PGI® Server 7.1
PGI® Workstation 7.1

Release Notes

The Portland Group™
STMicroelectronics, Inc
Two Centerpointe Drive
Lake Oswego, OR 97035
www.pgroup.com

While every precaution has been taken in the preparation of this document, The Portland Group™ (PGI®), a wholly-owned subsidiary of STMicroelectronics, Inc., makes no warranty for the use of its products and assumes no responsibility for any errors that may appear, or for damages resulting from the use of the information contained herein. STMicroelectronics, Inc. retains the right to make changes to this information at any time, without notice. The software described in this document is distributed under license from STMicroelectronics, Inc. and may be used or copied only in accordance with the terms of the license agreement. No part of this document may be reproduced or transmitted in any form or by any means, for any purpose other than the purchaser's personal use without the express written permission of STMicroelectronics, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this manual, STMicroelectronics was aware of a trademark claim. The designations have been printed in caps or initial caps.

PGF95, *PGF90* and *PGC++* are trademarks and *PGI*, *PGHPF*, *PGF77*, *PGCC*, *PGPROF*, and *PGDBG* are registered trademarks of STMicroelectronics, Inc. *Other brands and names are the property of their respective owners.

PGI Server 7.1 / PGI Workstation 7.1

Release Notes

Copyright © 2007-2008

The Portland Group™

STMicroelectronics, Inc. - All rights reserved.

Printed in the United States of America

First Printing:	Release 7.1-1, October, 2007
Second Printing:	Release 7.1-2, November, 2007
Third Printing:	Release 7.1-3, December, 2007
Fourth Printing:	Release 7.1-4, January, 2008
Fifth Printing:	Release 7.1-5, February, 2008

Technical support: http://www.pgroup.com/support

Table of Contents

1 Introduction

Welcome to Release 7.1 of *PGI Workstation* and *PGI Server*, a set of Fortran, C, and C++ compilers and development tools for 32-bit and 64-bit x86-compatible processor-based workstations and servers running versions of the Linux, Windows, and Mac OS operating systems.

All workstation-class compilers and tools products from The Portland Group (*PGHPF Workstation*, for example) are subsets of the *PGI Workstation Complete* product. These workstation-class products provide a node-locked single-user license, meaning one user at a time can compile on the one system on which the *PGI Workstation* compilers and tools are installed.

PGI Server products are offered in configurations identical to the workstation-class products, but provide network-floating multi-user licenses. This means that two or more users can use the *PGI* compilers and tools concurrently on any compatible system networked to the system on which the *PGI Server* compilers are installed.

These release notes apply to all workstation-class and server-class compiler products from The Portland Group.

1.1 Product Overview

Release 7.1 of PGI Workstation and PGI Server includes the following components:

- *PGF95* OpenMP* and auto-parallelizing Fortran 90/95 compiler.
- *PGF77* OpenMP and auto-parallelizing FORTRAN 77 compiler.
- *PGHPF* data parallel High Performance Fortran compiler.
  NOTE: *PGHPF is supported only on Linux platform.*

- *PGCC* OpenMP and auto-parallelizing ANSI C99 and K&R *C* compiler.
- *PGC++* OpenMP and auto-parallelizing ANSI *C++* compiler.
- *PGPROF* graphical MPI/OpenMP/multi-thread performance profiler.
- *PGDBG* graphical MPI/OpenMP/multi-thread symbolic debugger
- *MPICH MPI libraries, version 1.2.7,* for both 32-bit and 64-bit development environments (Linux only)
- Online documentation in PDF, HTML and `man` page formats.
- A UNIX*-like shell environment for *Win32* and *Win64* platforms.

Depending on the product configuration you purchased, you may not have licensed all of the above components.

The MPI profiler and debugger included with *PGI Workstation* are limited to processes on a single node. *PGI Workstation* can be installed on a single computer, and that computer can be used to develop, debug, and profile MPI applications. The *PGI CDK Cluster Development Kit* supports general development on clusters.

1.2 Terms and Definitions

Following are definitions of terms used in the context of these release notes.

AMD64 – a 64-bit processor from AMD designed to be binary compatible with 32-bit *x86* processors, and incorporating new features such as additional registers and 64-bit addressing support for improved performance and greatly increased memory range. This includes the AMD* Athlon64*, AMD Opteron* and AMD Turion* processors.

Barcelona – In this document the Quad-Core AMD Opteron(TM) Processor (i.e. Opteron Rev x10) is referred to as Barcelona.

DLL – a dynamic linked library on *Win32* or *Win64* platforms of the form *xxx.dll* containing objects that are dynamically linked into a program at the time of execution.

driver – the compiler *driver* controls the compiler, linker, and assembler, and adds objects and libraries to create an executable. The *–dryrun* option illustrates operation of the driver. `pgf77`, `pgf95`, `pghpf`, `pgcc`, `pgCC` (Linux), and `pgcpp` are drivers for the PGI compilers. A `pgf90` driver is

retained for compatibility with existing makefiles, even though `pgf90` and `pgf95` drivers are identical.

Dual-, Quad-, or Multi-core – some x64 CPUs incorporate two or four complete processor cores (functional units, registers, level 1 cache, level 2 cache, etc) on a single silicon die. These are referred to as Dual-core or Quad-core (in general, Multi-core) processors. For purposes of OpenMP or auto-parallel threads, or MPI process parallelism, these cores function as distinct processors. However, the processing cores are on a single chip occupying a single socket on the system motherboard. In PGI 7.1, there are no longer software licensing limits on OpenMP threads for Multi-core.

EM64T – a 64-bit IA32 processor with *Extended Memory 64-bit Technology* extensions designed to be binary compatible with AMD64 processors. This includes Intel Pentium 4, Intel Xeon, and Intel Core 2 processors.

FLEXlm – The flexible license management software from Macrovision.

Hyperthreading (HT) – some IA32 CPUs incorporate extra registers that allow 2 threads to run on a single CPU with improved performance for some tasks. This is called *hyperthreading* and abbreviated *HT*. Some *linux86* and *linux86-64* environments treat IA32 CPUs with HT as though there were a 2nd *pseudo* CPU, even though there is only one physical CPU. Unless the Linux kernel is *hyperthread-aware*, the second thread of an *OpenMP* program will be assigned to the *pseudo* CPU, rather than a real second physical processor (if one exists in the system). *OpenMP* Programs can run very slowly if the second thread is not properly assigned.

IA32 – an Intel Architecture 32-bit processor designed to be binary compatible with *x86* processors, but incorporating new features such as streaming SIMD extensions (SSE) for improved performance. This includes Intel Pentium 4, Intel Xeon, and Intel Core 2 processors. For simplicity, these release notes refer to *x86* and *IA32* processors collectively as *32-bit x86* processors.

Large Arrays – arrays with aggregate size larger than 2GB**,** which require 64-bit index arithmetic for accesses to elements of arrays. Program units that use *Large Arrays* must be compiled using *–mcmodel=medium*. If *–Mlarge_arrays* is specified and *–mcmodel=medium* is not specified, the default small memory model is used, and all index arithmetic is performed in 64-bits. This can be a useful mode of execution for certain existing 64-bit applications that use the small memory model but allocate and manage a single contiguous data space larger than 2GB.

linux86 – 32-bit Linux operating system running on an *x86, AMD64* or *EM64T* processor-based system, with 32-bit GNU tools, utilities and libraries used by the PGI compilers to assemble and link for 32-bit execution.

linux86-64 – 64-bit Linux operating system running on an *AMD64* or *EM64T* processor-based system, with 64-bit and 32-bit GNU tools, utilities and libraries used by the PGI compilers to assemble and link for execution in either *linux86* or *linux86-64* environments. The 32-bit development tools and execution environment under *linux86-64* are considered a cross-development environment for *x86* processor-based applications.

Mac OS – collectively, all *osx86* and *osx86-64* platforms supported by the PGI compilers.

–mcmodel=small – compiler/linker switch to produce *small memory model* format objects/executables in which both code (*.text*) and data *(.bss*) sections are limited to less than 2GB. This switch is the default and only possible format for *linux86* 32-bit executables. This switch is the default format for *linux86-64* executables. Maximum address offset range is 32-bits, and total memory used for OS+Code+Data must be less than 2GB.

–mcmodel=medium – compiler/linker switch to produce *medium memory model* format objects/executables in which code sections are limited to less than 2GB, but data sections can be greater than 2GB. This option is supported *only* in *linux86-64* environments. It must be used to *compile* any program unit that will be linked in to a 64-bit executable that will use aggregate data sets larger than 2GB and will access data requiring address offsets greater than 2GB. This option must be used to *link* any 64-bit executable that will use aggregate data sets greater than 2GB in size. Executables linked using *–mcmodel=medium* can incorporate objects compiled using *–mcmodel=small* as long as the *small* objects are from a shared library.

MPI –MPI is a language-independent communications protocol used to program parallel computers.

MPICH – MPICH is a freely available, portable implementation of MPI, a standard for message-passing for distributed-memory applications used in parallel computing.

Network Installation – A term that applies to installing software on a shared file system available to many machines. Typically this type of installation allows multiple users to access and run the software – up to the number of licenses associated with the software.

NUMA – Non-Uniform Memory Access. A type of multi-processor system architecture in which the memory latency from a given processor to a given portion of memory can vary, resulting in the possibility for compiler or programming optimizations to ensure frequently accessed data is "close" to a given processor as determined by memory latency.

osx86 – 32-bit Apple Mac OS Operating Systems running on an *x86* Core 2 or Core 2 Duo processor-based system with the 32-bit Apple and GNU tools, utilities, and libraries used by the PGI compilers to assemble and link for 32-bit execution.

osx86-64 – 64-bit Apple Mac OS Operating Systems running on an *x64* Core 2 Duo processor-based system with the 64-bit and 32-bit Apple and GNU tools, utilities, and libraries used by the PGI compilers to assemble and link for either 64- or 32-bit execution.

SFU – Windows Services for Unix is the precursor to *SUA*. *SFU* supports 32-bit applications on *Windows 2000*, *Windows Server 2003*, and *XP*.

Shared library – a Linux library of the form *libxxx.so* containing objects that are dynamically linked into a program at the time of execution.

SSE – collectively, all SSE extensions supported by the PGI compilers.

SSE1 – 32-bit IEEE 754 FPU and associated *streaming SIMD extensions* (SSE) instructions on Pentium III, AthlonXP* and later 32-bit *x86*, *AMD64* and *EM64T* compatible CPUs, enabling scalar and packed vector arithmetic on single-precision floating-point data.

SSE2 – 64-bit IEEE 754 FPU and associated SSE instructions on P4/Xeon and later 32-bit *x86*, *AMD64* and *EM64T* compatible CPUs. SSE2 enables scalar and packed vector arithmetic on double-precision floating-point data.

SSE3 – additional 32-bit and 64-bit SSE instructions to enable more efficient support of arithmetic on complex floating-point data on 32-bit *x86*, *AMD64* and *EM64T* compatible CPUs with so-called *Prescott New Instructions* (PNI), such as Intel IA32 processors with EM64T extensions and newer generation (Revision E and beyond) AMD64 processors.

SSE4A and ABM – AMD Instruction Set enhancements for the Quad-Core AMD Opteron Processor. Support for these instructions is enabled by the `-tp barcelona` or `-tp barcelona-64` switch.

SSSE3 – an extension of the SSE3 instruction set found on the Intel Core 2.

Static linking – a method of linking:

- On Linux, use *–Bstatic* to ensure all objects are included in a generated executable at link time. Static linking causes objects from static library archives of the form libxxx.a to be linked in to your executable, rather than dynamically linking the corresponding libxxx.so shared library.
- On Windows, the Windows linker links statically or dynamically depending on whether the libraries on the link-line are DLL import libraries or static libraries. By default, the static PGI libraries are included on the link line. To link with DLL versions of the PGI libraries instead of static libraries, compile and link with the *–Bdynamic option*.

SUA – the Subsystem for Unix-based Applications is a source-compatible subsystem for compiling and running 32- and 64-bit UNIX-based applications on a computer running Windows. *SUA* is supported on *Windows 2003 R2* and *Vista*. *SUA* is bundled with Windows; however, the full *SUA Utilities SDK* must be installed to link programs.

Win32 – any of the 32-bit Microsoft* Windows* Operating Systems (XP/2000/Server 2003) running on an *x86, AMD64* or *EM64T* processor-based system. On these targets, the PGI compiler products include additional Microsoft tools and libraries needed to build executables for 32-bit Windows systems.

Win64 – any of the 64-bit Microsoft Windows Operating Systems (XP Professional / Windows Server 2003 x64 Editions) running on an *x64* processor-based system. On these targets, the PGI compiler products include additional Microsoft tools and libraries needed to build executables for either Win32 or Win64 environments.

Windows – collectively, all *Win32* and *Win64* platforms supported by the PGI compilers.

x64 – collectively, all AMD64 and EM64T processors supported by the PGI compilers.

x86 – a processor designed to be binary compatible with i386/i486 and previous generation processors from Intel* Corporation. Used to refer collectively to such processors up to and including 32-bit variants.

x87 – 80-bit IEEE stack-based floating-point unit (FPU) and associated instructions on *x86*–compatible CPUs.

2 PGI Release 7.1 Overview

This document describes changes between previous releases and Release 7.1 of the PGI compilers, as well as late-breaking information not included in the current printing of the *PGI User's Guide*. There are nine platforms supported by the *PGI Workstation* and *PGI Server* compilers and tools:

- *32-bit Linux* – supported on *32-bit Linux operating systems* running on either a 32-bit *x86* compatible or an *x64* compatible processor.
- *64-bit/32-bit Linux* – includes all features and capabilities of the 32-bit Linux version, and is also supported on *64-bit Linux operating systems* running an *x64* compatible processor.
- *32-bit Windows* – supported on *32-bit Windows operating systems* running on either a 32-bit *x86* compatible or an *x64* compatible processor.
- *64-bit/32-bit Windows* – includes all features and capabilities of the 32-bit Windows version, and is also supported on *64-bit Windows operating systems* running an *x64* compatible processor.
- *32-bit SFU* – supported on *32-bit Windows operating systems* running on either a 32-bit *x86* compatible or an *x64* compatible processor.
- *32-bit SUA* – supported on *32-bit Windows operating systems* running on either a 32-bit *x86* compatible or an *x64* compatible processor.
- *64-bit/32-bit SUA* – includes all features and capabilities of the 32-bit *SUA* version, and is also supported on *64-bit Windows operating systems* running an *x64* compatible processor.
- *32-bit Apple Mac OS X* – supported on 32-bit Apple Mac operating systems running on either a 32-bit or 64-bit Intel-based Mac system.

- *64-bit Apple Mac OS X* – supported on 64-bit Apple Mac operating systems running on a 64-bit Intel-based Mac system.

These release notes distinguish these versions where necessary.

2.1 PGI Release 7.1 Contents

Release 7.1 of PGI Workstation and PGI Server includes the following components:

- *PGF95* native OpenMP and auto-parallelizing Fortran 95 compiler.
- *PGF77* native OpenMP and auto-parallelizing FORTRAN 77 compiler.
- *PGHPF* data parallel High Performance Fortran compiler.
  Note: *PGHPF is not supported in Windows environments.*
- *PGCC* native OpenMP and auto-parallelizing ANSI C99 and K&R *C* compiler.
- *PGC++* native OpenMP and auto-parallelizing ANSI *C++* compiler.
- *PGPROF* multi-thread, OpenMP and MPI graphical profiler.
- *PGDBG* multi-thread, OpenMP and MPI graphical debugger.
  - *MPICH MPI libraries, version 1.2.7,* for both 32-bit and 64-bit development environments (Linux only)
  - Complete online documentation in PDF, HTML and UNIX `man` page formats.
  - A UNIX-like shell environment for *Win32* and *Win64* environments.

Depending on the product you purchased, you may not have licensed all of the above components.

2.2 Supported Processors

The following table contains the processors on which Release 7.1 of the PGI compilers and tools is supported. The *–tp <target>* command-line option generates executables that utilize features and optimizations specific to a given CPU and operating system environment. Compilers included in a 64-bit/32-bit PGI installation can produce executables targeted to any 64-bit or 32-bit target, including cross-targeting for AMD and Intel 64-bit AMD64 compatible CPUs.

In addition to the capability to generate binaries optimized for specific AMD or Intel processors, the PGI 7.1 compilers can produce PGI Unified Binary object or executable files containing code streams fully optimized and supported for both AMD and Intel x64 CPUs. The *–tp <target>* option must be used to produce unified binary files.

The table also includes the CPUs available and supported in multi-core versions.

Processors Supported by PGI 7.1

Brand	CPU	Cores	*<target>*	Memory Address	Floating Point HW					
					SSE1	SSE2	SSE3	SSSE3	SSE4	ABM and SSE4a
AMD	Opteron/Quadcore	4	barcelona-64	64-bit	Yes	Yes	Yes	No	No	Yes
AMD	Opteron/Quadcore	4	barcelona	32-bit	Yes	Yes	Yes	No	No	Yes
AMD	Opteron/Athlon64	2	k8-64	32-bit	Yes	Yes	Yes	No	No	No
AMD	Opteron/Athlon64	2	k8-32	32-bit	Yes	Yes	Yes	No	No	No
AMD	Opteron Rev E/F Turion /Athlon64	2	k8-64e	64-bit	Yes	Yes	Yes	No	No	No
AMD	Opteron Rev E/F	2	k8-32	32-bit	Yes	Yes	No	No	No	No
AMD	Turion64 Turion /Athlon64	1	k8-64e	64-bit	Yes	Yes	Yes	No	No	No
AMD	Turion64	1	k8-32	32-bit	Yes	Yes	No	No	No	No
Intel	Core 2	2	core2	32-bit	Yes	Yes	Yes	Yes	Yes	No
Intel	Core 2	2	core2-64	64-bit	Yes	Yes	Yes	Yes	Yes	No
Intel	P4/Xeon EM64T	2	p7-64	64-bit	Yes	Yes	Yes	Yes	No	No
Intel	P4/Xeon EM64T	2	p7	32-bit	Yes	Yes	Yes	Yes	No	No
Intel	Xeon/Pentium4	1	p7	32-bit	Yes	Yes	No	No	No	No
AMD	Athlon XP/MP	1	athlonxp	32-bit	Yes	No	No	No	No	No
Intel	Pentium III	1	piii	32-bit	Yes	No	No	No	No	No
AMD	Athlon	1	athlon	32-bit	No	No	No	No	No	No
AMD	K6	1	k6	32-bit	No	No	No	No	No	No
Intel	Pentium II	1	p6	32-bit	No	No	No	No	No	No
Generic	Generic x86	1	p5 or px	32-bit	No	No	No	No	No	No

2.3 Supported Operating Systems

The table lists the operating systems, and their equivalents, that Release 7.1 of the PGI compilers and tools supports.

To determine if Release 7.1 will install and run under a Linux equivalent version, such as Mandrake*, Debian*, Gentoo*, and so on, check the table for a supported system with the same glibc and gcc versions. Version differences in other operating system components can cause difficulties, but often these can be overcome with minor adjustments to the PGI software installation or operating system environment.

- Newer distributions of the Linux and Windows operating systems include support for x64 compatible processors and are designated 64-bit in the table. These are the only distributions on which the 64-bit versions of the PGI compilers and tools will fully install.
- If you attempt to install the 64-bit/32-bit Linux version on a system running a 32-bit Linux distribution, only the 32-bit PGI compilers and tools are installed.
- If you attempt to install the 64-bit Windows version on a system running 32-bit Windows, the installation fails.

Most newer Linux distributions support the *Native POSIX Threads Library* (NPTL), a new threads library that can be utilized in place of the *libpthreads* library available in earlier versions of Linux. Distributions that include NPTL are designated in the table. Parallel executables generated using the *OpenMP* and auto-parallelization features of the PGI compilers automatically make use of NPTL on distributions when it is available. In addition, the *PGDBG* debugger is capable of debugging executables built using either NPTL or earlier thread library implementations.

Multi-processor AMD Opteron processor-based servers use a *NUMA* (Non-Uniform Memory Access) architecture in which the memory latency from a given processor to a given portion of memory can vary. Newer Linux distributions, including SuSE 9/10 and SLES 9/10, include NUMA libraries that can be leveraged by a compiler and associated runtime libraries to optimize placement of data in memory.

In the table headings, HT = hyper-threading, NPTL = Native POSIX Threads Library, and NUMA = Non-Uniform Memory Access. For more information on these terms, see Terms and Definitions on page 2.

Operating Systems and Features Supported in PGI 7.1									
Distribution	**Type**	**64-bit**	**HT**	**pgC++**	**pgdbg**	**NPTL**	**NUMA**	**glibc**	**GCC**
RHEL 5.0	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.5	4.1.2
RHEL 4.0	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.4	3.4.3
Fedora 7	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.6	4.1.2
Fedora 6	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.5	4.1.1
Fedora 5	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.4	4.1.0
Fedora 4	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.5	4.0.0
Fedora 3	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.3	3.4.2
Fedora 2	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.3	3.3.3
SuSE 10.3	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.6.1	4.2.1
SuSE 10.2	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.5	4.1.0
SuSE 10.1	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.4	4.1.0
SuSE 10.0	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.3.5	4.0.2
SuSE 9.3	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.3.4	3.3.5
SuSE 9.2	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.3.3	3.3.4
SLES 10	***Linux***	Yes	Yes	Yes	Yes	Yes	Yes	2.4	4.1.0
SLES 9	***Linux***	Yes	Yes	Yes	Yes	No	Yes	2.3.3	3.3.3
SuSE 9.1	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.3	3.3.3
RHEL 3.0	***Linux***	Yes	Yes	Yes	Yes	Yes	No	2.3.2	3.2.3
SuSE 9.0	***Linux***	Yes	Yes	Yes	Yes	No	No	2.3.2	3.3.1
SuSE 8.2	***Linux***	Yes	Yes	Yes	Yes	No	No	2.3.2	3.3
Red Hat 9.0	***Linux***	No	No	Yes	Yes	Yes	No	2.3.2	3.2.2
Microsoft Windows	***XP***	No	Yes	Yes	Yes	NA	Yes	NA	NA
	2003	No	No	Yes	Yes	NA	Yes	NA	NA
	XP x64	Yes	Yes	Yes	Yes	NA	Yes	NA	NA
	2003 x64	Yes	Yes	Yes	Yes	NA	Yes	NA	NA
	SFU	No	Yes	Yes	Yes	NA	Yes	SFU	3.3
	SUA x86	No	Yes	Yes	Yes	NA	Yes	SUA	3.3
	SUA x64	Yes	Yes	Yes	Yes	NA	Yes	SUA	3.3
	Vista	Yes	Yes	Yes	Yes	NA	Yes	NA	NA
Apple Mac OS X	*Core 2*	No	No	Yes	Yes	NA	NA	NA	4.0.1
	Core 2 Duo	Yes	No	Yes	Yes	NA	NA	NA	4.0.1

Note. *www.pgroup.com/support/install.htm* lists any new Linux, Apple or Windows distributions that may be explicitly supported by the PGI compilers. If your operating system is newer than any of those listed in the preceding table, the installation may still be successful.

3 New or Modified Compiler Features

Following are the new features of Release 7.1 of the PGI compilers and tools as compared to prior releases.

- *Multiple PGI Unified Binary Targets* – The `-tp` switch now supports a comma-separated list of targets allowing programs to be optimized for more than two 64-bit targets. Unified Binary directives and pragmas support the latest processors including Barcelona and Core 2 Duo.
- *Performance improvements* – 5% to 10% performance improvements measured on several standard benchmarks, research community codes and ISV codes (see this link for recent benchmark testing: http://www.pgroup.com/benchmark.
- *OpenMP* – Unlimited OpenMP thread counts are available in all PGI configurations. The number of threads is now unlicensed in the OpenMP run-time libraries, the debugger, and the profiler up to the hard limit of 64 threads.
- *PGC++ low-overhead exceptions implementation* – A new implementation of exception handling defers the cost of exception handling until an exception is thrown. For programs with many exception regions and few throws, this option may lead to improved run-time performance. The new implementation is available on Windows and newer Linux systems with libgcc_eh.
- *Improved memory allocation on Windows* – A wrapper library for the Windows system memory allocation routines is available with the -Msmartalloc options. The wrapper library configures the small-block heap size to improve performance for programs that allocate lots of memory. The memory allocator also attempts to align large blocks of memory to avoid cache conflicts.

- *Use huge pages for memory allocation – –Msmartalloc* has been enhanced to support large TLBs on Linux and Windows. For programs that access a lot of memory, huge pages may reduce the number of TLB misses and improve performance. This option is most effective on Barcelona and Core 2 systems; older architectures may have less benefit due to fewer available TLB entries.
- *New Fortran intrinsics, subroutines, and attributes* – Implemented `GET_COMMAND_ARGUMENT`, `GET_ENVIRONMENT_VARIABLES`, `GET_COMMAND`, `LEADZ`, `POPCNT`, `POPPAR`, `SIZEOF`, `CTIME8` and `TIME8`. On Windows, added `GETDAT`, `GETTIM`, `TIME64`, `CTIME64`, and `LOCALTIME64` and support for the attribute, `DECORATE`.
- *Additional Fortran I/O features* – Added `ACCESS=APPEND` parameter and support for the `FLUSH` statement.
- *OpenMP enhancements* – Implemented C99/C++ style parallel *for* loops and added 64-bit `long`, `long long`, and all unsigned types as index variables.
- *Windows Environment* – Expanded coverage of Windows library routines for inter-procedural analysis (IPA). Support long command lines and linking programs with many arguments with response files (@filename) on the command line.
- *Enhanced gcc/g++ compatibility* – Support for the extension `__builtin_expect` and the attributes `aligned`, `weak`, and `alias`. Added support for variables in specified registers and controlling names used in assembler code.
- *Enhanced profile-feedback and code placement* optimizations – Profile-based feedback optimizations and code layout is available on 64- and 32-bit platforms for F95, F77, C, and C++. New optimizations based on profile feedback include call-site inlining, switch-statement expansion, and register allocation. Static code layout is enabled on all platforms.
- *Extended SSE vectorization* –Loops with indirect addressing are now vectorized and `FMAX`, `FMIN`, `DMAX` and `DMIN` reductions are recognized. More loops with single-to-double precision conversions are vectorized.
- *Reduced use of temporary variables* – Reduce the size of the stack frame by reusing temporary arrays and especially the associated descriptors when copying subprogram actual arguments. Eliminate copies when returning values from array-valued functions.
- *FLEXlm licensing upgrade* – FLEXlm version 11.4.1 is included on all platforms.

- *Expanded OS support* – Added support for Fedora Core 7 and SuSE 10.3 on Windows, for native compilation and execution under SFU and SUA using the Gnu *ld*.
- *Quad-Core AMD Opteron Processor support* –Updated tuning parameters and memory copy routines for Barcelona Rev B. Updated the opcode for Barcelona POPCNT from 'OF B8' (Rev A) to 'F3 OF B8' (Rev B) in the Windows assembler.
- *Apple Core 2 and Core 2 Duo support*– PGI Workstation for *Mac OS X* is available from http://www.pgroup.com.

3.1 Getting Started

By default, the PGI 7.1 compilers generate code that is optimized for the type of processor on which compilation is performed, the compilation host. If you are unfamiliar with the PGI compilers and tools, a good option to use by default is –fast or -fastsse.

3.2 Using –fast, –fastsse, and Other Performance-Enhancing Options

These options create a generally optimal set of flags for targets that support SSE/SSE2 capability. These options incorporate optimization options to enable use of vector streaming SIMD (SSE/SSE2) instructions for 64-bit targets. They enable vectorization with SEE instructions, cache alignment, and flushz.

Note. The contents of the *–fast* and *–fastsse* options are host-dependent.

- -fast and –fastsse typically include these options:

`-O2`	Specifies a code optimization level of 2.
`-Munroll=c:1`	Unrolls loops, executing multiple instances of the loop during each iteration.
`-Mnoframe`	Indicates to not generate code to set up a stack frame Note. With this option, a stack trace does not work.
`-Mlre`.	Indicates loop-carried redundancy elimination

- These additional options are also typically available when using *–fast* for 64-bit targets and *–fastsse* for both 32- and 64-bit targets:

`-Mvect=sse`	Generates SSE instructions
`-Mscalarsse`	Generates scalar SSE code with xmm registers; implies -Mflushz
`-Mcache_align`	Aligns long objects on cache-line boundaries. Note. On 32-bit systems, if one file is compiled with the –Mcache_align option, all files should be compiled with it. This is not true on 64-bit systems.
`-Mflushz`	Sets SSE to flush-to-zero mode
`-M[no]vect`	Controls automatic vector pipelining.

Note. For best performance on processors that support SSE instructions, use the *PGF95* compiler, even for FORTRAN 77 code, and the `-fastsse` option.

In addition to `-fast and -fastsse`, the `-Mipa=fast` option for inter-procedural analysis and optimization can improve performance. You may also be able to obtain further performance improvements by experimenting with the individual `-Mpgflag` options detailed in the *PGI User's Guide*, such as `-Mvect`, `-Munroll`, `-Minline`, `-Mconcur`, `-Mpfi/-Mpfo`, and so on. However, increased speeds using these options are typically application- and system-dependent. It is important to time your application carefully when using these options to ensure no performance degradations occur.

3.3 New or Modified Compiler Options

Unknown options are now treated as errors instead of warnings. This change makes it a compiler error to pass switches that are not known to the compiler; however, you can use the new switch *–noswitcherror* to issue warnings instead of errors for unknown switches.

The following compiler options have been added or modified in PGI 7.1:

- *–Bdynamic* – Compiles for and links to the DLL version of the PGI runtime libraries. For more information, refer to *–Bdynamic* on page 35.
- *–Bstatic* – Compiles for and links to the static version of the PGI runtime libraries. For more information, refer to *–Bstatic* on page 35.
- *–fast* – The *–fast* option adds *–Mautoinline* for C/C++ on 64-bit targets.

- *–Minfo=[no]file* – The *–Minfo* option adds a sub-option, *-[no]file,* which prints source file names as they are compiled. The default is *–Minfo=file.*
- *–noswitcherror* – Issue warnings instead of errors for unknown switches. This behavior can be configured in the `siterc` file with set `NOSWITCHERROR=1`
- *–Minline=[no]reshape* – Allow (disallow) inlining in Fortran even when array shapes do not match. The default is *–Minline=noreshape*, except with *–Mconcur* or *–mp*, when the default is *–Minline=reshape*.
- *–Mipa=[no]reshape* – Enable (disable) Fortran inline with mismatched array shapes.
- *–Mmpi=mpich1* – Use the default MPICH-1 communication libraries. Replaces the –Mmpi option, which is deprecated. Other MPI-related options are available with the PGI CDK.
- *–Mprof=mpich1* – Use the profiled MPICH-1 communication library. Replaces the –Mprof=mpi option, which is deprecated.
- *–M[no]smartalloc[=huge|huge:<n>|hugebss]* – *–Msmartalloc* has been enhanced to support large TLBs on Linux and Windows. This option must be used to compile the main routine to enable optimized *malloc* routines. The sub-option huge enables large 2 megabyte pages to be used by a single program. The effect is to reduce the number of TLB entries required to execute a program. This option is most effective on Barcelona and Core 2 systems; older architectures may have less benefit due to fewer available TLB entries.

  The supported sub-options are:

huge	Link in the huge page runtime library
huge:<n>	Limit the number of used pages to N
hugebss	Put the BSS section in huge pages

  By itself, the huge sub-option will try to allocate as many huge pages as required. The number of huge pages can be limited with the :n sub-option or the environment variable `PGI_HUGE_PAGES`. The *hugebss* attempts to put a program's uninitialized data section into huge pages.

- *–M[no]unroll[=c:<n>|n:<n>|m:<n>]* – Added the capability to unroll loops with multiple blocks, particularly those with conditional statements, can now be unrolled. A new option to *–Munroll*, "*m*", is available to control this feature.
  n:<n> Unroll single-block loops n times
  m:<n> Unroll multi-block loops n times
  The default setting is not to enable *–Munroll=m*. The default count *with –Munroll=m* is 4.
- *–O4* – The *–O4* option now enables enhancements to algebraic transformations and register allocation.
- *–Mvect[=[no]gather]* – The *gather* sub-option will vectorize loops containing indirect array references, such as:

```
sum = 0.d0
do k=d(j),d(j+1)-1
   sum = sum + a(k)*b(c(k))
enddo
```

  The default is *–Mvect=gather*.
- *–M[no]propcond* – Enable (disable) constant propagation from assertions derived from equality conditionals. Enabled by default.
- *– [no]traceback* – Adds debug information for runtime *traceback* for use with the environment variable `$PGI_TERM`. By default, *traceback* is enabled for f77 and f90 and disabled for C and C++. Setting `set TRACEBACK=OFF;` in a `siterc` or `.mypg*rc` will also disable default traceback. Using `ON` instead of `OFF` enabled default traceback.
- *–M[no]fpapprox[=div|sqrt|rsqrt]* – Perform certain single-precision fp operations using low-precision approximation.
  div Approximate floating point division
  sqrt Approximate floating point square root
  rsqrt Approximate floating point reciprocal square root
  By default *–Mfpapprox* is not used. If *–Mfpapprox* is used without sub-options, it defaults to use approximate *div*, *sqrt*, and *rsqrt*.
- *–M[no]fpmisalign* – Allow use of vector arithmetic instructions with memory operands whose addresses are not aligned on 16-byte boundaries for barcelona. On barcelona, an initialization routine to mask misalign fault on Barcelona processors guarded by runtime check of CPU ID. The default is *–Mnofpmisalign* on all processors, including barcelona. The option is effective if and only if it is used in conjunction with *–tp barcelona-64* or *–tp barcelona*, or the code is compiled on a barcelona processor. The compiled code can only be run on a barcelona processor.

- *–M[no]loop32* – Align innermost loops on 32-byte boundaries on barcelona. Small loops on barcelona may run fast if aligned on 32-byte boundaries; however, in practice, most assemblers do not yet implement efficient padding causing some programs to run more slowly with this option. Use *–Mloop32* on systems with an assembler tuned for barcelona. The default is *–Mnoloop32*.
- *–alias=[ansi|traditional]* – Select optimizations based on type-based pointer alias rules in C and C++.
  ansi Enable optimizations using ANSI C type-based pointer disambiguation
  traditional Disable type-based pointer disambiguation
  For C, the default is *–alias=ansi*. For C++, the default is *–alias=traditional*.
- *–Xs* – Use legacy standard mode for C and C++; now also implies *–alias=traditional*.
- *–Xt* – Use legacy transitional mode for C and C++; now also implies *–alias=traditional.*
- *—zc_eh* – Generate zero-overhead exception regions. The *—zc_eh* option defers the cost of exception handling until an exception is thrown, so for program with many exception regions and few throws, this option may lead to improved run-time performance. The default is not to use *—zc_eh* but instead to use *—sjlj_eh* which implements exception handling with *setjmp* and *longjmp*. This option is compatible with C++ code that was compiled with previous versions of PGI C++. The *—zc_eh* option is available only on newer Linux systems that supply the system unwind libraries in *libgcc_eh* and on Windows
- *–Xm* – This option has been removed. It instructed the C++ compiler to allow dollar signs in names. Now, like always, dollar signs are accepted.
- *–Bstatic* and *–Bdynamic* – On Windows, the *–Bstatic* and *–Bdynamic* options are new. Use *–Bstatic* to compile and link programs with static libraries. Use *–Bdynamic* to compile and link programs with DLLs. ***Note***: You must consistently use the same options to compile and link all of the files that will be linked into an executable. The default is *–Bstatic*.
- *–Mmakedll* – The *–Mmakedll* option now implies *–Mdynamic.*
- *–Mdll* – The *–Mdll* option has been removed. Use *–Bdynamic* instead.

- *–stack=nocheck* – The –stack option was changed to disable the automatic run-time stack extension feature on Windows. If the reserve and commit sub-options are set large enough to hold the full stack, no automatic extension is needed and the stack check can be avoided. The default is *–stack=check*. On Win64, there is no default `reserve` or `commit`. On Win32, the `reserve` and `commit` default sizes are each 2,097,152 bytes.
- *–Mchkstk* – Programs compiled with *–Mchkstk* are also instrumented to collect the stack high-water mark. If the environment variable `PGI_STACK_USAGE` is set at run time, the stack high-water mark will be printed at program exit.
- *–[no]compress_names* – Compress C++ mangled names to fit into 1024 characters. Highly nested template parameters can cause very long function names. These long names can cause problems for older assemblers. The current default is *–no_compress_names*. All C++ user code must be recompiled when using this switch. Libraries supplied by PGI work with *—compress_names*.
- *–V* – The *–V* option now prints the processor names. For example, on a Core 2 Duo, *–V* will print `-tp core2-64`.
- *–Mstandard* – *–Mstandard* now implies *–Mbackslash*, inhibiting recognizing the backslash escape sequences when -Mstandard is present, i.e., a backslash is treated as a normal character.

4 New or Modified PGDBG and PGPROF Features

PGI Workstation 7.1 includes several new features and enhancements in the *PGDBG* parallel debugger and the PGPROF parallel profiler. This chapter describes those features.

4.1 PGDBG New and Modified Features

- PGDBG 7.1 now supports debugging of applications consisting of up to four MPICH-1 processes running on the same machine as PGDBG.
- PGDBG 7.1 supports debugging of MSMPI applications running on Microsoft Windows CCS clusters.
- PGDBG 7.1 enhancements including:
  - Improved stack trace capability
  - Improved interoperability with Microsoft Visual C++
  - Improved interoperability with gcc/g++
  - Fast disassembly and overall performance improvements

For a description of the usage and capabilities of PGDBG, see the *PGI Tools Guide*. For limitations and workarounds, see *www.pgroup.com/support/faq.htm*.

4.2 PGPROF New and Modified Features

PGI Workstation 7.1 includes several new features and enhancements in the *PGPROF* parallel profiler.

- PGPROF 7.1 now supports MPI profiling of collective communication routines.
- PGPROF 7.1 now supports MSMPI profiling on Microsoft Windows CCS clusters.
- PGPROF 7.1 includes some graphical user interface improvements.

For a description of the usage and capabilities of PGPROF, see the *PGI Tools Guide*. For limitations and workarounds, see http://www.pgroup.com/support/faq.htm.

4.3 Running an MPICH Program on Linux

PGI Workstation 7.1 for *Linux* includes MPICH libraries, tools, and licenses required to compile, execute, profile, and debug MPI programs. *PGI Workstation* can be installed on a single node, and the node can be treated as if it is a cluster. The MPI profiler and debugger are limited to processes on a single node in *PGI Workstation*. The *PGI CDK Cluster Development Kit* supports general development on clusters.

The *PGI Tools Guide* describes the MPI enabled tool in detail:

- *PGPROF* graphical MPI/OpenMP/multi-thread performance profiler.
- PGDBG graphical MPI/OpenMP/multi-thread symbolic debugger
- MPICH MPI libraries, version 1.2.7, for both 32-bit and 64-bit development environments (Linux only)

Try the MPI "hello world" program in the `bench/mpihello` subdirectory:

```
% cp -r $PGI/linux86/7.1/examples/mpi ./mpihello
% cd ./mpihello/mpihello
% pgf77 -o mpihello mpihello.f -Mmpi
% mpirun mpihello
Hello world!  I'm node 0
% mpirun -np 4 mpihello
Hello world!  I'm node 0
Hello world!  I'm node 2
Hello world!  I'm node 1
Hello world!  I'm node 3
```

To run the debugger on an mpich program, do this:

```
mpirun -dbg=pgdbg -np 4 mpihello
```

4.4 Using the PGI Windows CDK with Microsoft Compute Cluster Server

If you have a PGI Windows CDK (Cluster Development Kit) license, then your PGI software includes support for working with Microsoft Compute Cluster Server and MSMPI. Specifically, this software includes support for these things:

- Building MPI applications with MSMPI
- Using PGPROF to do MPI profiling of MSMPI applications
- Debugging cluster applications on a Windows CCS cluster with PGDBG

This section provides information on these tasks.

4.4.1 Build MPI Applications with MSMPI

Note. For the options `-Mprof=msmpi and -Mmpi=msmpi` to work properly, the CCP_HOME environment variable must be set. This variable is typically set when the Microsoft Compute Cluster SDK is installed.

Using MSMPI libraries

To build an application using the MSMPI libraries, use the `-Mmpi=msmpi` option. This option inserts options into the compile and link lines to pick up the MSMPI headers and libraries.

Generate MPI Profile Data

To build an application that generates MPI profile data, use the `-Mprof=msmpi` option. This option performs MPICH-style profiling for Microsoft MSMPI. For Microsoft Compute Cluster Server only, using this option implies `-Mmpi=msmpi`.

The profile data generated by running an application built with the option `-Mprof=msmpi` contains information about the number of sends and receives, as well as the number of bytes sent and received, correlated with the source location associated with the sends and receives. `-Mprof=msmpi` must be used in conjunction with `-Mprof=func` or `-Mprof=lines`. When invoked using this type of profile data, PGPROF automatically displays MPI statistics.

4.4.2 Debug Cluster Applications with PGDBG

To invoke the PGDBG debugger to debug an MSMPI application, use the pgdbg `-mpi` option:

```
% pgdbg -mpi[:<path>] <mpiexec_args> [-program_args arg1,...argn]
```

The location of `mpiexec` should be part of your PATH environment variable. Otherwise, you should specify the pathname for `mpiexec`, or another similar launcher, as `<path>` in `-mpi[:<path>]`.

To start a distributed debugging session, you must use the `job submit` command on the command line, as illustrated in the Example that follows. You must also ensure that the debugger has access to the `pgserv.exe` remote debug agent on all nodes of the cluster used for the debug session.

To make `pgserv.exe` available, copy it from the PGI installation directory, such as `C:\Program Files\PGI\win64\7.1-5\bin\pgserv.exe`, into a directory or directories such that the path to `pgserv.exe` is the same on all nodes in the debug session.

Then you can start the debug session as follows:

```
% pgdbg -pgserv:<path_to_pgserv.exe> -mpi[:<job submit command>]
```

If you use a command similar to the following one, it assumes that a copy of pgserv.exe can be found in the current directory.

```
% pgdbg -pgserv -mpi[:<job submit command>]
```

Example

Suppose you wanted to debug the following job invocation:

```
"job.cmd" submit /numprocessors:4 /workdir:\\cce-
head\d\srt /stdout:sendrecv.out mpiexec sendrecv.exe
```

Use this command:

```
% pgdbg -pgserv "-mpi:c:\Program Files\Microsoft Compute
Cluster Pack\Bin\job.cmd" submit /numprocessors:4
/workdir:\\cce-head\d\srt /stdout:sendrecv.out mpiexec
sendrecv.exe
```

Important. For this command to execute properly, a copy of `pgserv.exe` must be located in `\\cce-head\d\srt`.

Since the CCP installation updates the default PATH, the following command is equivalent to the previous one:

```
% pgdbg -pgserv -mpi:job.cmd submit /numprocessors:4
/workdir:\\cce-head\d\srt /stdout:sendrecv.out mpiexec
sendrecv.exe
```

Note. The use of quotes around the `-mpi` option varies, depending on the type of shell you are using. In the example, or if you are using `cmd`, specify the option as `"-mpi:..."`, including the quotes around the option as well as around the optional job submit command. When invoking in a Cygwin bash shell, you can specify the `-mpi` option as `-mpi:"..."`, using the quotes around only the job submit command.

5 PGI Workstation 7.1

This chapter describes the updates and changes to PGI Workstation 7.1 that are specific to Linux, Windows, and Mac OS X, such as using the module load command on Linux.

5.1 PGI Workstation 7.1 for Linux

5.1.1 Using Environment Modules

On Linux, if you use the Environment Modules package (e.g., the `module load` command), PGI 7.1 includes a script to set up the appropriate module files.

Assuming your installation base directory is `/opt/pgi`, and your `MODULEPATH` environment variable is `/usr/local/Modules/modulefiles`, execute this command:

```
/opt/pgi/linux86/7.1-5/etc/modulefiles/pgi.module.install \
       -all -install /usr/local/Modules/modulefiles
```

This command creates module files for all installed versions of the PGI compilers. You must have write permission to the `modulefiles` directory to enable the module commands:

```
module load pgi32/7.1
module load pgi64/7.1
module load pgi/7.1
```

where "pgi/7.1" uses the 32-bit compilers on a 32-bit system and uses 64-bit compilers on a 64-bit system.

To see what versions are available, use this command:

```
module avail pgi
```

The `module load` command sets or modifies the environment variables as follows:

This environment variable…	Is set or modified by the module load command to be…
PGI	the base installation directory
CC	full path to pgcc
FC	full path to pgf90
F90	full path to pgf90
F77	full path to pgf77
CPP	full path to pgCC
CXX	path to pgCC
C++	path to pgCC
PATH	prepends the PGI compiler and tools bin directory
MANPATH	prepends the PGI man page directory
LD_LIBRARY_PATH	prepends the PGI library directory

PGI does not support the Environment Modules package. For more information about the package, go to: *http://modules.sourceforge.net*.

5.2 PGI Workstation 7.1 for Windows, SFU, and SUA

PGI Workstation 7.1 for *Windows* supports most of the features of the 32- and 64-bit versions for *linux86* and *linux86-64* environments. The product optionally provides a familiar and somewhat compatible development environment for Linux or RISC/UNIX users porting to or developing programs for Windows systems. Except where noted in the *PGI User's Guide* and these release notes, the PGI compilers and tools on *Windows* function identically to their *Linux* counterparts.

PGI Workstation includes the *Microsoft Open Tools* tools, libraries, and include files. *Open Tools* includes C header files that use C++ style comments. Also, some Microsoft header files generate warnings about things such as multiple definitions of types. In most cases these warnings may be safely ignored.

5.2.1 The Windows Command Environment

A UNIX-like shell environment, *Cygwin*, is bundled with *PGI Workstation 7.1* for *Windows* to provide a familiar development environment for Linux or UNIX users. *PGI Workstation* for SFU and SUA does not include Cygwin; it runs in the SFU/SUA shell environment.

After installation, a double-left-click on the *PGI Workstation* icon on your desktop will launch a Cygwin `bash` shell command window with pre-initialized environment settings. Many familiar UNIX commands are available, such as `vi`, `sed`, `grep`, `awk`, `make`, and so on. If you are unfamiliar with the `bash` shell, refer to the user's guide included with the online HTML documentation.

On *Win64*, the desktop icon starts a `bash` shell configured for building 64-bit programs. To start a `bash` shell configured for building 32-bit programs, launch *PGI Workstation (32-bit)* from the Start menu.

Alternatively, by selecting the appropriate option from the *PGI Workstation* program group accessed in the usual way through the "Start" menu, you can launch a standard Windows command window that is pre-initialized to enable use of the PGI compilers and tools.

The command window launched by *PGI Workstation* can be customized using the "Properties" selection on the menu accessible by right-clicking the window's title bar.

5.2.2 MKS Toolkit Compatibility

The MKS Toolkit is a commercially available product providing a suite of UNIX and Windows utilities and is available for Win32 and Win64. You can use *PGI Workstation* compilers and tools in any of the MKS toolkit shells. To use *PGI Workstation* in an MKS shell, you must first configure your environment.

In the following example, the Windows system drive is `C:` and default installations were selected for the Java JRE and *PGI Workstation*. Open a Windows command prompt and execute the following commands:

```
> set PGI=C:\Program Files\PGI
> PATH=C:\Program Files (x86)\Java\j2re1.5.0_05/bin;%PATH%
> PATH=%PGI%\win64\7.1-5\bin;%PATH%
> set TMPDIR=C:\temp
```

Invoke an MKS shell. For example, to start the bash shell, type:

```
> bash.exe
```

For more information or to obtain the MKS Toolkit, visit the MKS website at *http://www.mkssoftware.com/*.

5.2.3 Using Shared object files in SFU and SUA

PGI Workstation for 32-bit SFU and 32-bit SUA now uses the GNU ld for its linker, unlike previous versions that used the Windows LINK.EXE. With this change, the PGI compilers and tools are now able to generate shared object (.so) files. You use the -shared switch to generate a shared object file.

The following example creates a shared object file, “hello.so”, and then creates a program called “hello” that uses it.

Example 1:

First create a shared object file called "hello.so"

```
pgcc -shared hello.c -o hello.so
```

Then create a program that uses the shared object, in this example, "hello.so":

```
pgcc hi.c hello.so -o hello
```

When running a program that uses a shared object, you may encounter an error message similar to the following:

```
hello: error in loading shared libraries
hello.so: cannot open shared object file: No such file
or directory
```

This error message either means that the shared object file does not exist or its location is not specified in your `LD_LIBRARY_PATH` variable. To specify the location in your `LD_LIBRARY_PATH` variable, add the shared object’s directory to your variable.

Example 2:

The following example adds the current directory to your `LD_LIBRARY_PATH` variable under C Shell, enter:

```
setenv LD_LIBRARY_PATH "$LD_LIBRARY_PATH":"./"
```

The following list is a summary of compiler switches that support shared objects:

-shared	Used to produce shared libraries
-Bdynamic	Passed to linker; specify dynamic binding
-Bstatic	Passed to linker; specify static binding
-Bstatic_pgi	Use to link static PGI libraries with dynamic system libraries; implies -Mnorpath
-L<libdir>	Passed to linker; Add directory to library search path
-Mnorpath	Don't add -rpath paths to link line
-Mnostartup	Do not use standard linker startup file
-Mnostdlib	Do not use standard linker libraries
-R<ldarg>	Passed to linker; just link symbols from object, or add directory to run time search path

5.3 PGI Workstation 7.1 for Mac OS X

PGI Workstation 7.1 for *Mac OS X* supports most of the features of the 32- and 64-bit versions for *linux86* and *linux86-64* environments. Except where noted in these release notes or the reference manuals, the PGI compilers and tools on *Mac OS X* function identically to their *Linux* counterparts.

5.3.1 Mac OS X Debugging

Note. Due to technical issues, the PGDBG Debugger is not supported in this release for Mac OS X. PGI is working with Apple to resolve these issues, and plans to include PGDBG in a future build as soon as possible.

6 Distribution and Deployment

This chapter contains a number of topics that are related to using the compilers, including optimizing through the use of unified binaries, using the linking options on Windows, and customizing with siterc and user rc files.

6.1 Generating PGI Unified Binaries

All PGI compilers can produce PGI Unified Binary programs containing code streams fully optimized and supported for both AMD64 and Intel EM64T processors using the *-tp* target option. The compilers generate and combine into one executable multiple binary code streams each optimized for a specific platform. At runtime, this one executable senses the environment and dynamically selects the appropriate code stream.

Different processors have differences, some subtle, in hardware features such as instruction sets and cache size. The compilers make architecture-specific decisions about such things as instruction selection, instruction scheduling, and vectorization. PGI unified binaries provide a low-overhead means for a single program to run well on a number of hardware platforms.

Executable size is automatically controlled via unified binary culling. Only those functions and subroutines where the target affects the generated code will have unique binary images, resulting in a code-size savings of 10-90% compared to generating full copies of code for each target.

Programs can use PGI Unified Binary even if all of the object files and libraries are not compiled as unified binaries. Like any other object files, you can use PGI Unified Binary object files to create programs or libraries. No special start up code is needed; support is linked in from the PGI libraries.

The *-Mpfi* option disables generation of PGI Unified Binary. Instead, the default target auto-detect rules for the host are used to select the target processor.

6.1.1 Unified Binary Command-line Switches

The PGI Unified Binary command-line switch is an extension of the target processor switch, *-tp*, which may be applied to individual files during compilation.

The target processor switch, *-tp*, accepts a comma-separated list of 64-bit targets and generates code optimized for each listed target.

The following example generates optimized code for three targets.

```
-tp k8-64,p7-64,core2-64
```

A special target switch, *-tp x64*, is the same as *-tp k8-64,p7-64*.

6.1.2 Unified Binary Directives and Pragmas

Unified binary directives and pragmas may be applied to functions, subroutines, or whole files. The directives and pragmas cause the compiler to generate PGI Unified Binary code optimized for one or more targets. No special command line options are needed for these pragmas and directives to take effect.

The syntax of the Fortran directive is

```
!pgi$[g|r| ] pgi tp [target]...
```

where the scope is g (global), r (routine) or blank. The default is r, routine.

The following example indicates that the whole file, represented by *g*, should be optimized for both *k8_64* and *p7_64*.

```
!pgi$g pgi tp k8_64 p7_64
```

The syntax of the C/C++ pragma is

```
#pragma [global|routine|] tp [target]...
```

where the scope is global, routine, or blank. The default is routine.

For example, the following syntax indicates that the next function should be optimized for *k8_64*, *p7_64*, and *core2_64*.

```
#pragma routine tp k8_64 p7_64 core2_64
```

6.2 Static and Dynamic Linking on Windows

Prior to the 7.1 release, on Windows, all executables were linked against the PGI runtime DLL called pg.dll and the multi-threaded DLL version of the Microsoft runtime libraries. All executables were therefore dependent upon pg.dll and the Microsoft runtime. PGI now provides two compiler options that allow you to select static or dynamic linking.

6.2.1 –Bdynamic

Use this option to compile for and link to the DLL version of the PGI runtime libraries. This flag is required when linking with any DLL built by the PGI compilers. This flag corresponds to the /MD flag used by Microsoft's cl compilers.

On Windows, *–Bdynamic* must be used for *both* compiling and linking.

When you use the PGI compiler flag *–Bdynamic* to create an executable that links to the DLL form of the runtime, the executable built is smaller than one built without *–Bdynamic*. The PGI runtime DLLs, however, must be available on the system where the executable is run. The *–Bdynamic* flag must be used when an executable is linked against a DLL built by the PGI compilers.

6.2.2 –Bstatic

Use this option to explicitly compile for and link to the static version of the PGI runtime libraries.

On Windows, *–Bstatic* must be used for *both* compiling and linking.

For more information and usage examples for *–Bdynamic* and *–Bstatic* as well as information on using and creating static and dynamically-linked libraries, refer to the *PGI User's Guide*.

6.3 The REDIST Directories

Programs built with PGI compiler may depend on run-time library files. These library files must be distributed with such programs to enable them to execute on systems where the PGI compilers are not installed. There are PGI redistributable files for all platforms. On Windows, PGI also supplies Microsoft redistributable files.

6.3.1 PGI Redistributables

The PGI 7.1 release includes these directories:

- *$PGI/linux86/7.1/REDIST*
- *$PGI/linux86-64/7.1/REDIST*
- *$PGI/win64/7.1-5/REDIST*
- *$PGI/win32/7.1/REDIST*

These directories contain all of the PGI Linux runtime library shared object files or Windows dynamically linked libraries that can be re-distributed by PGI 7.1 licensees under the terms of the PGI End-user License Agreement (EULA). For reference, a text-form copy of the PGI EULA is included in the 7.1 directory.

The Linux REDIST directories contain the PGI runtime library shared objects for all supported targets. This enables users of the PGI compilers to create packages of executables and PGI runtime libraries that will execute successfully on almost any PGI-supported target system, subject to these requirements:

- End-users of the executable have properly initialized their environment
- On Linux, users have set LD_LIBRARY_PATH to use the relevant version of the PGI shared objects.

6.3.2 Microsoft Redistributables

The PGI products on Windows include Microsoft Open Tools. The Microsoft Open Tools directory contains a subdirectory named “redist”. PGI 7.1 licensees may redistribute the files contained in this directory in accordance with the terms of the PGI End-User License Agreement.

6.4 Customizing With siterc and User rc Files

The PGI 7.1 release for Linux platforms includes a `siterc` file in the `bin` directory to enable site-specific customization of the PGI compiler drivers. Using `siterc`, you can control how the compiler drivers invoke the various components in the compilation tool chain. In addition to the `siterc` file, user *rc* files can reside in a given user's home directory: `.mypgf77rc`, `.mypgf90rc`, `.mypgccrc`, `.mypgcpprc`, and `.mypghpfrc` can be used to control the respective PGI compilers. All of these files are optional.

The following table contains examples that show how these *rc* files can be used to tailor a given installation for a particular purpose.

To perform this task….	**Do this:**
Make the libraries found in */opt/newlibs/64* available to all *linux86-64* compilations	Add the line: `set SITELIB=/opt/newlibs/64;` to */opt/pgi/linux86-64/7.1/bin/siterc*
Make the libraries found in */opt/newlibs/32* available to all *linux86* compilations.	Add the line: `set SITELIB=/opt/newlibs/32;` to */opt/pgi/linux86/7.1/bin/siterc*
Add a new library path `/opt/local/fast` to *all linux86-64 compilations.*	Add the line: `append SITELIB=/opt/local/fast;` to */opt/pgi/linux86-64/7.1/bin/siterc*
Make the include path `/opt/acml/include` available to all compilations; *–I/opt/acml/include*.	Add the line: `set SITEINC=/opt/acml/include;` to */opt/pgi/linux86/7.1/bin/siterc* and to */opt/pgi/linux86-64/7.1/bin/siterc*

To perform this task….	Do this:

Change *–Mmpi* to link in */opt/mympi/64/libmpix.a* with *linux86-64* compilations.	Add the line:
	`set MPILIBDIR=/opt/mympi/64;`
	`set MPILIBNAME=mpix;`
	to */opt/pgi/linux86-64/7.1/bin/siterc*

Have *linux86-64* compilations always add *–DIS64BIT –DAMD*	Add the line: `set SITEDEF=IS64BIT AMD;` to */opt/pgi/linux86-64/7.1/bin/siterc*
A user builds an F90 executable for *linux86-64* or *linux86* that resolves PGI shared objects in the relative directory *./REDIST*	Add the line: `set RPATH=./REDIST;` to *~/.mypgf95rc*. *NOTE:* this will only affect the behavior of PGF95 for the given user.

7 Known Limitations and Corrections

This chapter contains information about known limitations as well as the corrections that have occurred to PGI Workstation 7.1.

7.1 Known Limitations

The frequently asked questions (FAQ) section of the *pgroup.com* web page at *http://www.pgroup.com/support/index.htm* provides more up-to-date information about the state of the current release.

- When debugging a MPI job that is launched under pgserv, the processes in the job are stopped before the first instruction of the program. Since there is no source level debugging information at this point, issuing the source level next command executes very slowly. To avoid having to run the job until it completes, stops due to an exception, or stops by a PGDBG halt command entered by the user, the user should set an initial breakpoint. If a Fortran program is being debugged, set the initial breakpoint at main, or MAIN_, or at another point on the execution path before issuing the continue command.
- Object and module files created using *PGI Workstation 7.1* compilers are incompatible with object files from *PGI Workstation 5.x* and prior releases.
- Object files compiled with *–Mipa* using *PGI Workstation 6.1* and prior releases must be recompiled with *PGI Workstation 7.1*.

- On Windows, the version of *vi* included in Cygwin can have problems when the SHELL variable is defined to something it does not expect. In this case, the following messages appear when vi is invoked:

  ```
  E79: Cannot expand wildcards
  Hit ENTER or type command to continue
  ```

  To workaround this problem, set SHELL to refer to a shell in the cygwin bin directory, e.g. /bin/bash.
- The *–i8* option can make programs incompatible with MPI, use of any INTEGER*8 array size argument can cause failures with these libraries.
- The –i8 option can make programs incompatible with the bundled ACML library. Visit *developer.amd.com* to check for compatible libraries.
- On Apple Mac OS platform, the *PGI Workstation 7.1* compilers do not support static linking of user binaries. For compatibility with future Apple updates, the compilers support dynamic linking of user binaries.
- Programs that incorporate object files compiled using *–mcmodel=medium* cannot be statically linked. This is a limitation of the *linux86-64* environment, not a limitation specific to the PGI compilers and tools.
- Using *–Mipa=vestigial* in combination with *–Mipa=libopt* with *PGCC*, you may encounter unresolved references at link time. This problem is due to the erroneous removal of functions by the *vestigial* sub-option to *–Mipa*. You can work around this problem by listing specific sub-options to *–Mipa*, not including *vestigial.*
- Using *–Mprof=func*, *–mcmodel=medium* and *–mp* together on any of the PGI compilers can result in segmentation faults by the generated executable. These options should not be used together.
- Programs compiled and linked for *gprof*-style performance profiling using *–pg* can result in segmentation faults on system running version 2.6.4 Linux kernels.
- *OpenMP* programs compiled using *–mp* and run on multiple processors of a *SuSE 9.0* system can run very slowly. These same executables deliver the expected performance and speed-up on similar hardware running *SuSE 9.1* and above.

- ACML 3.6 is built using the *–fastsse* compile/link option, which includes *–Mcache_align*. When linking with ACML using the *–lacml* option on 32-bit targets, all program units must be compiled with *–Mcache_align*, or an aggregate option such as *–fastsse* which incorporates *–Mcache_align*. This process is not an issue on 64-bit targets where the stack is 16-byte aligned by default. The lower-performance, but fully portable, *libblas.a* and *liblapack.a* libraries can be used on CPUs that do not support SSE instructions.
- When compiling with *–fPIC* and linking with *–lacml*, you may get the message "error while loading shared libraries: libacml_mv.so: cannot open shared object file: No such file or directory." In this case, you must add *-lacml_mv* library to the link line.
- Using *-Mpfi* and *-mp* together is not supported. The *-Mpfi* flag will disable *-mp* at compile time, which can cause run-time errors in programs that depend on interpretation of OpenMP directives or pragmas. Programs that do not depend on OpenMP processing for correctness can still use profile feedback. The *-Mpfo* flag does not disable OpenMP processing.
- On Windows, runtime libraries built for debugging (e.g. msvcrtd and libcmtd) are not included with *PGI Workstation*. When a program is linked with *–g*, for debugging, the standard non-debug versions of both the PGI runtime libraries and the Microsoft runtime libraries are always used. This limitation does not affect debugging of application code.
- Dynamic Link Libraries (DLLs) built on the Microsoft Windows platform by the *PGI Workstation 7.1* compilers have the following known limitations:
  - DLLs cannot be produced with the *PGI Workstation* C++ compiler.
  - If a DLL is built with the *PGI Workstation* compilers, the runtime DLLs must be used. The compiler option *–Mmakedll* ensures the correct runtime libraries are used.
  - If an executable is linked with any *PGI Workstation*-compiled DLL, the *PGI Workstation* runtime library DLLs must be used; this means the static libraries, which are used by default, cannot be used. To accomplish this, use the compiler option *–Bdynamic* when creating the executable.
- *PGPROF* – Do not use *–Mprof* with *PGI Workstation* runtime library DLLs. To build an executable for profiling, use the static libraries. When the compiler option *–Bdynamic* is *not* used, the static libraries are the default.

- *PGPROF* – Times reported for multi-threaded sample-based profiles, that is, profiling invoked with *–pg* or *–Mprof=time* options, are for the master thread only. PGI-style instrumentation profiling with *–Mprof={lines | func}* or hardware counter-based profiling using *–Mprof=hwcts* must be used to obtain profile data on individual threads.
- *PGDBG* – The `watch` family of commands is unreliable when used with local variables. Calling a function or subroutine from within the scope of the `watch`ed local variable may cause missed events and/or false positive events. Local variables may be `watch`ed reliably if program scope does not leave the scope of the `watch`ed variable. Using the `watch` family of commands with global or static variables is reliable.
- *PGDBG* – The `call` command does not support the following F90/F95 features: array-valued functions, pointer-valued functions, assumed-shape array arguments, or pointer arguments. There is no known workaround to this limitation.
- *PGDBG* – Before PGDBG can set a breakpoint in code contained in a shared library, .so or .dll, the shared library must be loaded.
- *PGDBG* – Debugging of unified binaries, that is, programs built with the *-tp=x64* option, is not fully supported. The names of some subprograms are modified in the creation of the unified binary, and PGDBG does not translate these names back to the names used in the application source code. For detailed information on how to debug a unified binary, see http://www.pgroup.com/support/tools.htm.
- *PGDBG Windows* – In *PGDBG* on the *Windows* platform, use the forward slash ('/') character to delimit directory names in file path names.
  Note. This requirement does not apply to the PGDBG `debug` command or to target executable names on the command line, although this convention will work with those commands.
- *PGDBG SFU/SUA* – Due operating system limitations, PGDBG does not support hardware watchpoints, that is, the "hwatch" command, on SFU/SUA systems.
- *PGDBG SFU/SUA* – Due to operating system limitations, PGDBG supports multi-thread debugging only with 32-bit SUA programs, with one restriction: once stopped, the process may be continued by continuing all threads, or a single thread, but not a partial set of the threads. Attempts to continue a partial set of threads results in the entire process, all threads, being continued.

7.2 Corrections

The following problems have been corrected in the *PGI Workstation 7.1* release. Most were reported in *PGI Workstation 7.0* or previous releases. Problems found in *PGI Workstation 7.0* may not have occurred in previous releases. A table is provided that describes the summary description of the problem. An *Internal Compiler Error* (ICE) is usually the result of checks the compiler components make on internal data structures, discovering inconsistencies that could lead to faulty code generation. For a complete and up-to-date list of TPRs fixed in recent releases of the PGI compilers and tools, see www.pgroup.com/support/release_tprs.htm.

7.2.1 Corrections in 7.1-5

The following TPRs have been resolved in 7.1-5.

TPR	Language / Tool	Description
3662	64-bit Fortran	MPI program fails to complete. Added a new routine, fsync 3F, to help mitigate race conditions in user code that arise because open, seek, write, and close calls may happen in the system kernel buffers but not necessarily on the disk.
4159	32/64-bit Fortran	Use of Module causes 'Severe error - Intrinsic not supported in initialization: mod PGF90 now implements the MOD intrinsic for integer initializations.
4322	64-bit Fortran	pgCC generates vector sse code which produces wrong answers. The optimizer may not have computed the proper alias information when two different pointers access the same class member and that class member is a pointer or array reference.
4352	32/64-bit C/C++	#PRAGMA OMP PARALLEL NUM_THREADS(1) -- ERROR MSG - EXPECTED AN IDENTIFIER PGC++ now allows expressions in an OpenMP directive that uses num_threads.

TPR	Language / Tool	Description
4353	32/64-bit Fortran	pgf90 program fails to compile ICE - "mk_mem_ptr_shape: no static desc for pointer" Fixed a problem when processing a deallocate statement for an allocatable member of a derived type.
4367	32/64-bit C/C++	pgCC with OpenMP ICE – sym_is_refd: bad sptr Fixed a problem with non-integer reduction variables.
14204	32-bit Fortran	select case(trim(adjustl(ftype))) segfaults on 7.1-2, not 7.0-7 Using trim in a case expression no longer causes a seg fault.
14205	32/64-bit Fortran	OpenMP regression 'FAIL: private copies are not disassociated' Corrected an error that caused private copies of threadprivate pointer variables not to be disassociated.
14208	32/64-bit Fortran	pgf90 example exposes an extra deallocation. Corrected an error where some internal variables were not created private in a parallel region.
14210	32/64-bit Fortran	pgf90 example causes ICE 'can't get const value 0' Corrected an error where the MIN intrinsic in an initialization expression might cause an internal compiler error.
14264	32/64-bit Fortran	pgf90 -fast ICE 'mismatched carry-around expression' The optimizer now handles the case when a negated subexpression is redundant with an existing redundant subexpression.
14279	64-bit Fortran	pgf95 -Mallocatable=03 gives wrong answer with -Mbounds and -g flags Corrected the bounds information for dummy arguments when used with -Mbounds and -g.
14285	32-bit C/C++	C++ Generates Bad Assembly code, identifer too long Fixed a problem generating assembly code for long identifiers.
14286	64-bit C/C++	C++ OpenMP fails where C OpenMP works Removed an outdated restriction in PGC++ that prevented sections inside of barrier.

TPR	Language / Tool	Description
14295	32/64-bit C/C++	Internal Compiler Error peel_last_iteration when using '-Mvect=sse' A compiler saftey check was incorrectly identifying an otherwise legal case as an internal error.
14313	32/64-bit C/C++	Can't inline intrinsics PGC++ failed to inline the call to the intrinsic mm_add_ps. The definition of mm_add_ps and other intrinsics have been updated to match the strict type-matching rules of the C++ inliner.
14319	64-bit Fortran	-O2 64-bit failed on WPS2.2.1 on core-2 in 7.1 The optimizer no longer considers loads of dummy arguments as invariant if the dummy argument is optional.
14329	32/64-bit C/C++	Boost C++ build gets "no instance of overloaded operator new Added new overloadings of new and delete to the PGI standard library: extern void *operator new(size_t sz, std::locale * ll); extern void operator delete(void *, std::locale * ll);
14331	32/64-bit Fortran	PGF90 example segfaults at 'SELECT CASE(TRIM(ADJUST(pt)))=ac' Duplicate of TPR#14202, also fixed in PGI 7.1-5.
14336	64-bit Fortran	_PGIC_version macros broken on 64-bit Apple OSX Updated the version macros for PGI 7.1.
14348	32/64-bit Fortran	Code that compiles with ifort fails with pgf90 Corrected a problem when passing an array to an elemental intrinsic in an initialization expression.
14351	ALL	Add Installation Guide to release The Installation Guide is now a separate document. The Release Notes no longer include installation instructions. Both the Installation Guide and the Release Notes may be found in the "doc" subdirectory of a release and on http://www.pgroup.com.
14362	32/64-bit Fortran	'pgf90 -Mipa=inline,except:xxx' fails to NOT inline xxx The command-line option expected the so-called decoarated function name. Now, both the user-visible and the decoarated name can be used.

TPR	Language / Tool	Description
14372	32-bit Fortran	Mac OS X 7.1-4 - mistake in startup script for FLEXlm lmgrd The startup script inadvertently referred to the PGI 7.0 release of lmgrd.
14373	64-bit C/C++	pgcc fails to properly link header file alloca.h on 64-bit Mac OS X An updated version of alloca.h is now in the include directory.

7.2.2 Corrections in 7.1-4

The following TPRs have been resolved in 7.1-4.

TPR	Language / Tool	Description
4217	32/64-bit Fortran	Read from character string succeeds, expected failure. Using formatted read of a double from the string "1.x" now properly returns an error. Previously, the value "1." was read as a valid string.
13258	32/64-bit Fortran	Derived types with allocatables print incorrectly. When printing derived types with allocatable members, the implicit data structures for the allocatables were printed.
14169	All	Complex multiply not vectorized properly. Multiple complex assignments not vectorized properly.
14172	All	PGI 7.1 man pages are out-of-date. Several options, such as -zc_eh, -sjlj_eh, and -traceback, were not documented in the PGI 7.1 man pages.
14220	32/64-bit Fortran	Threadprivate character improperly shared. OpenMP THREADPRIVATE CHARACTER variables were being shared by multiple threads.
14238	C/C++	Regression in tail-recursion elimination. Functions that pass the address of a local variable are not candidates for tail-recursion elimination.

TPR	Language / Tool	Description
14242	32/64-bit Fortran	Automatic variables managed with heavyweight allocator. Allocations of automatic arrays were done using the standard Fortran 90 allocator. This allocator has overhead to maintain F90 semantics of allocate and deallocate. This extra processing isn't needed for automatics and a lightweight allocator can be used.
14252	32/64-bit Fortran	Add Quad-core Opteron 'fastmath' library. When compiling with '-tp barcelona-64' or natively on a Quad-core AMD Opteron, the compiler generates calls to math library routines optimized for the target.

7.2.3 Corrections in 7.1-3

The following TPRs have been resolved in 7.1-3.

TPR	Language / Tool	Description
3875	32/64-bit Fortran	Code with ambiguous error not flagged as error
3894	32/64-bit Fortran	Program reports errors at the wrong line number
3905	32/64-bit Fortran	Example generates severe error 'Illegal attempt to redefine symbol'
4090	32/64-bit Fortran	Inappropriate error 'Illegal use of symbol max - not public entity of module'
4184	32/64-bit Fortran	Illegal FORALL loop results in ICEs - better to have error message
4234	32/64-bit Fortran	Example causes false severe error 'Illegal implied DO expression'
4291	32/64-bit Fortran	pgf90 does not catch argument mismatches in CONTAINED code.
4305	32/64-bit Fortran	Legal program generates severe 'must be a deferred shape array' error
4326	32/64-bit C	Undefined reference when -O2 and -g used together
4327	32/64-bit Fortran	Return value from a C complex function not handled correctly
4330	32/64-bit Fortran	Internal error in template code
4350	32/64-bit Fortran	Performance regression when sequential data in pointers passed to F77 subroutines

TPR	Language / Tool	Description
4357	32/64-bit C/C++	PGCC-S-0155-Illegal context for barrier message should not be issued
4358	32/64-bit C/C++	‘#pragma omp barrier’ before '#pragma omp for ordered' causes PGCC-S-0155-Illegal context

7.2.4 Corrections in 7.1-2

The following TPRs have been resolved in 7.1-2.

TPR	Language / Tool	Description
4096	All	Want to be able to link to static or dynamic libraries on Windows
4123	32/64-bit Fortran	OpenMP error ' Statement not allowed in WORKSHARE construct'
4186	32/64-bit Fortran	Problem setting array bounds from intrinsic with constant strings
4196	32/64-bit Fortran	Parameters incorrectly initialized
4256	32/64-bit Fortran	Problem when array constructor depends on an outer loop variable
4315	64-bit C/C++	Boost needs access to GLIBC trunc and strptime
4317	32/64-bit Fortran	Compilation warnings emitted for valid code (PGF90-W-0164-Overlapping data initializations)
4331	32/64-bit Fortran	Array pointer assignment fails within OpenMP WORKSHARE
4332	32/64-bit Fortran	Mac OS X Block Data in an archive is not found at link time
4336	32/64-bit C/C++	Extended asm 'register' variables may lose stores

7.2.5 Corrections in 7.1-1

The following TPRs have been resolved in 7.1-1.

TPR	Language / Tool	Description
3195	32/64-bit Fortran	Would like function than translates Fortran 90 iostat error codes to error strings
3425	64-bit Fortran	Request for directives to ignore possible dependences in array assignments
3448	32/64-bit Fortran	Enhancement request - add support for LEADZ(),POPCNT(),POPPAR()
3564	32/64-bit Fortran	Request: make -Mbackslash the default, or make -Mstandard imply -Mbackslash
3595	32/64-bit C/C++	pgCC build seems to hang in IPA during link step
3602	32/64-bit C/C++	pgCC openmp program fails with schedule (static)
3717	32/64-bit C/C++	Enhancement request: support packed struct, __attribute__((packed)), for pgcc
3726	All	DWARF3: use DW_AT_MIPS_LINKAGE_NAME for mangled C++, Fortran 90 names
3754	64-bit C/C++	64-bit pgcc with -mp and omp parallel for cannot handle long iterators
3777	32/64-bit Fortran	Forward reference to module function from earlier module function in character length fails
3778	32/64-bit Fortran	Forward reference to module function from earlier module function in character length fails
3796	32/64-bit Fortran	Fortran use module, only:operator(.x.) gives spurious error message about .neqv.
3818	64-bit Fortran	Win64 -Mchkstk with -fPIC fails at link time
3928	32/64-bit Fortran	Fortran Dwarf2 problem - need DW_AT_type for function as dummy argument
3939	32/64-bit Fortran	Enhancement Request - User's example should not complain 'Non-constant expression'

TPR	Language / Tool	Description
3970	32/64-bit C/C++	C++ compiler fails (segmentation fault) with > 60 if/else in while loop
3982	32/64-bit Fortran	F90 with multiple directories with two .mod files of the same name confuses pgf90
4001	32/64-bit C/C++	pgcc #pragma omp for with 64-bit index variable not properly implemented
4046	32/64-bit Fortran	Multiple imports of pgf90 shared object using "dlopen" cause runtime failures.
4051	All	ScaLapack example needs revision to work with 6.2-5 CDK
4064	32/64-bit Fortran	Request for support of F2003 GET_COMMAND_ARGUMENT
4070	All	PDF documentation is not well formatted, margins too large
4079	32/64-bit Fortran	request for CVF 'sizeof' in fortran
4081	32-bit C/C++	C++ code compiled '-tp px -fastsee' causes internal compiler error "fr_decr_use: bad usect"
4086	64-bit C/C++	C++ compiler does not implement -Mfcon
4088	All	fstat64 fails for Win64 (Fortran)
4111	32-bit Fortran	On SUA32, -Mchkfpstk can give "fp stack is not empty" runtime errors
4128	32/64-bit Fortran	pgf90 fails to recognize constant PARAMETER array elements in array bounds
4132	32/64-bit C/C++	Feature request: define "__PIC__=1" when "-fPIC" is used.
4134	32/64-bit C/C++	Linking with shared and nonshared libs causes ' static object marked for destruction more than once'
4148	32/64-bit C/C++	pgcpp does not properly recognize "__builtin_expect" (gcc compatibility)
4149	32-bit Fortran	Allow for non-English Administrator group name for Windows/SUA installations
4150	32/64-bit Fortran	Fortran Reads of integers in ascii files with (*) format should catch non-integer strings

TPR	Language / Tool	Description
4151	64-bit C/C++	pgcpp fails with internal compiler error on Win64 with -MD switch
4154	32/64-bit Fortran	time() and ctime() return integer*8 - not integer*4 as documentation indicates
4158	32/64-bit Fortran	Request for F2008 C_SIZEOF function
4160	32/64-bit Fortran	PGF90 fails to inline character function with spurious message about argument count
4163	32/64-bit Fortran	PGF90 PARAMETER array set by a reshaping another array fails
4167	32/64-bit Fortran	pgf90 does not support transfer function in parameter specification
4173	32/64-bit Fortran	OpenMP parallel region IF clause with(.false.) incorrectly runs in parallel
4175	All	Linking with -lacml or -lacml_mp requires -lacml_mv as well
4180	32/64-bit Fortran	pgf90 reporting error when attempting to inline pointer function
4185	32-bit C/C++	pgcc fails with internal compiler error with -mp/-Mconcur, 'Unable to allocate a register 22'
4191	32-bit Fortran	32-bit pgf77/pgf90 gives internal compiler error at -O2, 'Unable to allocate a register'
4200	32/64-bit Fortran	SAVE attribute assigned to allocatable, flagged as warning
4201	32/64-bit Fortran	SYSTEM_CLOCK does not have enough resolution
4202	32/64-bit Fortran	NINT() does not agree with other compiler output.
4203	32/64-bit Fortran	Fortran allocatable components of local derived type are not auto-deallocated
4205	32/64-bit Fortran	pghpf fails with internal compiler error - 'mk_mem_ptr_shape: extnt not subs'
4207	32/64-bit Fortran	Please refer to ISO/IEC 1539-1:1997 for Fortran reference
4208	All	64-bit 'Large-array' programming examples should add Win64 limitations

TPR	Language / Tool	Description
4213	64-bit C/C++	Compiler takes a long time to compile files with lots of functions
4214	32/64-bit Fortran	Fortran compiler fails with Internal compiler error in register allocation.
4219	32/64-bit Fortran	PGF90 USE,ONLY:OPERATOR(.x.) causes spurious error messages
4224	32/64-bit Fortran	OSX compilers do not support static linking
4226	32/64-bit C/C++	C++ link causes execution error "static object has been marked for destruction more than once"
4233	32/64-bit Fortran	Allow expression using PARAMETER array element in dimension of another module array
4235	32/64-bit Fortran	Fortran RAN() function does not work properly with -r8
4241	32/64-bit C/C++	pgcc gives spurious error message for barrier in nested OpenMP parallel region
4244	32/64-bit Fortran	Nested OpenMP parallel regions should compile with -mp
4245	32-bit Fortran	pgf77/pgf90 fail with internal compiler error in register allocation, with -O -fPIC
4246	32/64-bit Fortran	Incorrect error message with use, only::usergeneric.
4247	32/64-bit C/C++	Add pgCC -a, like pgCC -A, but generating warnings, not errors
4248	32/64-bit C/C++	pgCC failure with -O3 -g --- undefined label in generated asm code
4249	32/64-bit Fortran	Fortran compiler should give error, not fail, with SUM(1:n)
4251	32/64-bit Fortran	There was no documentation for the $PGI/target/version/src directory.
4253	32/64-bit Fortran	Compiler segfaults with Fortran PARAMETER initialized with implied DO
4254	32/64-bit Fortran	Fortran trim call and // in argument character length attribute causes runtime error
4257	32/64-bit Fortran	Allow non-default-kind logical argument to SUM MASK argument

TPR	Language / Tool	Description
4259	32/64-bit C/C++	Win32 compiler driver fails for large @objfilelist
4265	32/64-bit C/C++	pgCC -O2 -g example causes "Error: can't resolve `.text' .LBxxxx"
4266	32/64-bit Fortran	Host allocatable not auto-deallocated if only allocated in contained routine
4268	32/64-bit Fortran	Using MS$ATTRIBUTES C without explicit interface
4271	32-bit Fortran	pgf77 with -fpic -tp px generated bad code accessing static variables
4274	All	The install script should modify the default value of $PGI in lmgrd.rc to install directory.
4276	32-bit Fortran	Compiler fails with -mp with adjustable array in private clause of parallel region
4279	64-bit C/C++	asm() example with 'm' constraint was not restricted to memory
4280	32/64-bit C/C++	Bad assembly generated with C++ using -O -g
4283	32-bit Fortran	DEC$ATTRIBUTE DLLIMPORT,ALIAS names should not be decorated (Windows)
4284	32-bit Fortran	Request to add GETDAT and GETTIM to lib3f
4286	64-bit Fortran	Fortran LOG function with -Ktrap=denorm generates floating point exception
4287	All	When upgrading a CDK installation with a new compiler release, installcdk should not fail.
4288	All	User Guide should explain -Ktrap=inexact in more detail
4292	32/64-bit Fortran	Passing Dummy allocatable arrays not working
4298	32-bit Fortran	Failure when using -Munroll -tp px switches
4300	32/64-bit Fortran	Dwarf2 information was not being generated for Fortran module variables on Windows
4302	64-bit Fortran	pgf90 shape() fails with -Mlarge_arrays
4306	32/64-bit C/C++	pgcc -Mcpp -Bstatic causes the driver to call compiler with no input file

TPR	Language / Tool	Description
4308	32/64-bit Fortran	Fortran derived type dummy argument with initialized type component caused compiler failure
4310	32/64-bit C/C++	Add better messages when replacing loop by call to optimized runtime routine
4312	32-bit Fortran	Infinite loop in compiler with LRE example
4314	64-bit Fortran	Incorrect loop address used for Fortran dummy argument when expanded in vector loop
4316	64-bit Fortran	PGF90 syntax error issued for DO construct name immediately following USE
4318	64-bit Fortran	PGF95 fails with internal compiler error compiling program with error in deallocate statement
4319	64-bit Fortran	Use of -tp x64 -Mprof=func with contained subroutines yields link time undefined references

8 Contact Information and Documentation

You can contact The Portland Group at:

The Portland Group
STMicroelectronics, Inc.
Two Centerpointe Drive
Lake Oswego, OR 97035 USA

The PGI User Forum is monitored by members of the PGI engineering and support teams as well as other PGI customers. The forum newsgroups may contain answers to commonly asked questions. Log in to the PGI website to access the forum:

http://www.pgroup.com/userforum/index.php

Or contact us electronically using any of the following means:

Fax: *+1-503-682-2637*
Sales: *sales@pgroup.com*
Support: *trs@pgroup.com*
WWW: *http://www.pgroup.com*

All technical support is by e-mail or submissions using an online form at *http://www.pgroup.com/support*. Phone support is not currently available.

Many questions and problems can be resolved at our frequently asked questions (FAQ) site at *http://www.pgroup.com/support/faq.htm*.

Online documentation is available at *http://www.pgroup.com/doc* or in your local copy of the documentation in the release directory *doc/index.htm*.