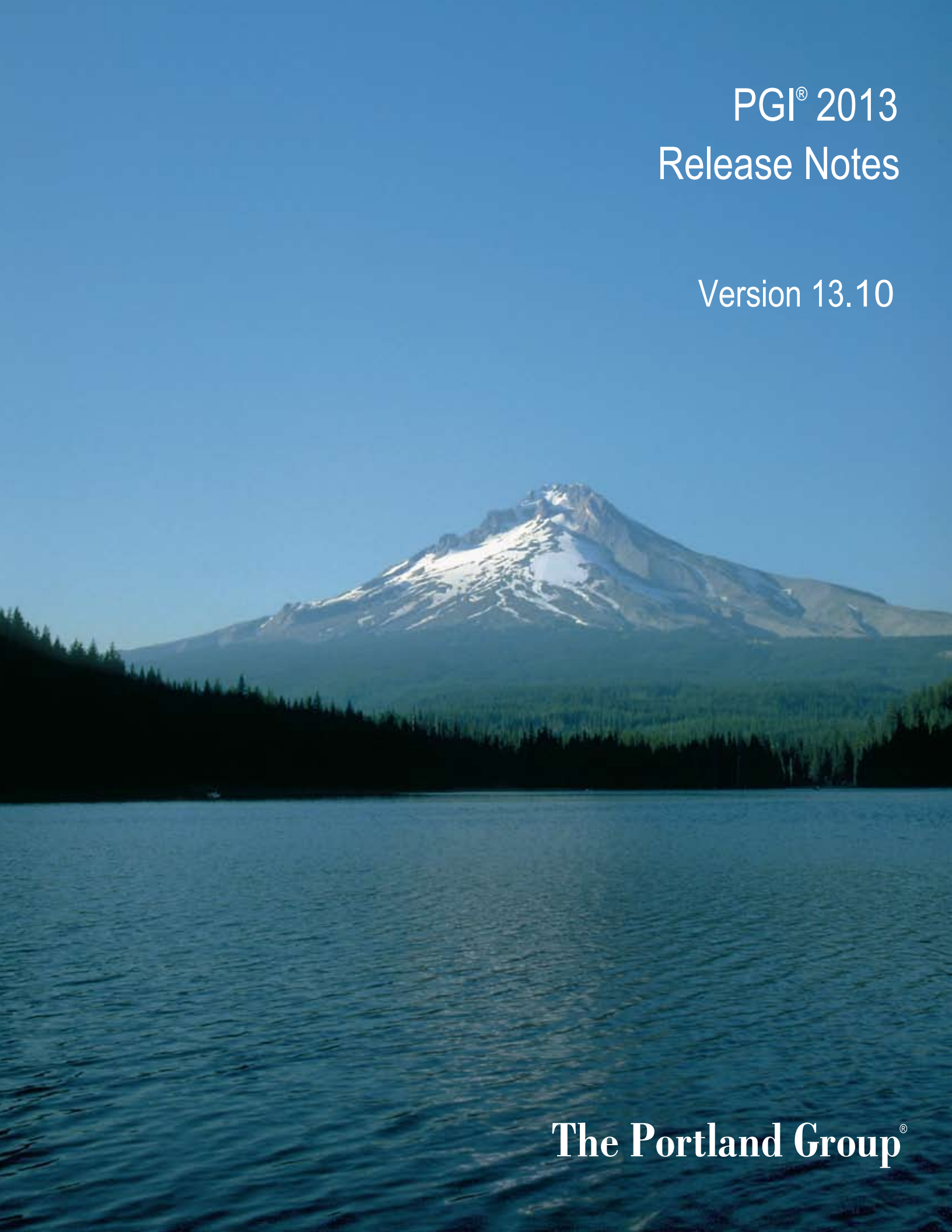
PGI® 2013 Release Notes

Version 13.10



The Portland Group®

PGI® 2013 Release Notes

Printed in the United States of America
First Printing: Release 2013, version 13.1, January 2013
Second Printing: Release 2013, version 13.2, February 2013
Third Printing: Release 2013, version 13.3, March 2013
Fourth Printing: Release 2013, version 13.4, April 2013
Fifth Printing: Release 2013, version 13.5, May 2013
Sixth Printing: Release 2013, version 13.6, June 2013
Seventh Printing: Release 2013, version 13.7, July 2013
Eighth Printing: Release 2013, version 13.8, August 2013
Ninth Printing: Release 2013, version 13.9, September 2013
Tenth Printing: Release 2013, version 13.10, October 2013

Technical support: trs@pgroup.com
Sales: sales@pgroup.com
Web: www.pgroup.com

ID: 132821348

Contents

Figures

Tables

Chapter 1. Release Overview

Welcome to Release 2013 of *PGI Workstation*™, *PGI Server*™, and the *PGI CDK*® *Cluster Development Kit*®, a set of compilers and development tools for 32-bit and 64-bit x86-compatible processor-based workstations, servers, and clusters running versions of the Linux operating system. *PGI Workstation* and *PGI Server* are also available for the Apple OS X operating system.

This document describes changes between previous versions of the PGI 2013 release as well as late-breaking information not included in the current printing of the *PGI Compiler User's Guide*.

Product Overview

PGI Workstation, PGI Server, and the *PGI CDK* include exactly the same PGI compiler and tools software. The difference is the manner in which the license keys enable the software.

Licensing Terminology

The PGI compilers and tools are license-managed. It is useful to have common terminology. These two terms are often confused, so they are clarified here:

- **License** – a legal agreement between NVIDIA and PGI end-users, to which users assent upon installation of any PGI product. The terms of the License are kept up-to-date in documents on pgroup.com and in the $PGI/<platform>/<rel_number> directory of every PGI software installation.
- **License keys** – ASCII text strings that enable use of the PGI software and are intended to enforce the terms of the License. License keys are generated by each PGI end-user on pgroup.com using a unique hostid and are typically stored in a file called `license.dat` that is accessible to the systems for which the PGI software is licensed.

License Options

PGI offers licenses for either x64+GPU or x64 only platforms. *PGI Accelerator*™ products, the x64+GPU platform products, include support for the directive-based PGI Accelerator programming model, CUDA Fortran and PGI CUDA-x86. PGI Accelerator compilers are supported on all Intel and AMD x64 processor-based systems with CUDA-enabled NVIDIA GPUs running Linux, OS X, or Windows.

PGI Workstation and PGI Server Comparison

- All *PGI Workstation* products include 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. The product and *license server* are on the same local machine.
- *PGI Server* products are offered in configurations identical to *PGI Workstation*, but include 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 *license server*, that is, the system on which the *PGI Server* license keys are installed. There can be multiple installations of the *PGI Server* compilers and tools on machines connected to the license server; and the users can use the product concurrently, provided they are issued a license key by the license server.

PGI CDK Cluster Development Kit

A cluster is a collection of compatible computers connected by a network. The *PGI CDK* supports parallel computation on clusters of 32-bit and 64-bit x86-compatible AMD and Intel processor-based Linux workstations or servers interconnected by a TCP/IP-based network, such as Ethernet.

Support for cluster programming does not extend to clusters combining 64-bit processor-based systems with 32-bit processor-based systems, unless all are running 32-bit applications built for a common set of working x86 instructions.

Note

Compilers and libraries can be installed on other platforms not in the user's cluster, including another cluster, as long as all platforms use a common floating license server.

Release Components

Release 2013 includes the following components:

- PGFORTRAN™ native OpenMP and auto-parallelizing Fortran 2003 compiler.
- PGCC® native OpenMP and auto-parallelizing ANSI C99 and K&R C compiler.
- PGC++® native OpenMP and auto-parallelizing ANSI C++ compiler.
- *PGPROF*® MPI, OpenMP, and multi-thread graphical profiler.
- *PGDBG*® MPI, OpenMP, and multi-thread graphical debugger.
- MPICH MPI libraries, version 1.2.7, for both 32-bit and 64-bit development environments (Linux only).

Note

64-bit linux86-64 MPI messages are limited to <2GB size each.

- Precompiled OpenMPI library for both 32-bit and 64-bit MacOS development environments.
- A UNIX-like shell environment for 32-bit and 64-bit Windows platforms.

- FlexNet license utilities.
- Documentation in PDF and man page formats.

Additional components for PGI CDK

PGI CDK for Linux also includes these components:

- MPICH2 MPI libraries, version 1.0.5p3, for both 32-bit and 64-bit development environments.
- MVAPICH MPI libraries, version 1.1, for both 32-bit and 64-bit development environments.
- ScaLAPACK linear algebra math library for distributed-memory systems, including BLACS version 1.1 – the Basic Linear Algebra Communication Subroutines) and ScaLAPACK version 1.7 for use with MPICH or MPICH2 and the PGI compilers on Linux systems with a kernel revision of 2.4.20 or higher. This is provided in both linux86 and linux86-64 versions for AMD64 or Intel 64 CPU-based installations.

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

You can use PGI products to develop, debug, and profile MPI applications. The MPI profiler and debugger included with *PGI Workstation* are limited to eight local processes. The MPI profiler and debugger included with *PGI Server* are limited to 16 local processes. The MPI profiler and debugger included with *PGI CDK* supports up to 256 remote processes.

Terms and Definitions

These release notes contain a number of terms and definitions with which you may or may not be familiar. If you encounter an unfamiliar term in these notes, please refer to the online glossary at

www.pgroup.com/support/definitions.htm

These two terms are used throughout the documentation to reflect groups of processors:

- 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 term includes the AMD Athlon64, AMD Opteron, AMD Turion, AMD Barcelona, AMD Shanghai, AMD Istanbul, and AMD Bulldozer processors.
- Intel 64 – 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, Intel Core 2, Intel Core 2 Duo (Penryn), Intel Core (i3, i5, i7) both first generation (Nehalem) and second generation (Sandy Bridge) processors.

Supported Platforms

There are six platforms supported by the *PGI Workstation* and *PGI Server* compilers and tools. Currently, *PGI CDK* supports only the first four of these.

- *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 OS X* – supported on 32-bit Apple operating systems running on either a 32-bit or 64-bit Intel-based Mac system.
- *64-bit OS X* – supported on 64-bit Apple operating systems running on a 64-bit Intel-based Mac system.

Supported Operating Systems

This section describes updates and changes to PGI 2013 that are specific to Linux, OS X, and Windows.

Linux

None.

OS X

PGI 2013 for OS X supports most of the features of the 32-bit and 64-bit versions for linux86 and linux86-64 environments. Except where noted in these release notes or the user manuals, the PGI compilers and tools on OS X function identically to their Linux counterparts.

Windows

PGI 2013 for Windows supports most of the features of the 32-bit and 64-bit versions for linux86 and linux86-64 environments.

Getting Started

By default, the PGI 2013 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`.

These aggregate options incorporate a generally optimal set of flags for targets that support SSE capability. These options incorporate optimization options to enable use of vector streaming SIMD instructions for 64-bit targets. They enable vectorization with SSE 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 original 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
`-Mpre`	Indicates partial redundancy elimination

`-fast` for 64-bit targets and `-fastsse` for both 32- and 64-bit targets also typically include:

`-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 PGFORTRAN 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 that are described in the *PGI Compiler Reference Manual*, 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.

Chapter 2. New or Modified Features

This chapter provides information about the new or modified features of Release 2013 of the PGI compilers and tools.

What's New in Release 2013

13.10 Updates and Additions

- This release contains a number of changes and additions to the `-ta` and `-Mcuda` options:

  Target Accelerators option chnages:

  - Added `-ta=nvidia:noL1` to prevent use of the L1 hardware data cache to cache global variables.
  - Replaced `-ta=nvidia:keepbin`, with `-ta=nvidia:keep` which keeps kernel files.
  - Replaced `-ta=nvidia:keepgpu` with `-ta=nvidia:keep` which keeps kernel files.
  - Replaced `-ta=nvidia:keepptx`) with `-ta=nvidia:keep` which keeps kernel files.
  - Removed `-ta=nvidia:mul24`.
  - Removed `-ta=nvidia:[no]wait`.
  - Removed `-ta=nvidia:analysis`.

  Cuda Fortran changes:

  - Added `-Mcuda=noL1` to prevent use of the L1 hardware data cache to cache global variables.
- A number of problems have been corrected in this release. Refer to `www.pgroup.com/support/release_tprs.htm` for a complete and up-to-date table of technical problem reports, TPRs, fixed in recent releases of the PGI compilers and tools.

13.9 Updates and Additions

- PGI Accelerator x64+GPU native Fortran 2003 compilers and CUDA Fortran support the CUDA 5.0 Toolkit as the default toolkit.

CUDA 5.0 has deprecated the use of character strings in the symbol API, i.e. functions such as cudaMemcpyToSymbol(). Therefore, CUDA Fortran no longer needs the cudaSymbol derived type, and its definition and use has been removed from the cudafor module. Users should recompile any source code which uses the cudafor module after updating to PGI 13.9 and later.

PGI compilers and tools also support the CUDA 5.5 Toolkit. For information on specifying the toolkit version, refer to "PGI Accelerator and CUDA Fortran Enhancements," on page 16.

- PGI now enables CUDA Fortran device code to access compute capability 3.x shuffle functions, including `__shfl()`, `__shfl_up()`, `__shfl_down()`, and `__shfl_xor()`. These functions enable access to variables between threads within a warp, referred to as lanes. In CUDA Fortran, lanes use Fortran's 1-based numbering scheme. For more information on these functions, refer to "Shuffle Functions," on page 21.

13.7 Updates and Additions

- OpenACC/CUDA Fortran profiling supports multiple GPUs. In addition, there are improvements to accelerator profiling as a whole. For more information, refer to "OpenACC/CUDA Fortran Profiling," on page 13.
- C++ `-Minfo` messages now use unmangled names.
- General performance improvements on AVX vectorized code.
- Support for new CUDA Fortran atomics. For a complete list, refer to "CUDA Fortran Atomic Functions," on page 12.
- PGI 13.7 implements the record format described in *Intel Fortran User's Guide Vol. 1, Building Applications, Variable-Length Records Greater than 2 Gigabytes.* This record format is now the default for the PGI Fortran compilers. To access files written using the old file format, set the environment variable `FORTRANOPT` to `pgi_legacy_large_rec_fmt`.
- Debugger support for hardware data watchpoints on Linux kernels greater than 2.6.32.

13.6 Updates and Additions

- Added support for the F2008 *impure* attribute. Fortran 2008 removed the restriction of Fortran 95 that elemental procedures be implicitly pure. Elemental procedures permit writing procedures as many as 16 times, once for each possible rank. In Fortran 2008, elemental procedures must now be explicitly declared with the prefix *impure*.

13.5 Updates and Additions

- PGDBG and PGPROF support debugging and profiling of MPI programs built with SGI MPI.

  To debug an SGI MPI program, use the PGDBG `-sgimpi` option, which has the same syntax as the `-mpi` option.

  To profile an SGI MPI program, build it with `-Mprof=func,sgimpi`, `-Mprof=lines,sgimpi`, or `-Mprof=time,sgimpi`. You must specify `sgimpi` even if you use `mpicc` or `mpif90` to build your program.
- Added support for two F2008 features:

  - Finding a unit when opening a file
  - Derived-type-style accesses of the real and imaginary parts of a complex number.
- Made several performance improvements for complex arithmetic, for all languages.
- A number of problems have been corrected in this release. Refer to `www.pgroup.com/support/release_tprs.htm` for a complete and up-to-date table of technical problem reports, TPRs, fixed in recent releases of the PGI compilers and tools.

13.4 Updates and Additions

- PGI 13.4 contains support for F2003 non-default derived-type i/o. This feature allows programmers to provide their own formatted and unformatted i/o routines for user-defined or, in certain cases, vendor-supplied derived types. The interfaces for the called subroutines are very specific. For more information on the interfaces refer to a recent Fortran reference manual.

  For a short example of how to use this new feature, refer to "New or Modified Fortran Functionality," on page 12.
- PGI 13.4 contains initial support for F90 pointers in CUDA Fortran device code. In the current implementation, the pointer declaration must be at module scope, in the module which contains the device code. As usual, the host code can use the module and manipulate the pointers. The host code can also pass the pointer as an argument.

  For an example of how to use this new feature, refer to "Using F90 Pointers in CUDA Fortran Device Code," on page 20.
- PGI 13.4 does not require the system configuration for debugging and profiling on Apple OS X that was required in previous releases. The debugger and profile data collector no longer need to run using "`group procmod`" privileges. However, to use PGDBG to attach to a running process, users must be able to enter the authentication credentials of a user who is a member of the unix group "_developer". For more information, refer to the section *System Configuration to Enable Debugger 'Attach'* in the *PGI 2013 Installation Guide*.

Updates and Additions prior to 13.4

- PGI now supports OpenMP 3.1. Specifically, PGI now supports these features:
  - Fortran, C, and C++ compilers and runtime:
  - omp_in_final runtime routine
  - `OMP_NUM_THREADS` and `OMP_PROC_BIND` environment variables
  - taskyield
  - final tasks and the final clause on task constructs
  - mergeable clause on task constructs
  - atomic read, write, capture, and update clauses
  - min/max reductions in C/C++

  - firstprivate with const data in C/C++, intent(in) in Fortran
- Added support for pgc++ on Linux, which is the PGI gnu-compatible C++ compiler.
- The PGI C++ compilers pgcpp and (Linux) pgc++ support the OpenACC and PGI Accelerator directives.
- Added support for F2003 deferred type parameters, recursive I/O. parameterized derived types, and deferred character length.
- PGI provides ACML version 5.3.0 for 64-bit Linux and Windows. For 32-bit operating systems, PGI continues to provide ACML version 4.4.0.
- PGI Accelerator x64+GPU native Fortran 2003 and C99 compilers and CUDA Fortran support the CUDA 4.2 Toolkit as the default toolkit. PGI compilers and tools also support CUDA 5.0 Toolkit. For information on specifying the Toolkit version, refer to "PGI Accelerator and CUDA Fortran Enhancements," on page 16.
- Starting in PGI 13.1, our C++ Compiler now utilizes EDG version 4.5. This version update enables a number of C++11 language features, when compiled with the `--c++11` option. For a list of these features, refer to "C++11 Features," on page 23.
- PGI 13.1 supports a number of new command line options as well as new keyword subarguments for the existing command line options: `-ta=nvidia` and `-Mcuda`. For more information, refer to "New or Modified Compiler Options," on page 10.
- CUDA Fortran now supports separate compilation and linking of device routines, including device routines in Fortran modules.

  To enable separate compilation and linking, include the command line option `-Mcuda=rdc` on both the compile and the link steps.
- The OpenACC runtime has the concept of device type and device number. For more information, refer to "Device ID and Device Number," on page 17
- In Fortran and C, there is a new `deviceid` clause for the data, parallel, kernels, update and wait directives. The `deviceid` clause has a single scalar integer argument. For more information, refer to "DEVICEID clause," on page 18
- PGI supports a command line option, `-Mstack_arrays`, which allows Fortran automatic arrays to be allocated from the stack.

New or Modified Compiler Options

PGI 13.10 supports a number of new command line options as well as new keyword subarguments for existing command line options. These changes include:

- `-Mstack_arrays` and `-Mnostack_arrays`

  `-Mstack_arrays` places automatic arrays on the stack. while `-Mnostack_arrays` allocates automatic arrays on the heap. `-Mnostack_arrays` is the default and what traditionally has been the approach used.
- `-ta=nvidia` has the following new target accelerator arguments:
  - `-ta=nvidia:noL1` prevents the use of L1 hardware data cache to cache global variables.

- `-ta=nvidia:keep` keeps kernel files.
- `-ta=nvidia,tesla` is equivalent to `-ta=nvidia,cclx`
- `-ta=nvidia,fermi` is equivalent to `-ta=nvidia,cc2x`
- `-ta=nvidia,kepler` is equivalent to `-ta=nvidia,cc3x`

PGI continues to support the specific compute capability options (cc10, cc11, etc.) as well as the new cc35 option for compute capability 3.5.

- `-Mcuda` has the following new target accelerator arguments:
  - `-Mcuda=noL1` prevents the use of L1 hardware data cache to cache global variables.
  - `-Mcuda,tesla` is equivalent to `-Mcuda,cclx`
  - `-Mcuda,fermi` is equivalent to `-Mcuda,cc2x`
  - `-Mcuda,kepler` is equivalent to `-Mcuda,cc3x`

  PGI continues to support the specific compute capability options (cc10, cc11, etc.) as well as the new cc35 option for compute capability 3.5.
- `-Mcuda` has the new option `rdc` that enables CUDA Fortran separate compilation and linking of device routines, including device routines in Fortran modules. To enable separate compilation and linking, include the command line option `-Mcuda=rdc` on *both* the compile and the link steps.
- `-Mipa` has the new suboption `-Mipa=reaggregation` that enables IPA-guided structure reaggregation. This automatically attempts to reorder elements in a struct or to split structs into substructs to improve memory locality and cache utilization.
- `-O`

  The optimizations performed at `-O`, `-O1`, `-O2`, `-O3` and `-O4` have changed. Beginning in 13.1, these options have these meanings:

  –O0
  : Level zero specifies no optimization. A basic block is generated for each language statement.

  –O1
  : Level one specifies local optimization. Scheduling of basic blocks is performed. Register allocation is performed.

  –O
  : When no level is specified, level two global optimizations are performed, including traditional scalar optimizations, induction recognition, and loop invariant motion. No SIMD vectorization is enabled.

  –O2
  : Level two specifies global optimization. This level performs all level-one local optimization as well as level-two global optimization described in –O. In addition, more advanced optimizations such as SIMD code generation, cache alignment, and partial redundancy elimination are enabled.

–O3
: Level three specifies aggressive global optimization. This 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 four performs all level-one, level-two, and level-three optimizations and enables hoisting of guarded invariant floating point expressions.

- `-Mfprelaxed` has the following new suboption: `-Mfprelaxed=intrinsic` that enables use of relaxed precision intrinsics.
- `-tp` has the following new command line target processor options:

  -piledriver
  : generate code that is usable on any Piledriver processor-based system.

  -piledriver-32
  : generate 32-bit code that is usable on any Piledriver processor-based system.

  -piledriver-64
  : generate 64-bit code that is usable on any Piledriver processor-based system.

Unknown options are treated as errors instead of warnings. This feature means it is a compiler error to pass switches that are not known to the compiler; however, you can use the switch `–noswitcherror` to issue warnings instead of errors for unknown switches.

New or Modified Fortran Functionality

PGI 13.10 contains additional fortran functionality.

CUDA Fortran Atomic Functions

PGI 13.10 supports new CUDA Fortran atomics. Table 2.1 indicates the data types supported prior to 13.8 (`prev`) and the additional data types that PGI now supports (`new`) for each atomic function.

Table 2.1. Supported CUDA Fortran Atomic Functions

	integer*4	integer*8	real*4	real*8
atomicCAS	prev	new	new	new
atomicADD	prev	prev	prev	prev
atomicEXCH	prev	prev	prev	new
atomicSUB	prev	new	new	new
atomicMAX	prev	new	new	new
atomicMIN	prev	new	new	new

F2003 Non-default Derived Type I/O

PGI 13.10 supports F2003 non-default derived-type i/o. Using this feature allows programmers to provide their own formatted and unformatted i/o routines for user-defined derived types and for some vendor-supplied derived types. The following module is one example of a user-defined derived type:

```
module m1
   use iso_c_binding
   interface write(formatted)
     module procedure print_c_ptr
   end interface
   contains
     subroutine print_c_ptr(dtv,unit,iotype,v_list,iostat,iomsg)
       class(c_ptr), intent(in)    :: dtv
       integer, intent(in)         :: unit
       character(*), intent(in)    :: iotype
       integer, intent(in)         :: v_list(:)
       integer, intent(out)        :: iostat
       character(*), intent(inout) :: iomsg
       integer(c_intptr_t)         :: iptrval
       iptrval = transfer(dtv, iptrval)
       if (c_intptr_t.eq.4) then
         write(unit,fmt='("0x",z8.8)') iptrval
       else
         write(unit,fmt='("0x",z16.16)') iptrval
       endif
     end subroutine print_c_ptr
end module m1

program p1
use m1
type(c_ptr) :: p
integer, target :: i, j
p = c_loc(i)
print *,p
end
```

You can compile and run the preceding example using 32 or 64-bit compilers:

32-bit compilers	64-bit compilers
% pgf90 -m32 derivedio.f90 % ./a.out 0x080AA4D4	% pgf90 -m64 derivedio.f90 % ./a.out 0x00007FFF76AD6490

New or Modified Tools Functionality

This section provides information about the debugger, *PGDBG*, and the profiler, *PGPROF*.

OpenACC/CUDA Fortran Profiling

In addition to general improvements to accelerator profiling, *PGPROF* includes all-new support for accelerator profiling of multiple devices. As with profiling of single devices, you can view information such as accelerator performance or device configuration. Figure 2.1 shows a view of *PGPROF* that details the accelerator performance on two GPUs for a routine.

Figure 2.1. Accelerator Performance when Profiling Multiple Devices

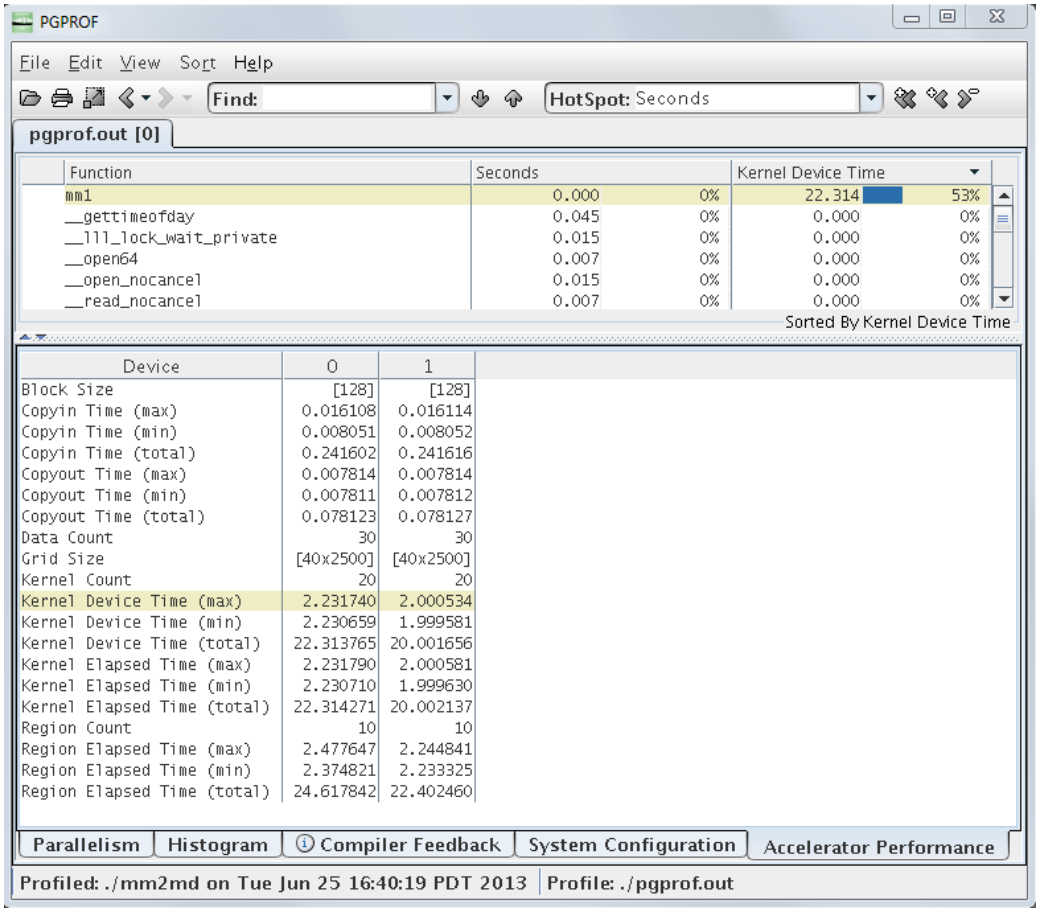
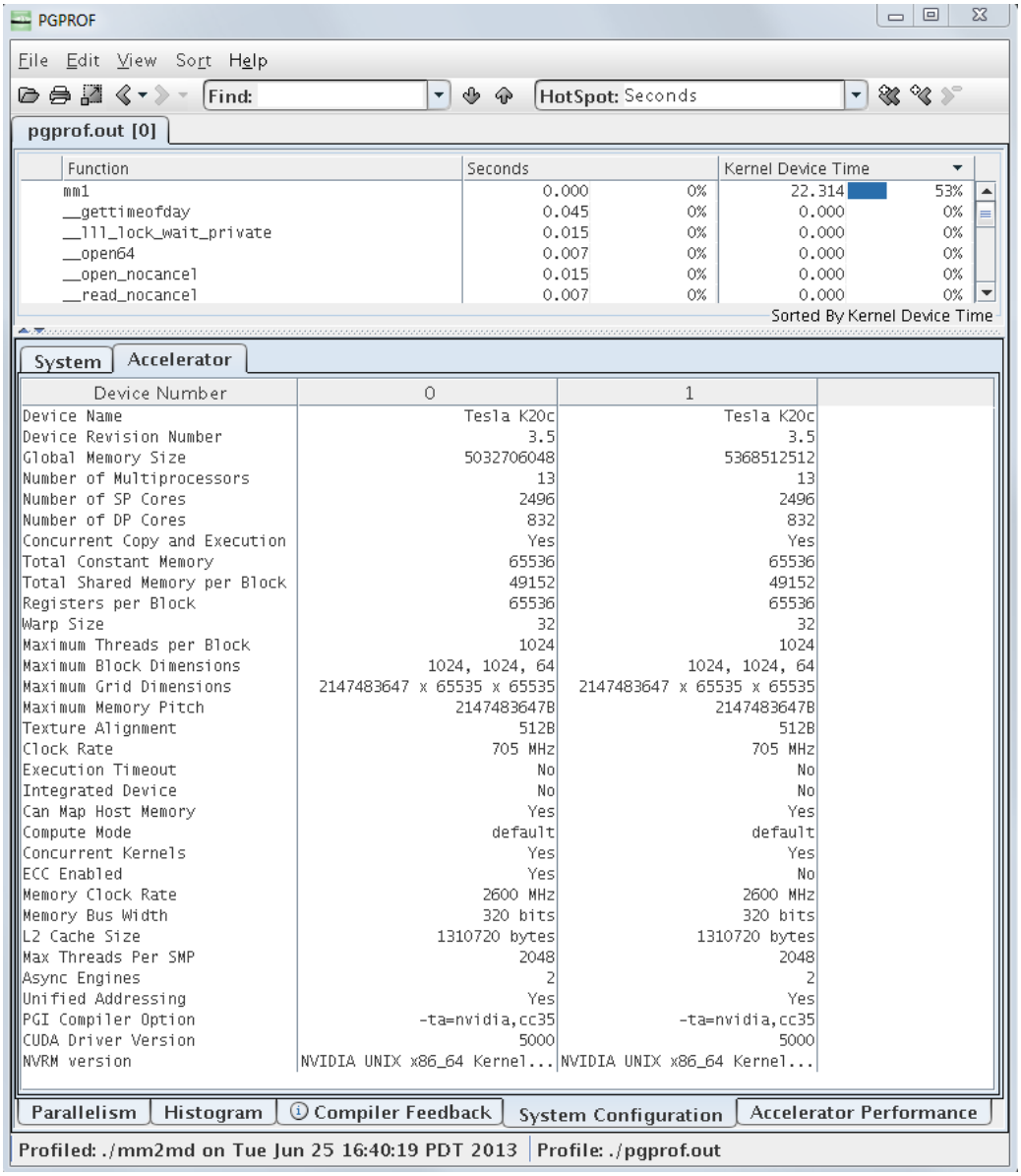


Figure 2.2 shows the details about the accelerators on the system where the profile was run.

Figure 2.2. Accelerator Details when Profiling Multiple Devices



Debug SGI MPI Programs

In PGI 13.10 PGDBG and PGPROF support debugging and profiling of MPI programs built with SGI MPI. To debug an SGI MPI program, use the PGDBG `-sgimpi` option, which has the same syntax as the `-mpi` option. To profile an SGI MPI program, build it with `-Mprof=func,sgimpi`, `-Mprof=lines,sgimpi`, or `-Mprof=time,sgimpi`. You must specify `sgimpi` even if you use `mpicc` or `mpif90` to build your program.

Local and Remote Debugging

PGDBG is licensed software available from The Portland Group. *PGDBG* supports debugging programs running on local and remote systems. The PGI license keys that enable *PGDBG* to debug must be located on the same system where the program you want to debug is running.

Local debugging
: If you want to debug a program running on the system where you have launched *PGDBG*, you are doing local debugging and you need license keys on that local system.

Remote debugging
: If you want to debug a program running on a system other than the one on which *PGDBG* is launched, then you are doing remote debugging and you need license keys on the remote system. The remote system also needs an installed copy of PGI Workstation, PGI Server, or PGI CDK.

Using MPICH-2 on Linux

PGI CDK for Linux includes MPICH-2 libraries, tools, and licenses required to compile, execute, profile, and debug MPI programs.

If you want to build your MPI application using the instance of MPICH-2 installed with the PGI compilers, you need to append the location of `libmpl.so.1` to the `LD_LIBRARY_PATH` environment variable.

For 32-bit:

```
%setenv LD_LIBRARY_PATH "$LD_LIBRARY_PATH":$PGI/linux86/2013/mpi2/mpich/libso:
/$PGI/linux86/13.10/libso
```

For 64-bit:

```
%setenv LD_LIBRARY_PATH "$LD_LIBRARY_PATH":$PGI/linux86-64/2013/mpi2/mpich/libso:
$PGI/linux86-64/13.10/libso
```

You may need to put in `.cshrc/.bashrc` when running a program on slave nodes.

Then add the `-Mmpi=mpich2` option to the compilation and link steps, or you can use the `-Mprof=mpich2` option to instrument for MPICH-2 profiling. The `-Mmpi=mpich2` option automatically sets up the include and library paths to use the MPICH-2 headers and libraries. For example, you can use the following command to compile for profiling with MPICH-2:

```
% pgfortran -fast -Mprof=mpich2,time my_mpi_app.f90
```

To use a different instance of MPICH-2, set the `MPIDIR` environment variable before invoking and linking with `-Mmpi=mpich2`. `MPIDIR` specifies the location of the instance of MPI to use. For example, set `MPIDIR` to the root of the MPICH-2 installation directory that you want to use, that is, the directory that contains `bin`, `include`, `lib`, and so on.

PGI Accelerator and CUDA Fortran Enhancements

CUDA Toolkit Version

The PGI Accelerator x64+GPU compilers with OpenACC and CUDA Fortran compilers now support the CUDA 5.0 toolkit as the default. The compilers and tools also support the CUDA 5.5 Toolkit. To specify the version of the CUDA Toolkit, use one of the following options:

In PGI Accelerator:

For CUDA Toolkit 4.2: `-ta=nvidia:cuda4.2`

For CUDA Toolkit 5.0: `-ta=nvidia:cuda5.0`
For CUDA Toolkit 5.5: `-ta=nvidia:cuda5.5`

For CUDA Fortran:

For CUDA Toolkit 4.2: `-Mcuda=cuda4.2`
For CUDA Toolkit 5.0: `-Mcuda=cuda5.0`
For CUDA Toolkit 5.5: `-Mcuda=cuda5.5`

You may also specify a default version by adding a line to the `siterc` file in the installation `bin/` directory or to a file named `.mypgirc` in your `home` directory. For example, to specify CUDA Toolkit 5.5, add the following line to one of these files:

```
    set DEFCUDAVERSION=5.5;
```

Support for CUDA Toolkit versions 3.2 and earlier has been removed.

Default Target Accelerator

When compiling with the OpenACC command line option `-acc`, the default target accelerators are 'nvidia' and 'host' if CUDA extensions are not enabled. The host is treated as another accelerator device, and is enabled by default, for both 64-bit and 32-bit targets. Compiling for NVIDIA GPUs only can be enabled by including the command line option `-ta=nvidia`. You may also change the default target accelerators by adding a line to the `siterc` file in the installation `bin/` directory or to a file named `.mypgirc` in your `home` directory. To set the default target accelerator to NVIDIA GPUs only, add the following line to one of these files:

```
    set DEFACCEL=nvidia;
```

Compiling for both NVIDIA GPUs and the host can always be enabled by including the command line option `-ta=nvidia,host`.

CUDA extensions are enabled with a `.cuf` or `.CUF` file suffix, or by including the `-Mcuda` command line option. When CUDA extensions are enabled, the default target accelerator is `nvidia`. Compiling with CUDA extensions for the host is not fully supported in this release.

Multiple Devices and Host as Device

This release allows use of multiple devices from a single program and a single host thread. The host is treated as another device. The routine `acc_get_num_devices` now returns a non-negative number, since the host is always available as a device. The program may dynamically set and reset which device to use by calling the `acc_set_device_type` or `acc_set_device_num` API routines during program execution. When setting the device type to `acc_device_host`, the device number is ignored.

Device ID and Device Number

The OpenACC runtime has the concept of device type and device number. The supported device types for this release are NVIDIA GPUs (`acc_device_nvidia`) and X86 host (`acc_device_host`). For NVIDIA GPUs, the device number is assigned by the CUDA driver API; the first device is number zero, and other devices have positive numbers.

Device ID

This release supports the concept of *device ID*, which is set by the OpenACC runtime. The available devices are enumerated and given a nonnegative integer device ID, starting at one; the host itself is given the highest device ID value.

The number of available devices can be determined by calling the `acc_get_num_devices` API routine. Since the host is always available as a device, this routine always returns a nonnegative number.

At any point in the program, the runtime keeps track of the current device to be used for the next accelerator construct or directive. The current device can be changed for the host thread by calling the `acc_set_device_type` or `acc_set_device_num` API routines. The current device type and number can be determined by calling the `acc_get_device_type` and `acc_get_device_num` routines.

New, simpler API routines are now supported to allow setting or querying the current device using the device ID.

- `acc_set_deviceid` takes a single integer argument, and sets the current device to the device that corresponds to that device ID.
- `acc_get_deviceid` returns the device ID for the current device.

  A value of zero for a device ID argument corresponds to using the current device for this host thread.

If you build your program with the `-ta=nvidia` command line option, only NVIDIA GPU code is generated for the accelerator regions and constructs. At runtime, `acc_get_num_devices` still returns the count of all available devices, including the host, even though your program was not built to run those regions on the host. If your program changes the current device to the host, you get a runtime error when you try to execute a compute region that was only compiled for the NVIDIA GPU.

DEVICEID clause

In Fortran and C, there is a new `deviceid` clause for the data, parallel, kernels, update and wait directives. The `deviceid` clause has a single scalar integer argument.

- If the `deviceid` clause argument is zero, the current device for this host thread is used.
- If the `deviceid` clause argument is nonzero, the current device for this host thread is changed to the corresponding device for the duration of the region created by this construct.

PGI Accelerator Runtime Routines

PGI_ACC_TIME

When timing the accelerator constructs by setting `PGI_ACC_TIME` to 1, you must run the program with the `LD_LIBRARY_PATH` environment variable set to include the `$PGI/linux86-64/13.10/lib or $PGI/linux86/13.10/lib` directory (as appropriate). This release dynamically loads a shared object to implement the profiling feature, and the path to the library must be available.

For complete descriptions of the PGI Accelerator model runtime routines available in version 13.10, refer to Chapter 4, "PGI Accelerator Compilers Reference" of the *PGI Compiler Reference Manual*.

Memory Management in CUDA

A new memory management routine, `cudaMemGetInfo`, returns the amount of free and total memory available (in bytes) for allocation on the device.

The syntax for `cudaMemGetInfo` is:

```
integer function cudaMemGetInfo( free, total )
     integer(kind=cuda_count_kind) :: free, total
```

Declaring Interfaces to CUDA Device Built-in Routines

A Fortran module is available to declare interfaces to many of the CUDA device built-in routines.

To access this module, do one of the following:

- Add this line to your Fortran program:

  ```
  use cudadevice
  ```

- Add this line to your C program:

  ```
  #include <cudadevice.h>
  ```

You can also use these routines in CUDA Fortran global and device subprograms, in CUF kernels, and in PGI Accelerator compute regions both in Fortran and in C. Further, the PGI compilers come with implementations of these routines for host code, though these implementations are not specifically optimized for the host.

For a complete list of the CUDA built-in routines that are available, refer to the *PGI CUDA Fortran Programming and Reference*.

Using the `texture` Attribute in CUDA Fortran

To use the texture attribute as supported in this release, do the following:

1. Add a declaration similar to the following one to a module declaration section that is used in both the host and device code:

   ```
            real, texture, pointer :: t(:)
   ```

2. In your host code, add the target attribute to the device data that you wish to put into texture memory:

   Change:

   ```
          real, device :: a(n)
   ```

   To:

   ```
          real, target, device :: a(n)
   ```

   The target attribute is standard F90/F2003 syntax to denote an array or other data structure that may be "pointed to" by another entity.

3. Tie the global (by module use-association in both the host program and device subroutine) texture declaration to the device array by using the F90 pointer assignment operator, so a simple expression like the following one performs all the underlying CUDA texture binding operations.

   ```
            t =>  a
   ```

Your CUDA Fortran device code contained in the module that declares `t`, or uses a module that contains the declaration of `t`, can now access `t` without any other declaration. For example:

```
! Vector add, s through device memory, t is through texture memory
i = threadIdx%x + (blockIdx%x-1)*blockDim%x
s(i) = s(i) + t(i)
```

Accesses of `t`, targeting `a`, go through the texture cache.

Using F90 Pointers in CUDA Fortran Device Code

PGI 13.10 contains support for F90 pointers in CUDA Fortran device code. In the current implementation, the pointer declaration must be at module scope, in the module which contains the device code. The host code can use the module and manipulate the pointers. The host code can also pass the pointer as an argument.

The following example shows how to use F90 pointers in CUDA Fortran as supported by the current release:

```
! Device pointer in module, and passed as argument

module devptr
! pointer declarations must be in the module in which they are used
   real, device, pointer, dimension(:) :: mod_dev_ptr
   real, device, pointer, dimension(:) :: arg_dev_ptr
   real, device, target,  dimension(4) :: mod_dev_arr
   real, device, dimension(4) :: mod_res_arr
contains
   attributes(global) subroutine test(arg_ptr)
     real, device, pointer, dimension(:) :: arg_ptr
     if (associated(arg_ptr)) then
       mod_res_arr = arg_ptr
     else
       mod_res_arr = mod_dev_ptr
     end if
   end subroutine test
end module devptr

program test
use devptr
real, device, target, dimension(4) :: a_dev
real  result(20)

a_dev = (/ 1.0, 2.0, 3.0, 4.0 /)

! Pointer assignment to device array declared on host,
! passed as argument
arg_dev_ptr => a_dev
call test<<<1,1>>>(arg_dev_ptr)
result(1:4) = mod_res_arr

!$cuf kernel do <<<*,*>>>
do i = 1, 4
   mod_dev_arr(i) = a_dev(i) + 4.0
   a_dev(i)       = a_dev(i) + 8.0
end do

! Pointer assignment to module array, argument nullified
mod_dev_ptr => mod_dev_arr
arg_dev_ptr => null()
call test<<<1,1>>>(arg_dev_ptr)
```

```
result(5:8) = mod_res_arr

! Pointer assignment to updated device array, now asssociated
arg_dev_ptr => a_dev
call test<<<1,1>>>(arg_dev_ptr)
result(9:12) = mod_res_arr

!$cuf kernel do <<<*,*>>>
do i = 1, 4
   mod_dev_arr(i) = 25.0 - mod_dev_arr(i)
   a_dev(i)       = 25.0 - a_dev(i)
end do

! Non-contiguous pointer assignment to updated device array
arg_dev_ptr => a_dev(4:1:-1)
call test<<<1,1>>>(arg_dev_ptr)
result(13:16) = mod_res_arr

! Non-contiguous pointer assignment to updated module array
nullify(arg_dev_ptr)
mod_dev_ptr => mod_dev_arr(4:1:-1)
call test<<<1,1>>>(arg_dev_ptr)
result(17:20) = mod_res_arr

print *,result
end
```

You can compile and run the preceding example as follows:

```
% pgf90 devptr.cuf
% ./a.out
     1.000000        2.000000        3.000000        4.000000
     5.000000        6.000000        7.000000        8.000000
     9.000000        10.00000        11.00000        12.00000
     13.00000        14.00000        15.00000        16.00000
     17.00000        18.00000        19.00000        20.00000
```

Shuffle Functions

PGI 13.10 enables CUDA Fortran device code to access compute capability 3.x shuffle functions. These functions enable access to variables between threads within a warp, referred to as *lanes*. In CUDA Fortran, lanes use Fortran's 1-based numbering scheme.

__shfl()

`__shfl()` returns the value of `var` held by the thread whose ID is given by `srcLane`. If the `srcLane` is outside the range of `1:width`, then the thread's own value of `var` is returned. The `width` argument is optional in all shuffle functions and has a default value of 32, the current warp size.

```
    integer(4) function __shfl(var, srcLane, width)
       integer(4) var, srcLane
       integer(4), optional :: width
```

```
    real(4) function __shfl(var, srcLane, width)
       real(4) :: var
       integer(4) :: srcLane
       integer(4), optional :: width
```

```
real(8) function __shfl(var, srcLane, width)
   real(8) :: var
   integer(4) :: srcLane
   integer(4), optional :: width
```

__shfl_up()

`__shfl_up()` calculates a source lane ID by subtracting delta from the caller's thread ID. The value of `var` held by the resulting thread ID is returned; in effect, `var` is shifted up the warp by `delta` lanes.

The source lane index will not wrap around the value of `width`, so the lower `delta` lanes are unchanged.

```
integer(4) function __shfl_up(var, delta, width)
   integer(4) var, delta
   integer(4), optional :: width
```

```
real(4) function __shfl_up(var, delta, width)
   real(4) :: var
   integer(4) :: delta
   integer(4), optional :: width
```

```
real(8) function __shfl_up(var, delta, width)
   real(8) :: var
   integer(4) :: delta
   integer(4), optional :: width
```

__shfl_down()

`__shfl_down()` calculates a source lane ID by adding `delta` to the caller's thread ID. The value of `var` held by the resulting thread ID is returned: this has the effect of shifting `var` down the warp by `delta` lanes. The ID number of the source lane will not wrap around the value of width, so the upper `delta` lanes remain unchanged.

```
integer(4) function __shfl_down(var, delta, width)
   integer(4) var, delta
   integer(4), optional :: width
```

```
real(4) function __shfl_down(var, delta, width)
   real(4) :: var
   integer(4) :: delta
   integer(4), optional :: width
```

```
real(8) function __shfl_down(var, delta, width)
   real(8) :: var
   integer(4) :: delta
   integer(4), optional :: width
```

__shfl_xor()

`__shfl_xor()` uses ID-1 to calculate the source lane ID by performing a bitwise XOR of the caller's lane ID with the `laneMask`. The value of `var` held by the resulting lane ID is returned. If the resulting lane ID falls outside the range permitted by `width`, the thread's own value of `var` is returned. This mode implements a butterfly addressing pattern such as is used in tree reduction and broadcast.

```
integer(4) function __shfl_xor(var, laneMask, width)
   integer(4) var, laneMask
   integer(4), optional :: width
```

```
real(4) function __shfl_xor(var, laneMask, width)
   real(4) :: var
   integer(4) :: laneMask
   integer(4), optional :: width
```

```
real(8) function __shfl_xor(var, laneMask, width)
   real(8) :: var
   integer(4) :: laneMask
   integer(4), optional :: width
```

Here is an example using `__shfl_xor()` to compute the sum of each thread's variable contribution within a warp:

```
      j = . . .
      k = __shfl_xor(j,1);  j = j + k
      k = __shfl_xor(j,2);  j = j + k
      k = __shfl_xor(j,4);  j = j + k
      k = __shfl_xor(j,8);  j = j + k
      k = __shfl_xor(j,16); j = j + k
```

C++ Compiler

C++ and OpenACC

This release introduces the OpenACC directives to our C++ compilers, pgcpp and (Linux only) pgc++. There are limitations to the data that can appear in data constructs and compute regions:

- Variable-length arrays are not supported in OpenACC data clauses; VLAs are not part of the C++ standard.
- Variables of class type that require constructors and destructors do not behave properly when they appear in data clauses.
- Exceptions are not handled in compute regions.
- Any function call in a compute region must be inlined. This includes implicit functions such as for I/O operators, operators on class type, user-defined operators, STL functions, lambda operators, and so on.

C++ Compatibility

PGI 2013 C++ object code is incompatible with prior releases.

All C++ source files and libraries that were built with prior releases must be recompiled to link with PGI 2013 or higher object files.

C++11 Features

Starting in PGI 13.1, our C++ Compiler now utilizes EDG version 4.5. This update enables the following C++11 language features, when compiled with the `--c++11` option:

- A "right shift token" (>>) can be treated as two closing angle brackets.
- The `static_assert` construct is supported

- The friend class syntax is extended to allow nonclass types as well as class types expressed through a typedef or without an elaborated type name.
- Local and unnamed types (and types based on such types) can be used for template type arguments. They can also be used in the signatures of functions, if they appear there through template substitution.
- Mixed string literal concatenations are accepted.
- C99 preprocessor extensions are carried over.
- The C99-style _Pragma operator is supported.
- In function bodies, the reserved identifier __func__ refers to a predefined array containing a string representing the function's name.
- A trailing comma in the definition of an enumeration type is silently accepted.
- The type long long is accepted.
- An explicit instantiation directive may be prefixed with the extern keyword to suppress the instantiation of the specified entity.
- Export templates are disabled.
- The keyword `typename` followed by a qualified-id can appear outside a template declaration.
- The keyword `auto` can be used as a type specifier in the declaration of a variable or reference.
- Trailing return types are allowed in top-level function declarators.
- The keyword `decltype` is supported: It allows types to be described in terms of expressions.
- Changes in the constraints on the code points implied by universal character names (UCNs).
- Scoped enumeration types (defined with the keyword sequence `enum class`) and explicit underlying integer types for enumeration types are supported.
- Lambdas are supported.
- Rvalue references are supported.
- Functions can be "deleted".
- Special member functions can be explicitly "defaulted".
- Move constructors and move assignment operators are now generated as specified by the C++11 standard.
- Conversion functions can be marked `explicit` to indicate that they should only be considered for explicit conversions, and in certain contexts that require a boolean value (like the controlling expression for an `if`-statement).
- The operand of `sizeof`, `typeid`, or `decltype` can refer directly to a non-static data member of a class without using a member access expression.
- The keyword `nullptr` can be used as both a null pointer constant and a null pointer-to-member constant.
- Attributes delimited by double square brackets ([[...]]) are accepted in declarations. The standard attributes noreturn and carries_dependency are supported.

- The context-sensitive keyword `final` is accepted on class types (to indicate they cannot be derived from) and on virtual member functions (to indicate they cannot be overridden).
- The context-sensitive keyword `override` can be specified on virtual member functions to assert that they override a corresponding base class member.
- Alias and alias template declarations are supported.
- Variadic templates are supported.
- U-literals as well as the `char16_t` and `char32_t` keywords are supported.
- Many errors in expressions that arise during the substitution of template parameters in function templates are now treated as deduction failures rather than definite errors.
- Access checking of names used as base classes is done in the context of the class being defined.
- Inline namespaces are supported.
- Initializer lists are supported.
- The `noexcept` specifier and operator are supported.
- Range-based `for` loops are supported.

New or Modified Runtime Library Routines

PGI 2013 supports new runtime library routines associated with the PGI Accelerator compilers. For more information, refer to the ”Using an Accelerator” chapter of the *PGI Compiler User's Guide*.

Library Interfaces

PGI provides access to a number of libraries that export C interfaces by using Fortran modules. These libraries and functions are described in Chapter 8 of the *PGI Compiler User's Guide*.

Environment Modules

Note

This section is only applicable to *PGI CDK*

On Linux, if you use the Environment Modules package (e.g., the **module load** command), then PGI 2013 includes a script to set up the appropriate module files.

OS X Mountain Lion Support

If you upgraded to OS X Mountain Lion, it is best to update Xcode to 4.0 or later before installing the PGI compilers and tools. To update, follow these steps:

1. Go to Apple App store. Apple menu | App Store...
2. Search for "Xcode"
3. Click the "Install" button.
4. Once step 3 is complete, double click the "Install Xcode" icon in the Application folder and follow the directions on the screen.

Chapter 3. Distribution and Deployment

Once you have successfully built, debugged and tuned your application, you may want to distribute it to users who need to run it on a variety of systems. This chapter addresses how to effectively distribute applications built using PGI compilers and tools.

Application Deployment and Redistributables

Programs built with PGI compilers may depend on runtime 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.

PGI Redistributables

The PGI 2013 release includes these directories:

$PGI/linux86/13.10/REDIST
$PGI/linux86-64/13.10/REDIST
$PGI/osx86/13.10/REDIST
$PGI/win32/13.10/REDIST
$PGI/win64/13.10/REDIST

These directories contain all of the PGI Linux runtime library shared object files, Mac OS dynamic libraries, or Windows dynamically linked libraries that can be re-distributed by PGI 2013 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 2013 directory.

Linux Redistributables

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.
- Users have set LD_LIBRARY_PATH to use the relevant version of the PGI shared objects.

Microsoft Redistributables

The PGI products on Windows include Microsoft Open Tools. The Microsoft Open Tools directory contains a subdirectory named "redist". PGI 2013 licensees may redistribute the files contained in this directory in accordance with the terms of the PGI End-User License Agreement.

Microsoft supplies installation packages, `vcredist_x86.exe` and `vcredist_x64.exe`, containing these runtime files. These files are available in the `redist` directory.

Chapter 4. Troubleshooting Tips and Known Limitations

This chapter contains information about known limitations, documentation errors, and corrections.

For up-to-date information about the state of the current release, visit the frequently asked questions (FAQ) section on pgroup.com at: www.pgroup.com/support/index.htm

General Issues

Most issues in this section are related to specific uses of compiler options and suboptions.

- Object files created with prior releases of PGI compiler are incompatible with object files from PGI 2013 and should be recompiled.
- The `-i8` option can make programs incompatible with the ACML libraries; use of any INTEGER*8 array size argument can cause failures. Visit developer.amd.com to check for compatible libraries.
- 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`.
- 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.

Platform-specific Issues

Linux

The following are known issues on Linux:

- 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 of the PGI compilers and tools.

Apple OS X

The following are known issues on Apple OS X:

- On MacOS platform, the PGI 2013 compilers do not support static linking of binaries. For compatibility with future Apple updates, the compilers only support dynamic linking of binaries.
- Using `–Mprof=func` or `–Mprof=lines` is not supported.

Microsoft Windows

The following are known issues on Windows:

- For the Cygwin `emacs` editor to function properly, you must set the environment variable `CYGWIN` to the value "`tty`" before invoking the shell in which `emacs` will run. However, this setting is incompatible with the PGBDG command line interface (`-text`), so you are not able to use `pgdbg -text` in shells using this setting.

  The Cygwin team is working to resolve this issue.
- 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.
- C++ programs on Win64 that are compiled with the option `-tp x64` fail when using PGI Unified Binaries. The `-tp x64` switch is not yet supported on the Windows platform for C++.
- 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.

The following are known issues on Windows and *PGDBG*:

- In *PGDBG* on the Windows platform, Windows times out stepi/nexti operations when single stepping over blocked system calls. For more information on the workaround for this issue, refer to the online FAQs at www.pgroup.com/support/tools.htm.

The following are known issues on Windows and *PGPROF*:

- Do not use `-Mprof` with PGI 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.

PGDBG-related Issues

The following are known issues on *PGDBG*:

- Before *PGDBG* can set a breakpoint in code contained in a shared library, `.so` or `.dll`, the shared library must be loaded.
- Breakpoints in processes other than the process with rank 0 may be ignored when debugging MPICH-1 applications when the loading of shared libraries to randomized addresses is enabled.
- Debugging of PGI Unified Binaries, that is, 64-bit programs built with more than one `-tp` option, is not fully supported. The names of some subprograms are modified in the creation, 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 PGI Unified Binary, see www.pgroup.com/support/tools.htm.

PGPROF-related Issues

The following are known issues on *PGPROF*:

- Accelerator profiling via `pgcollect` is disabled in PGI 2013. PGI Accelerator and OpenACC programs profiled using `pgcollect` will not generate any performance data related to the GPU. This capability is expected to be restored in a future release.

  Workaround: Set the environment variable `PGI_ACC_TIME` to '1' for the program when it runs (not when compiling). This setting causes the program to print some performance data to `stdout` on exit.

  CUDA Fortran profiling is still available for CUDA Fortran programs.
- 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.
- Times reported for multi-threaded sample-based profiles, that is, profiling invoked with options `-pg` or `-Mprof=time`, are for the master thread only. To obtain profile data on individual threads, PGI-style instrumentation profiling with `-Mprof={lines | func}` or **pgcollect** must be used.

CUDA Fortran Toolkit Issues

The CUDA 5.0 Toolkit is set as the default in PGI 2013. To use the CUDA 5.0 Toolkit, first download the CUDA 5.0 driver from NVIDIA at www.nvidia.com/cuda.

You can compile with the CUDA 5.0 Toolkit either by adding the `-ta=nvidia:cuda5.0` option to the command line or by adding `set CUDAVERSION=5.0` to the `siterc` file.

`pgaccelinfo` prints the driver version as the first line of output. For a 4.2 driver, it prints:

```
CUDA Driver Version 5000
```

Corrections

A number of problems have been corrected in the PGI 2013 release. Refer to www.pgroup.com/support/release_tprs.htm for a complete and up-to-date table of technical problem reports, TPRs, fixed in recent releases of the PGI compilers and tools. This table contains a summary description of each problem as well as the version in which it was fixed.

Chapter 5. Contact Information

You can contact The Portland Group at:

The Portland Group
Two Centerpointe Drive, Suite 320
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:

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	www.pgroup.com

All technical support is by email or submissions using an online form at 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 www.pgroup.com/support/faq.htm.

PGI documentation is available at www.pgroup.com/resources/docs.htm or in your local copy of the documentation in the release directory doc/index.htm.

NOTICE

TRADEMARKS

COPYRIGHT