SmartHeap library is used.*

Table 11. Best-Known Peak Switches for the 64-Bit PGI Compilers for Microsoft® Windows®

Application Area	Benchmark	Language	Best Known Peak Switches
XML Processing	*483.xalancbmk*	C++	Use base binaries and/or base results for peak and also **srcalt=pgiwin**.
CFP2006			
Fluid Dynamics	*410.bwaves*	Fortran 77	Use base binaries and/or base results for peak.
Quantum Chemistry	*416.gamess*	Fortran	pgf95 -w -fast -Mipa=fast, inline -Mfprelaxed -Mnovect -Mnodll -tp barcelona-64 - DSPEC_CPU_P64
Physics/Quantum Chromodynamics	*433.milc*	C	pgcc -w -fast -O4 -Mdse -Mfprelaxed -Msmartalloc=huge:448 -Mpfi(pass 1) -Mipa=fast, inline, noarg(pass 2) -Mpfo(pass 2) -tp barcelona-64 -DSPEC_CPU_LP64
Physics / CFD	.*434.zeusmp*	Fortran 77	Use base binaries and/or base results for peak.
Biochemistry / Molecular Dynamics	*435.gromacs*	C	pgcc -w -fast Mfpapprox=rsqrt -Mipa=fast,inline -Mfprelaxed -Msmartalloc -Mnodll -tp barcelona-64 LDPORTABILITY = -Mnomain CPORTABILITY=-DSPEC_CPU_APPEND_UNDERSCORE -DSPEC_CPU_HAVE_ERF srcalt=have_erf
		Fortran	pgf95 -w -fast -Mfpapprox=rsqrt -Mipa=fast,inline -Mfprelaxed -Msmartalloc -Mnodll-tp barcelona-64 -Mnomain -DSPEC_CPU_P64
Physics / General Relativity	*436.cactusADM*	ANSI C	Use base binaries and/or base results for peak.
		Fortran 90	Use base binaries and/or base results for peak.
Fluid Dynamics	*437.leslie3d*	Fortran 90	Use base binaries and/or base results for peak.
Biology / Molecular Dynamics	*444.namd*	C++	pgcpp -w -fast -O4 -Mfprelaxed -Msmartalloc -zc_eh -tp barcelona-64 -Mnodepchk -Mprefetch -Msafe_lastval -Msafeptr=static -Mstride0 -Munroll=n:4 -Mvect=noidiom -Mvect=prefetch -DSPEC_CPU_P64
Finite Element Analysis	*447.dealII*	C++[2]	pgcpp -w -fast -Mfprelaxed -Msmartalloc -zc_eh -Mnovect -alias=ansi -Mipa=fast,inline -Mnodll -tp barcelona-64 -DSPEC_CPU_P64
Linear Programming, Optimization	*450.soplex*	ANSI C++	pgcpp -w -fast -Mipa=fast, inline -Mfprelaxed -zc_eh -Mnodll -tp barcelona-64 -DSPEC_CPU_P64

Notes:

1. *Mathematical library (libm) required.*
2. *Boost Library required.*
3. *SmartHeap libraries utilized.*
4. *SmartHeap library is used.*

Table 11. Best-Known Peak Switches for the 64-Bit PGI Compilers for Microsoft® Windows®

Application Area	Benchmark	Language	Best Known Peak Switches
Image Ray-Tracing	*453.povray*	ISO C++	Use base binaries and/or base results for peak.
Structural Mechanics	*454.calculix*	C	Use base binaries and/or base results for peak.
		Fortran90	Use base binaries and/or base results for peak.
Computational Electromagnetics	*459.GemsFDTD*	Fortran 90	pgf95 -w -fast -O4 -Mdse -Mipa=fast,inline -Mfprelaxed -Mnodll -tp barcelona-64 -DSPEC_CPU_P64
Quantum Chemistry	*465.tonto*	Fortran 95	pgf95 w -fast -O4 -Mfprelaxed -Msmartalloc -Mipa=fast,inline -Mvect=noaltcode -Mnodll -tp barcelona-64 -DSPEC_CPU_P64
Fluid Dynamics	*470.lbm*	ANSI C	Use base binaries and/or base results for peak.
Weather	*481.wrf*	C	pgcc -w -fast -Mfprelaxed -Msmartalloc -Mvect=noaltcode -Mnodll -tp barcelona-64 -DSPEC_CPU_P64 CPORTABILITY=-DSPEC_CPU_CASE_FLAG -DSPEC_CPU_NEED_IO_H srcalt=need_io_h
		Fortran 90	pgf95 -w -fast -Mfprelaxed -Msmartalloc -Mvect=noaltcode -Mnodll -tp barcelona-64 -DSPEC_CPU_P64
Speech recognition	*482.sphinx3*	C	pgcc -w -fast -Mipa=fast, inline -Mfprelaxed -Mnodll -tp barcelona-64 -DSPEC_CPU_P64

Notes:
1. *Mathematical library (libm) required.*
2. *Boost Library required.*
3. *SmartHeap libraries utilized.*
4. *SmartHeap library is used.*

5.3 SuSE GCC 4.2.0(64-Bit) C/C++ Compiler for Linux®

Table 12 shows the best-known peak switches for various benchmarks in the SPEC-CPU2000 suite for the SuSE 64-bit GCC C/C++ compiler for Linux® on AMD Athlon™ 64 processor-based platforms and AMD Opteron™ processor-based platforms. For AMD Family 10h processor-based platforms, add the **-march=amdfam10** switch.

Table 12. Best-Known Peak Switches for the 64-Bit SuSE GCC 3.3.3 C/C++ Compiler for Linux®

Benchmark Program	Best-Known Peak Switches
Note: *The **-m32** switch improves the performance of 181.mcf, 197.parser and 300.twolf by reducing memory footprint.*	

Table 12. Best-Known Peak Switches for the 64-Bit SuSE GCC 3.3.3 C/C++ Compiler for Linux® (Continued)

164.gzip:	**-O3 -funroll-all-loops -finline-limit=900 -freduce-all-givs** and **-fprofile-arcs/-fbranch-probabilities**
175.vpr:	**-O3 -funroll-all-loops -finline-limit=1000** and **-fprofile-arcs/-fbranch-probabilities**
176.gcc:	**-O3 -funroll-all-loops -finline-limit=900** and **-fprofile-arcs/-fbranch-probabilities**
181.mcf:	**-O3 -funroll-all-loops -m32**, and **-fprofile-arcs/-fbranch-probabilities**
186.crafty:	**-O3 -funroll-all-loops** and **-fprefetch-loop-arrays**
197.parser:	**-O3 -funroll-all-loops -m32**, and **-fprofile-arcs/-fbranch-probabilities**
252.eon:	**-O3 -funroll-all-loops -ffast-math -finline-limit=3000** and **-fprofile-arcs/-fbranch-probabilities**
253.perlbmk:	**-O3 -funroll-all-loops -finline-limit=1000** and **-fprofile-arcs/-fbranch-probabilities**
254.gap:	**-O3 -funroll-all-loops** and **-fprofile-arcs/-fbranch-probabilities**
255.vortex:	**-O3 -funroll-all-loops -finline-limit=1000** and **-fprofile-arcs/-fbranch-probabilities**
256.bzip2:	**-O3 -funroll-all-loops -freduce-all-givs -finline-limit=2700** and **-fprofile-arcs/-fbranch-probabilities**
300.twolf:	**-O3 -funroll-all-loops -freduce-all-givs -finline-limit=2000** and **-fprofile-arcs/-fbranch-probabilities**
17.mesa:	**-O3 -funroll-all-loops -finline-limit=2000** and **-fprofile-arcs/-fbranch-probabilities**
179.art:	**-O3 -funroll-all-loops -ffast-math -finline-limit=1500** and **-fprofile-arcs/-fbranch-probabilities**
183.equake:	**-O3 -funroll-all-loops -ffast-math -finline-limit=2000** and **-fprofile-arcs/-fbranch-probabilities**
188.ammp:	**-O3 -funroll-all-loops -ffast-math -finline-limit=2000** and **-fprofile-arcs/-fbranch-probabilities**
Note: *The* ***-m32*** *switch improves the performance of 181.mcf, 197.parser and 300.twolf by reducing memory footprint.*	

5.4 Pathscale EKO 3.0 C/C++ Compiler (64-Bit) for Linux®

Table 13 shows the best-known peak switches for various benchmarks in the SPEC-CPU2000 suite for the PathScale C/C++ compiler (64-bit) for Linux® on AMD Athlon™ 64 processor-based platforms and AMD Opteron™ processor-based platforms.

Table 13. Best-Known Peak Switches for the Pathscale 1.4 C/C++ Compiler for Linux®

Benchmark Program	Best-Known Peak Switches
164.gzip:	**-O3 -ipa -m3dnow -WOPT:val=0 +FDO**
175.vpr:	**-O2 -ipa -OPT:alias=disjoint -CG:p2align_freq=500000 +FDO -INLINE:aggressive=on -IPA:space=300:plimit=10000:callee_limit=5000:linear=on**
176.gcc:	**-O3 -ipa -OPT:goto=off +FDO**
181.mcf:	**-O3 -ipa -IPA:field_reorder=on -m32 +FDO**
186.crafty:	**-O3 -OPT:goto=off +FDO**
197.parser:	**-O3 -ipa -m32 -IPA:ctype=on +FDO**
252.eon:	**-Ofast -CG:gcm=off:p2align_freq=1:prefetch=off -OPT:treeheight=on -LNO:fu=10:full_unroll_outer=on -TENV:X=4:frame_pointer=off -fno-exceptions -IPA:plimit=4000 +FDO**
253.perlbmk:	**-O3 -ipa +FDO -IPA:plimit=10000**
254.gap:	**-Ofast -WOPT:aggstr=off +FDO**
255.vortex	**-Ofast -IPA:plimit=1800 -OPT:goto=off -CG:p2align=on +FDO**
256.bzip2	**-Ofast +FDO**
300.twolf	**-O2 -CG:gcm=off:p2align_freq=100000 -WOPT:mem_opnds=on +FDO -m32 -OPT:unroll_times_max=8:unroll_size=256:alias=disjoint:Ofast**
177.mesa	**-O2 -ipa -OPT:Ofast -fno-math-errno -CG:local_fwd_sched=on +FDO**
179.art	**-O3 -OPT:ro=2:div_split=on:alias=typed -fno-math-errno -m32 +FDO**
183.equake	**-Ofast -WOPT:mem_opnds=on -m32**
188.ammp	**-O3 -OPT:alias=disjoint:unroll_times_max=8:Ofast:ro=3 -fno-math-errno -TENV:X=4 +FDO**
Note: *FDO is feedback optimization-PASS1= -fb_create fbdata and PASS2= -fb_opt fbdata.*	

5.5 Pathscale EKO 3.0 Fortran Compiler (64-bit) for Linux®

Table 14 shows the best-known peak switches for various benchmarks in the SPEC-CPU2000 suite for the Pathscale Fortran compiler (64-bit) for Linux® on AMD Athlon™ 64 processor-based platforms and AMD Opteron™ processor-based platforms.

Table 14. Best-Known Peak Switches for the 64-bit Pathscale 2.4 Fortran Compiler for Linux®

Benchmark Program	Best-Known Peak Switches
168.wupwise:	**-Ofast -LNO:prefetch_ahead=5:prefetch=3** **-OPT:unroll_times_max=8:unroll_size=128:IEEE_NaN_Inf=off:ro=3** **-TENV:X=4 +FDO** **-IPA:space=1000:linear=on:plimit=50000:callee_limit=5000** **-INLINE:aggressive=on**
171.swim:	**-Ofast -LNO:fusion=2 -m3dnow**
172.mgrid:	**-O3 -LNO:fusion=2:blocking=off** **-OPT:Ofast:unroll_times_max=8:unroll_size=256:ro=3** **-CG:gcm=off:cflow=off -m3dnow**
173.applu:	**-Ofast -CG:local_fwd_sched=on** **-LNO:fusion=2:fission=2:full_unroll_size=10000:prefetch=3** **-OPT:ro=3 -TENV:X=3 -WOPT:val=2**
178.galgel:	**-Ofast -OPT:fast_complex -CG:use_movlpd=on +ACML**
187.facerec:	**-Ofast -OPT:treeheight=on:IEEE_NaN_Inf=off:ro=3 -CG:load_exe=0** **-LNO:fusion=2 -IPA:plimit=1500 -WOPT:if_conv=off +FDO**
189.lucas:	**-Ofast -CG:local_fwd_sched=on -LNO:fusion=2** **+FDO**
191.fma3d:	**-O2 -ipa -WOPT:mem_opnds=on:retype_expr=on -CG:load_exe=1** **-OPT:Ofast:IEEE_arith=3:ro=3 +FDO -IPA:pu_reorder=1**
200.sixtrack:	**-O3 -CG:load_exe=1 -OPT:Ofast:Olimit=6000 -fno-math-errno +FDO**
301.apsi:	**-Ofast -TENV:X=4 -LNO:fusion=2:prefetch=0**
***Note: +FDO** is feedback optimization—PASS1= **-fb_create fbdata** and PASS2= **-fb_opt fbdata**.*	

5.6 Intel 9.0 C/C++ Compiler for (32-Bit) Microsoft® Windows®

Table 15 shows the best-known peak switches for various programs in the SPEC-CPU2000 benchmarks for the 32-bit Intel 8.0 C/C++ compiler for Microsoft Windows on AMD Athlon™ 64 processor-based platforms and AMD Opteron™ processor-based platforms.

Table 15. Best-Known Peak Switches for the 32-Bit Intel 8.0 C/C++ Compiler for Microsoft® Windows®

Benchmark Program	Best-Known Peak Switches
164.gzip:	**-fast, -arch:SSE, shlW32M6.lib, and -prof_gen/-prof_use**
175.vpr:	**-fast, -arch:SSE2, -prof_gen/-prof_use, -Qoption,c,-ip_ninl_max_stats=2000, and -Qoption,c,-ip_ninl_max_total_stats=4500**
176.gcc:	**-fast, -arch:SSE2, -prof_gen/-prof_use, -Oi-, and -Qunroll3**
181.mcf:	**-fast, -QaxN, and -prof_gen/-prof_use**
186.crafty:	**-fast**, **-arch:SSE2**, and **-prof_gen/-prof_use**
197.parser:	**-arch:SSE2**, **-prof_gen/-prof_use**, **-Oi-**, and **-Qipo**
252.eon:	**-fast -arch:SSE2 -prof_gen/-prof_use -Qansi_alias**, **-Qoption,c,-ip_ninl_max_stats=2000** and **-Qoption,c,-ip_ninl_max_total_stats=4500**
253.perlbmk:	**-arch:SSE2 -prof_gen/-prof_use -Qipo** and **shlW32M6.lib**
254.gap:	**-fast -arch:SSE2 -prof_gen/-prof_use -Oi- -Oa -Qoption,c,-ip_ninl_max_stats=500** and **-Qoption,c,-ip_ninl_max_total_stats=3000**
255.vortex:	**-fast -arch:SSE -prof_gen/-prof_use -Oi- shlW32M6.lib -Qoption,c,-ip_ninl_max_stats=2000** and **-Qoption,c,- ip_ninl_max_total_stats=4500**
256.bzip2:	**-fast** and **-Qunroll2**
300.twolf:	**-fast -arch:SSE2 -prof_gen/-prof_use -Qunroll3 shlW32M6.lib** and **-Qansi_alias**
177.mesa:	**-Qipo -arch:SSE2 -Qunroll1 -Qansi_alias -Qoption,f,-ip_ninl_max_stats=1500 -Qoption,f,-ip_ninl_max_total_stats=4500** and **-Qprof_gen/-Qprof_use**
179.art:	**-Qipo** and **-Zp4**
183.equake:	**-fast -arch:SSE2 -QaxW -Qansi_alias** and **-Qprof_gen/-Qprof_use**
188.ammp:	**-Oa -arch:SSE2 -Zp4 -Qansi_alias** and **-Qprof_gen/-Qprof_use**

5.7 Sun C/C++ Compiler (64-bit) for Solaris

Table 16 shows the best-known peak switches for various programs in the SPEC-CPU2000 benchmarks for the 64-bit Sun C and C++ compilers (version 5.7) for Solaris on AMD Athlon™ 64 processor-based platforms and AMD Opteron™ processor-based platforms.

Table 16. Best-Known Peak Switches for the 64-bit Sun C/C++ Compilers for Solaris

Benchmark Program	Best-Known Peak Switches
164.gzip:	**-fast -xpagesize=2m -xcrossfile -M /usr/lib/ld/map.bssalign**
175.vpr:	**-fast -xpagesize=2m -W2,-Ainline:inc=200:cs=500 -M /usr/lib/ld/map.bssalign -lmopt**
176.gcc:	**-fast -xpagesize=2m -M /usr/lib/ld/map.bssalign**
181.mcf:	**-fast -xpagesize=2m -xcrossfile -M /usr/lib/ld/map.bssalign**
186.crafty:	**-fast -xpagesize=2m -xcrossfile -xarch=amd64 -M /usr/lib/ld/map.bssalign -lbsdmalloc**
197.parser:	**-fast -xpagesize=2m -xipo=2 -W2,-Ainline:inc=200:cs=500 -M /usr/lib/ld/map.bssalign**
252.eon:	**-fast -xpagesize=2m -xcrossfile -Qoption ube -ZB -Qoption ube -xcallee=yes -xarch=amd64 -M /usr/lib/ld/map.bssalign**
253.perlbmk:	**-fast -xcrossfile -M /usr/lib/ld/map.bssalign -lbsdmalloc**
254.gap:	**-Xc -fast -xipo=2 -M /usr/lib/ld/map.bssalign**
255.vortex:	**-fast -xcrossfile -xarch=amd64 -Xc -Wu,-ZB -Wu,-xcallee=yes -M /usr/lib/ld/map.bssalign**
256.bzip2:	**-fast -xpagesize=2m -xcrossfile -xarch=sse2 -Xc -M /usr/lib/ld/map.bssalign -lbsdmalloc**
300.twolf:	**-fast -xpagesize=2m -xcrossfile -M /usr/lib/ld/map.bssalign**
177.mesa:	**-fast -xipo=2 -xarch=amd64 -xalias_level=strong -xpagesize=2m +FDO**
179.art:	**-fast -xipo=2 -xarch=amd64 -xalias_level=std -xpagesize=2m -Xc -M /usr/lib/ld/map.bssalign -lm**
183.equake:	**-fast -xipo=2 -xprefetch -xalias_level=strong -xpagesize=2m -lmopt -lm +FDO**
188.ammp:	**-fast -xipo=2 -xarch=amd64 -xalias_level=std -xpagesize_heap=2m -lmopt -lm**
***Note: FDO** is feedback optimization—PASS1= **-xprofile=collect** and PASS2= **-xprofile=use**.*	

5.8 Sun Fortran Compiler (64-bit) for Solaris

Table 17 shows the best-known peak switches for various programs in the SPEC-CPU2000 benchmarks for the 64-bit Sun Fortran compiler (version 5.7) for Solaris on AMD Athlon™ 64 processor-based platforms and AMD Opteron™ processor-based platforms.

Table 17. Best-Known Peak Switches for the 64-bit Sun Fortran Compiler for Solaris

Benchmark Program	Best-Known Peak Switches
168.wupwise:	**-fast -xipo=2 -xarch=amd64 -xprefetch_level=3 -xpagesize_heap=2m**
171.swim:	**-fast -xipo=2 -xprefetch_level=3 -Qoption iropt -Atile:skewp,-Ainline:cs=700 -xarch=amd64 -qoption ube_ipa -inl_alt -xpagesize_stack=2m**
172.mgrid:	**-fast -xipo=2 -xarch=amd64 -xprefetch_level=3 -xvector -xpagesize=2m -Qoption ld -M,/usr/lib/ld/map.bssalign**
173.applu:	**-fast -xipo=2 -xprefetch_level=3 -xarch=amd64 -Qoption iropt -Aujam:inner=g -xpagesize_heap=2m**
178.galgel:	**-fast -xipo=2 -xprefetch_level=3 -xvector=simd -xarch=amd64 -xpagesize_heap=2m -Qoption ld -M,/usr/lib/ld/map.bssalign -xlic_lib=sunperf**
187.facerec:	**-fast -xipo=2 -xprefetch_level=3 -xpagesize=2m -xlic_lib=sunperf**
189.lucas:	**-fast -xprefetch_level=3 -Qoption ld -M,/usr/lib/ld/map.bssalign -xpagesize_stack=2m**
191.fma3d:	**-fast -xipo=2 -xprefetch_level=3 -xarch=amd64 -xpagesize_heap=2m +FDO**
200.sixtrack:	**-fast -xipo=2 -xprefetch_level=3 -xarch=amd64 -xpagesize_heap=2m -Qoption ld -M,/usr/lib/ld/map.bssalign +FDO**
301.apsi:	**-fast -xipo=2 -xprefetch_level=3 -xarch=amd64 -xpagesize=2m**
***Note: +FDO** is feedback optimization—PASS1= **-xprofile=collect** and PASS2= **-xprofile=use**.*	