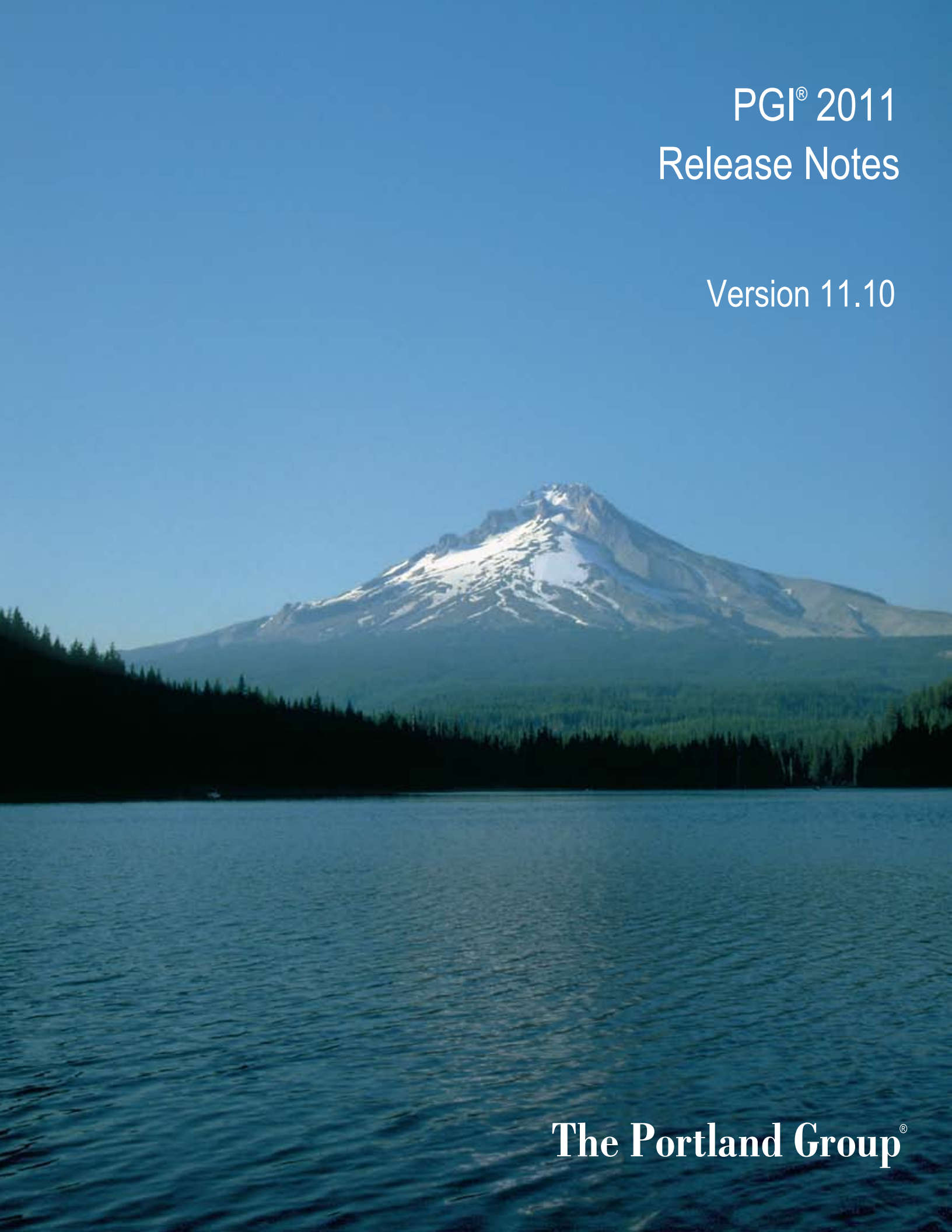
PGI® 2011 Release Notes

Version 11.10



The Portland Group®

PGI® 2011 Release Notes

Printed in the United States of America

First Printing: Release 2011, version 11.0, December 2010
Second Printing: Release 2011, version 11.1, January 2011
Third Printing: Release 2011, version 11.2, February 2011
Fourth Printing: Release 2011, version 11.3, March 2011
Fifth Printing: Release 2011, version 11.4, April 2011
Sixth Printing: Release 2011, version 11.5, May 2011
Seventh Printing: Release 2011, version 11.6, June 2011
Eighth Printing: Release 2011, version 11.7, July 2011
Ninth Printing: Release 2011, version 11.8, August 2011
Tenth Printing: Release 2011, version 11.9, September 2011
Eleventh Printing: Release 2011, version 11.10, October 2011

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

ID: 11291141

Contents

Tables

Chapter 1. Release Overview

Welcome to Release 2011 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 and Microsoft Windows operating systems. *PGI Workstation* and *PGI Server* are also available for the Apple Mac OS X operating system.

This document describes changes between previous versions of the PGI 2011 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 enables 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 ST 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, Mac 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 and Windows 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 2011 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.

  #### Note

  linux86-64 versions are limited.

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, and AMD Istanbul 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, Intel Core 2 (Penryn), Intel Core i3, i5, i7 (Nehalem) 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 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 Mac OS X* – supported on 64-bit Apple Mac operating systems running on a 64-bit Intel-based Mac system.

Supported Operating Systems

This section describes updates and changes to PGI 2011 that are specific to Linux, Windows, and Mac OS X.

Linux

Java Runtime Environment (JRE)

Although the PGI installation on Linux includes a 32-bit version of the Java Runtime Environment (JRE), sufficient 32-bit X Windows support must be available on the system for the JRE and the PGI software that depends on it to function properly. On some systems, notably recent releases of Fedora Core, these libraries are not part of the standard installation.

The required X Windows support generally includes these libraries:

```
libXau          libXdmcp          libxcb
libX11          libXext
```

Mac OS X

PGI 2011 for Mac 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 Mac OS X function identically to their Linux counterparts.

Windows

PGI 2011 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 2011 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 2011 of the PGI compilers and tools.

What's New in Release 2011

Most Recent Updates and Additions

- CUDA-x86 code generator now produces code that is optimized for the latest x64 platforms.
- Support is available for AMD's Bulldozer based processor. The option `-tp` now supports the suboption `bulldozer`.
- Hardware count-based profiling support for both Sandybridge and Bulldozer based processors is available. Use `pgcollect -hwtime`

11.8 Updates and Additions

- PGI 11.8 includes an update of Cygwin to version 1.7.9-1.
- PGI 11.8 includes support for the Mac OS X Lion operating system.

  #### Tip

  If you upgraded to Mac OS X Lion, it is best to update Xcode to 4.0 or later before installing the PGI compilers and tools. For more information, refer to"Mac OS X Lion Support," on page 22.

- **PGI Accelerator** compilers and **CUDA Fortran** include these modifications and enhancements:
  - CUDA Fortran supports the option `-mcmodel=medium`.
  - PGI provides a module which defines interfaces to the CUBLAS Library from PGI CUDA Fortran. These interfaces are made accessible by placing the following statement in the CUDA Fortran host-code program unit:

    ```
    use cublas
    ```

11.7 Updates and Additions

- Fortran 2003 now supports final subroutines that allow the programmer to specify subroutine(s) that get called when a variable is deallocated or ceases to exist (i.e., no longer in scope). This allows the program to clean up, perhaps release some resources, when the variable no longer exists.

  For more information, refer to "Object-Oriented Programming Enhancements," on page 21.

- **PGDBG** has a number of enhancements in this release:
  - New graphical presentation of stack trace information in PGDBG's Call Stack tab.
  - Enhanced support for command-line editing in PGDBG text mode and the PGDBG GUI's Command Prompt tab. Includes support for arrow and control keys and, in the GUI, text selection and replacement.

11.6 Updates and Additions

- The ALLOCATE statement now correctly accepts the `source=` qualifier for a polymorphic source type with allocatable members.
- Support is available for Intel's Sandy Bridge processor. The option `-tp` now supports the suboption `sandybridge`.
- The option `-Mvect` now supports the suboption `simd[:{128|256}]` which specifies to vectorize using SIMD instructions and data, either 128 bits or 256 bits wide, on processors where there is a choice.
- **PGI Accelerator** compilers and **CUDA Fortran** include these modifications and enhancements:
  - The default CUDA Toolkit version is now 3.2; the previous default was 3.1. To target a specific toolkit, refer to "PGI Accelerator and CUDA Fortran Enhancements," on page 13
  - Added support for the CUDA 4.0 Toolkit. Some features require this toolkit.

    When using the CUDA 4.0 Toolkit, **pgaccelinfo** displays the following line:

    ```
    CUDA Driver Version 4000
    ```

  - A `cublas` emulation library is now provided fro CUDA C++ for x86. For details on linking in this library, see the simple CUBLAS SDK example. A `cublas` module is also provided for PGI CUDA Fortran and PGI Accelerator Fortran use.
  - In CUDA Fortran, the print statement is allowed in attributes(global) and attributes(device) routines.

    ```
    print *,...
    ```

    The only items allowed on the `print*` statement are scalar integer, real, double precision, logical constants and variables, and character string constants.

    The output from multiple threads is interleaved. This issue may be considered in a future release.

  - When using `-ta=nvidia` in Fortran, if you have a call to a routine `SUB(x)` with an explicit interface, one or more dummy arguments can have the CUDA Fortran DEVICE attribute. If the actual argument has a visible device copy from the data region, or from a reflected or mirrored directive, the device copy is passed to the routine.

If SUB is an overloaded interface to two or more routines, one of which has a matching dummy argument with the CUDA Fortran DEVICE attribute and the other does not, the generic procedure matching process will prefer the routine with the argument that has the DEVICE attribute.

- In CUDA Fortran global routines, shared memory arrays may now have nonstatic size. The size may come from the blockdim members, from arguments or device variables. The launch of the kernel must specify enough space to hold all automatic shared memory arrays.

  Example:

```
attributes(global) subroutine foo(a, n)
 real,dimension(:) :: a
 integer, value :: n
 real,shared,dimension(blockdim%x) :: sa
...
end subroutine
```

11.5 Updates and Additions

- **PGI Accelerator** compilers and **CUDA Fortran** include these enhancements:
  - Two new flags are available that control flush-to-zero mode for floating point computations on GPU code:

    `-ta=nvidia,[no]flushz`
    Controls flush-to-zero mode for floating point computations on the GPU code generated for PGI Accelerator model compute regions. The default is `-ta=nvidia,noflushz`

    `-Mcuda=[no]flushz`
    Controls flush-to-zero mode for floating point computations on the GPU code generated for CUDA Fortran kernels. The default is `-Mcuda=noflushz`.

- PGI 11.5 contains the initial product release of the CUDA C++ compiler for x86.

  Developers can utilize the PGI C++ compiler to compile CUDA C++ code and then run it on an x86 target. For more information, refer to "PGI CUDA C++ Compilers for x86," on page 22.

- PGDBG supports debugging of CUDA-x86 programs.

11.4 Updates and Additions

- Additional Fortran modules are now available, including:
  - A Fortran module to declare interfaces to many of the routines in the standard C math library `libm`.
  - A Fortran module to declare interfaces to many of the CUDA device builtin routines.

  For more information on these modules, refer to "Fortran Modules," on page 13.

- **PGI Accelerator** and **CUDA Fortran** include these enhancements:
  - The PGI Accelerator model has the routines `acc_malloc` and `acc_free` that directly allocate and free memory in C programs on the GPU device.
  - New API routines access minimal profiling information for accelerator regions and kernels.

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

For more information on these enhancements, refer to "PGI Accelerator and CUDA Fortran Enhancements," on page 13.

- **PGPROF** has a number of enhancements in this release:
  - Reorganized menu items are now more closely aligned with standard menu functionality.
  - User preference settings are now saved on a per-user basis on Linux and Mac, and a per-user per-system basis on Windows.
  - User preference settings are now saved on exit by default.
  - Changing font size is now fully supported in all tabs. Changing font type is supported in all tabs except Compiler Feedback.
- **PGDBG** has a number of enhancements in this release:
  - All tabs in the debugger GUI are now fully dockable. Rearrange, tear-off, resize, close, and auto-hide all source and debug information windows.
  - User preference settings are now saved on a per-user basis on Linux and Mac, and a per-user per-system basis on Windows.
  - Changing font type and size is now fully supported in all tabs.

11.3 Updates and Additions

- The *PGPROF* option `-text` is re-enabled in this release.
- Accelerator and CUDA Fortran now support 64-bit OSX.

11.2 Updates and Additions

- PGI 11.2 includes an update of Cygwin to version 1.7.7-1. It includes new-to-pgi-workstation features like `ssh`, `emacs` and `perl`. The updated version should also provide better compatibility with Win7 and other recent versions of Windows.

Updates and Additions prior to 11.2

- A new option `-nomp` is available which disables the `-mp` option that is enabled by default for linking.
- Behavior of the `-tp` flags and `-m32/-m64` flags without a bit-length suffix. These flags now use the current bit width rather than defaulting to 32-bit.
- Enhanced error checks for mixing 32-bit and 64-bit targets, multiple 32-bit targets, and whether the 32-bit or 64-bit compilers are not installed.
- pgCC now uses low cost exception handling by default (`--zc_eh`). As a result, all user code must be recompiled. To return to the old setjmp and longjmp exception handling, use the `--nozc_eh` flag.

- Added support for PTXAS informational messages. Use `-Mcuda=ptxinfo` to show PTXAS informational messages during compilation.
- Added support for the CUDA 3.2 Toolkit.
- The *PGPROF* option `-text` has been disabled in this release. It may be re-enabled in a future release.

Fortran 2003 Features

Generic type-bound procedures
: Allow you to define a type-bound procedure to be generic by defining a generic statement within the type-bound procedure part. The statement is of the form:

```
generic [[ , access-spec ] ::] generic-spec => tbp-name-list
```

: where *tbp-name-list* is a list of the specific type-bound procedures to be included in the generic set. You can use these statements for named generics as well as for operators and assignments.

PROTECTED statement and attribute
: Protects a module variable against modification from outside the module. The PROTECTED attribute may only be specified for a variable in a module. This statement and attribute allow values of variables to be available without relinquishing control over their modification.

Associate Construct
: Associates a name either with a variable or with the value of an expression for the duration of a block. When the block is executed, the `associate-name` remains associated with the variable or retains the value specified, taking its type, type parameters, and rank from the association. For more information and an example, refer to the PGI Compiler User's Guide.

Sourced Allocation
: For deferred character and polymorphic objects, uses the `source=` clause in the `allocate` statement to take the type, type parameters, and values from another variable or expression. The allocated variable has the same dynamic type, parameters, and value as the source variable, essentially producing a "clone" of the source variable or expression.

```
subroutine example(x)
   class(t), allocate ::a
   class(t)           :: x
   allocate(a, source=x)
```

SELECT TYPE construct
: Provides the capability to execute alternative code depending on the dynamic type of a polymorphic entity, and to gain access to dynamic parts. Like the SELECT CASE construct, the code consists of a number of blocks and at most one is selected for execution.

Generic and Derived Types the same
: Allows generic procedure names to be the same as the type name. If ambiguity occurs, the generic name overrides the type name.

Unlimited polymorphic entities
: Allow the user to have an object that may refer to objects of any type, including non-extensible or intrinsic types. Unlimited polymorphic entities can only be used as an actual argument, as the pointer or target in a pointer assignment, or as the selector in a SELECT TYPE statement.

Deferred Character Length

A `len` type parameter value may be a colon in a type declaration statement, indicating that the type parameter has no defined value until it is given one by allocation or pointer assignment. For intrinsic types, only character length may be deferred.

Example:

```
character(len=:), pointer :: varchar
```

ABSTRACT types

Allows the programmer to create types that must be extended and further implemented before they can be instantiated.

Objects of insufficient type cannot be created. When `abstract` is specified, the compiler warns the user if any inherited type-bound procedure has not been overridden.

Use of `inf` and `NaN`s.

In Fortran 2003, input and output of IEEE infinities and NaNs is specified. All edit descriptors for reals treat these values in the same way; only the field width is required.

SIGN= specifier

Provides the ability to control the optional plus characters in formatted numeric output.

```
sign = [“suppress” | “plus” | “processor_defined” | “undefined” ]
```

“suppress” indicates to suppress the plus characters; “plus” indicates to show the plus characters; “processor_defined” indicates that the processor defines whether the plus characters are shown or hidden.

This specifier is available on the OPEN, WRITE, and INQUIRE statements.The “undefined” specifier is only available in the INQUIRE statement.

Use in a WRITE statement overrides a SIGN=specifier in an OPEN statement. The WRITE statement may also change the mode through use of the Fortran 95 edit descriptors: `SS`, `SP`, and `S`.

Round specifier

Ability to specify rounding during formatted input and output. Support for this feature is through use of the `ROUND=specifier` clause or through use of the `RU`, `RD`, `RZ`, `RN`, `RC`, and `RP` edit descriptors.

- The ROUND=*specifier* clause is available for OPEN and INQUIRE statements.

  In the OPEN statement *specifier* is one of: “`up`”, “`down`,” “`zero`”, “`nearest`”, “`compatible`”, or “`processor_defined`”. Both “`nearest`” and “`compatible`” refer to closest representable value. If these are equidistant, then it is processor-dependent for “`nearest`“and the value away from zero for “`compatible`.”

  In the INQUIRE statement, the processor returns one of these values: “`up`”, “`down`,” “`zero`”, “`nearest`”, “`compatible`”, “`processor_defined`”, or “`undefined`”, as appropriate. The processor returns “`processor_defined`” only if the rounding mode currently in effect behaves differently from other rounding modes.

- The RU, RD, RZ, RN, RC, and RP edit descriptors represent "up", "down," "zero", "nearest", "compatible" or "processor_defined" rounding modes, respectively. These modes are valid in format processing, such as in a READ or WRITE statement. The specific rounding mode takes effect immediately when encountered, and stays in effect until either another descriptor is encountered or until the end of the READ and WRITE statement.

Fortran Modules

PGI 2011 includes additional Fortran modules:

- A Fortran Module is available to declare interfaces to many of the routines in the standard C math library libm. To access this module, add the following to your Fortran program:

```
use libm
```

  Some libm routine names conflict with Fortran intrinsics. The following routines resolve to Fortran intrinsics.

asin	acos	atan2	cos	cosh
exp	log	log10	sin	sinh
sqrt	tan	tanh		

  You can also use libm routines in CUDA Fortran global and device subprograms, in CUF kernels, and in PGI Accelerator compute regions. When targeting NVIDIA devices, the libm routines translate to the corresponding libm device routine.

  For a complete list of routines that are available, refer to the *PGI Compiler Reference Manual*.

- As described in the following section, a Fortran module is available to declare interfaces to many of the CUDA device built-in routines.

PGI Accelerator and CUDA Fortran Enhancements

PGI Accelerator x64+GPU native Fortran 2003 and C99 compilers and **CUDA Fortran** now support the CUDA 4.0 Toolkit. PGI continues to ship CUDA 3.2 Toolkit and CUDA 3.1 Toolkit with the PGI compilers and tools, and you may leave it on your system if it exists from a previous installation.

The default toolkit for 11.10 is CUDA 3.2. Thus, CUDA Fortran host programs should be recompiled, as the PGI 11.10 CUDA Fortran runtime libraries are not compatible with previous CUDA Fortran releases.

To specify the version of the **CUDA Toolkit** that is targeted by the compilers, use one of the following options:

In PGI Accelerator:

For CUDA Toolkit 4.0

```
-ta=nvidia:cuda4.0 or -ta=nvidia:4.0
```

For CUDA Toolkit 3.2

```
-ta=nvidia:cuda3.2 or -ta=nvidia:3.2
```

For CUDA Toolkit 3.1

```
-ta=nvidia:cuda3.1 or -ta=nvidia:3.1
```

For CUDA Fortran:

For CUDA Toolkit 4.0

```
-Mcuda=cuda4.0  or  -Mcuda=4.0
```

For CUDA Toolkit 3.2

```
-Mcuda=cuda3.2  or  -Mcuda=3.2
```

For CUDA Toolkit 3.1

```
-Mcuda=cuda3.1  or  -Mcuda=3.1
```

You can also specify a specific 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 4.0, add the following line to one of these files:

```
set DEFCUDAVERSION=4.0;
```

Additional PGI Accelerator Runtime Routines

Table 2.1 contains a list of the PGI Accelerator model runtime routines available in Release 11.10. For a complete description of these routines, refer to Chapter 4, "PGI Accelerator Compilers Reference" of the *PGI Compiler Reference Manual*.

Table 2.1. Accelerator Runtime Library Routines

This Runtime Library Routine...	Does this...
acc_allocs	Returns the number of arrays allocated in data or compute regions.
acc_bytesalloc	Returns the total bytes allocated by data or compute regions.
acc_bytesin	Returns the total bytes copied in to the accelerator by data or compute regions.
acc_bytesout	Returns the total bytes copied out from the accelerator by data or compute regions.
acc_copyins	Returns the number of arrays copied in to the accelerator by data or compute regions.
acc_copyouts	Returns the number of arrays copied out from the accelerator by data or compute regions.
acc_disable_time	Tells the runtime to start profiling accelerator regions and kernels, if it is not already doing so.
acc_enable_time	Tells the runtime to start profiling accelerator regions and kernels, if it is not already doing so.
acc_exec_time	Returns the number of microseconds spent on the accelerator executing kernels.
acc_free	Frees memory allocated on the attached accelerator. [C only]
acc_frees	Returns the number of arrays freed or deallocated in data or compute regions.

This Runtime Library Routine...	Does this...
acc_get_device	Returns the type of accelerator device used to run the next accelerator region, if one is selected.
acc_get_device_num	Returns the number of the device being used to execute an accelerator region.
acc_get_free_memory	Returns the total available free memory on the attached accelerator device.
acc_get_memory	Returns the total memory on the attached accelerator device.
acc_get_num_devices	Returns the number of accelerator devices of the given type attached to the host.
acc_init	Connects to and initializes the accelerator device and allocates control structures in the accelerator library.
acc_kernels	Returns the number of accelerator kernels launched since the start of the program.
acc_malloc	Allocates memory on the attached accelerator. [C only]
acc_on_device	Tells the program whether it is executing on a particular device.
acc_regions	Returns the number of accelerator regions entered since the start of the program.
acc_set_device	Tells the runtime which type of device to use when executing an accelerator compute region.
acc_set_device_num	Tells the runtime which device of the given type to use among those that are attached.
acc_shutdown	Tells the runtime to shutdown the connection to the given accelerator device, and free up any runtime resources.
acc_total_time	Returns the number of microseconds spent in accelerator compute regions and in moving data for accelerator data regions.

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 Builtin Routines

A Fortran module is available to declare interfaces to many of the CUDA device builtin 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 builtin routines that are available, refer to the *PGI CUDA Fortran Programming and Reference*.

New or Modified Compiler Options

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.

The following compiler options have been added or modified in PGI 2011:

- The option `-tp` supports additional suboptions. To enable the option, add `-tp=` with the corresponding suboption to the compilation line.

bulldozer	Generate either 32-bit or 64-bit code for AMD Bulldozer and compatible processors. 32- or 64-bit depends on the driver on your path.
bulldozer-32	Generate 32-bit code for AMD Bulldozer and compatible processors.
bulldozer-64	Generate 64-bit code for AMD Bulldozer and compatible processors.
sandybridge	Generate either 32-bit or 64-bit code for Intel Sandy Bridge and compatible processors. 32- or 64-bit depends on the driver on your path.
sandybridge-32	Generate 32-bit code for Intel Sandy Bridge and compatible processors.
sandybridge-64	Generate 64-bit code for Intel Sandy Bridge and compatible processors.

- The option `-Mvect` now supports the suboption `simd[:{128|256}]` which specifies to vectorize using SIMD instructions and data, either 128 bits or 256 bits wide, on processors where there is a choice.
- The `-mp` option is now enabled by default for linking. To disable this option, use the `-nomp` option when linking.
- Behavior of the `-tp` flags and `-m32/-m64` flags without a bit-length suffix. These flags now use the current bit width rather than defaulting to 32-bit.

  For example, if you have the 32-bit driver on your path, then `-tp shanghai` defaults to 32-bit; while if you are using the 64-bit driver, it defaults to 64-bit. For example, `-tp p7 -m64`, `-m64 -tp p7`, `-tp p7-64`, `-m64 -tp p7-64`, and `-tp p7-64 -m64` all mean the same thing.

- `–ta=nvidia(,nvidia_suboptions),host` is a switch associated with the PGI Accelerator compilers. `–ta` defines the target architecture.

  In release 2011, the `nvidia_suboptions` include:

analysis	Perform loop analysis only; do not generate GPU code.

cc10, cc11, cc12, cc13, cc20	Generate code for compute capability 1.0, 1.1, 1.2, 1.3, or 2.0 respectively.
cuda2.3 or 2.3	Specify the CUDA 2.3 version of the toolkit.
cuda3.0 or 3.0	Specify the CUDA 3.0 version of the toolkit.
cuda3.1 or 3.1	Specify the CUDA 3.1 version of the toolkit.
cuda3.2 or 3.2	Specify the CUDA 3.2 version of the toolkit.
cuda4.0 or 4.0	Specify the CUDA 4.0 version of the toolkit.
fastmath	Use routines from the fast math library.
[no]flushz	[Do not] Enable flush-to-zero mode for floating point computations on the GPU code generated for PGI Accelerator model compute regions.
keepbin	Keep the binary (.bin) files.
keepgpu	Keep the kernel source (.gpu) files.
keepptx	Keep the portable assembly (.ptx) file for the GPU code.
maxregcount:n	Specify the maximum number of registers to use on the GPU. Leaving this blank indicates no limit.
mul24	Use 24-bit multiplication for subscripting.
nofma	Do not generate fused multiply-add instructions.
time	Link in a limited-profiling library.
[no]wait	[Do not] Wait for each kernel to finish before continuing in the host program. The default is wait.

- The option `-Mcuda` tells the compiler to enable CUDA Fortran. In Release 2011, `-Mcuda` has these suboptions:

cc10, cc11, cc12, cc13, cc20	Generate code for compute capability 1.0, 1.1, 1.2, 1.3, or 2.0 respectively.
cuda2.3 or 2.3	Specify the CUDA 2.3 version of the toolkit.
cuda3.0 or 3.0	Specify the CUDA 3.0 version of the toolkit.
cuda3.1 or 3.1	Specify the CUDA 3.1 version of the toolkit.
cuda3.2 or 3.2	Specify the CUDA 3.2 version of the toolkit.
cuda4.0 or 4.0	Specify the CUDA 4.0 version of the toolkit.
emu	Enable CUDA Fortran emulation mode.
fastmath	Use routines from the fast math library.
[no]flushz	[Do not] Enable flush-to-zero mode for floating point computations on the GPU code generated for CUDA Fortran kernels.
keepbin	Keep the generated binary (.bin) file for CUDA Fortran.
keepgpu	Keep the generated GPU code (.gpu) for CUDA Fortran.
keepptx	Keep the portable assembly (.ptx) file for the GPU code.

maxregcount:n	Specify the maximum number of registers to use on the GPU. Leaving this blank indicates no limit.
nofma	Do not generate fused multiply-add instructions.
ptxinfo	Show PTXAS informational messages during compilation.

C++ Compatibility

PGI 2011 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 2011 or higher object files.

New or Modified Runtime Library Routines

PGI 2011 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*.

New or Modified Tools Support

This section describes the improvements to the PGI profiler *PGPROF* and the PGI debugger *PGDBG*.

PGPROF

PGI 2011 includes several new and enhanced performance profiling features:

profiling of PGI Accelerator model programs
: PGCOLLECT automatically captures high-level GPU performance statistics from PGI Accelerator model programs, and *PGPROF* displays them in combination with host program performance data.

profiling of CUDA Fortran programs
: PGCOLLECT supports a command-line option, `-cuda`, to enable collection of performance data from CUDA Fortran programs. *PGPROF* displays such data in combination with host program performance data.

PGPROF user interface
: The *PGPROF* user interface has a new look and feel. Menus are reorganized and a new tree-structured view of parallel performance data is implemented.

CCFF
: *PGPROF* now provides expanded compiler feedback explanations and optimization hints using PGI's Common Compiler Feedback Format for more targeted assistance in optimizing code.

PAPI support discontinued
: Support for collection of performance data using PAPI has been discontinued in PGI 2011. This means that the compiler option `-Mprof=hwcts` is no longer supported.

PGDBG

PGI 2011 includes several new and enhanced performance debugging features:

CUDA-x86 programs
: Support is available for debugging CUDA-x86 programs.

Updated GUI
: The *PGDBG* graphical user interface is updated for PGI 2011. Buttons are simpler, menus are reorganized, and additional debug information features (i.e., Call Stack, Locals, Registers) are now top-level tabs.

Improved register display - CLI
: *PGDBG*'s command-line interface for displaying registers is expanded in PGI 2011. Use the new command **regs -info** to determine the available register groups and the formatting options available for each. Other new options to the regs command include `-grp`, `-fmt` and `-mode`.

Improved register display - GUI
: *PGDBG*'s new Registers tab displays registers using a table layout. Registers are displayed per-process and by category (i.e., General Purpose, XMM). Register values that update from one location to the next are highlighted in yellow.

64-bit Java Runtime Environment requirement
: With PGI 2011, *PGDBG* requires the 64-bit Java Runtime Environment (JRE) on 64-bit systems. The 32-bit JRE is still required on 32-bit systems. PGI products install the required versions of the JRE as needed, except on Mac OS X, where the JRE provided by Apple is used.

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 2011 includes a script to set up the appropriate module files.

Fortran 2003 Support Overview

This section provides an overview of all the F2003 features that PGI 2011 supports. Complete descriptions of these features are available in the *PGI Fortran Reference*.

I/O Support

These I/O capabilities are available in F2003:

- New specifier enhancements include:
  - SIGN= Specifier
  - NEXTREC, NUMBER, RECL, SIZE of all integer kinds
  - Round I/O specifier through the ROUND=specifier clause or through use of `RU`, `RD`, `RZ`, `RN`, `RC`, and `RP` edit descriptors.
  - SIZE of any integer kind in read/write statements

    - IOSTAT kind in all I/O statements
    - New specifier in WRITE statement: DELIMITER
    - New specifiers in READ statement: BLANK and PAD
    - New specifiers in INQUIRE statement: DECIMAL, POS, PENDING
- New I/O-related intrinsics include:
    - NEW_LINE
    - IS_IOSTAT_END
    - IS_IOSTAT_EOR
- Miscellaneous I/O capabilities include:
    - New decimal comma specifier descriptors, such as `DC` and `DP`
    - Asynchronous I/O
    - IMPORT and WAIT statements
    - ASYNCHRONOUS attribute and statement
    - I/O of Inf and NaN
    - I/O keyword encoding
    - Length of names and statements enhanced
    - Error message (ERRMSG) on ALLOCATE and DEALLOCATE
    - STOP statement warns about FP execution
    - Access = 'stream'
- Non-default derived type I/O

Data-related Enhancements

These data-related enhancements are available in F2003:

- allocatable scalars
- Generic and derived type may have the same name
- ABSTRACT types and interfaces
- VOLATILE attribute and statement
- MAX and MIN intrinsics take character parameter
- Structure and array Constructors
- Mixed component accessibility
- Deferred character length and deferred type parameters

- Typed allocation, including typed allocation for unlimited polymorphic entities
- Sourced allocation for deferred character, non-polymorphic, polymorphic types, and unlimited polymorphic entities
- Procedure pointers
- IEEE modules including IEEE_ARITHMETIC, IEEE_EXCEPTIONS, and IEEE_FEATURES. The IEEE_ARITHMETIC module allows operations on large arrays.
- IMPORT statement
- PROTECTED attribute and statement
- move_alloc() function
- Rename user-defined operators
- Pointer reshaping
- Square brackets

Object-Oriented Programming Enhancements

These object-oriented programming enhancements are available in F2003:

- Classes and inheritance association
- New intrinsics including EXTENDS_TYPE_OF and SAME_TYPE_AS
- Attributes including PASS, NOPASS, PRIVATE, PUBLIC and NON_OVERRIDABLE
- ASSOCIATE and SELECT TYPE constructs, including the SELECT TYPE construct for unlimited polymorphic entities
- Type-bound, deferred type-bound procedures and generic type-bound procedures
- Type extension
- Polymorphic entities and unlimited polymorphic entities
- Parameterized derived types
- ABSTRACT types
- PRIVATE statement for type bound procedures
- Type uses CONTAINS declaration
- Final Subroutines

  Final subroutines allow the programmer to specify subroutine(s) that get called when a variable is deallocated or ceases to exist (i.e., no longer in scope). This allows the program to clean up, perhaps release some resources, when the variable no longer exists.

  Final subroutines are only available in derived types and must be module procedures with a single dummy argument. These dummy arguments shall be nonoptional, nonpointer, nonallocatable, nonpolymorphic,

and of the same type as the derived type being defined. Moreover, the dummy argument shall not have the `INTENT(OUT)` or `VALUE` attribute. All length type parameters must be assumed, that is, the length value must be '*'.

Final subroutines are always "`accessible`", never "`private`". They are not inherited through type extension and cannot be overridden. If a type extension and its parent type both have final subroutines, then the program calls the type extension's final subroutines before calling the parent's final subroutines.

Interoperability with C Enhancements

These features for interoperability with C are available in F2003:

- Subroutines c_associated, c_f_pointer, and c_f_procpointer
- Enumerators
- Modules ISO_C_BINDING and ISO_FORTRAN_ENV

Miscellaneous F2003 Enhancements

In addition to the enhancements related in the previous sections, these features are available in F2003:

- Optional KIND to intrinsics
- SYSTEM_CLOCK count_rate can be type `real`
- INTERFACE statements
- NAMELIST with internal file and group entities

Mac OS X Lion Support

If you upgraded to Mac OS X 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-screen.

PGI CUDA C++ Compilers for x86

PGI 11.5 contains the initial product release of the CUDA C++ compiler for x86. Developers can utilize the PGI C++ compiler to compile CUDA C/C++ code and then run it on an x86 target.

Compiler Options

Certain options may be useful when targeting your CUDA build for x86.

`-Mcudax86`
Add this option on the PGI C++ command line. If the file extension is `.cu`, then this option may not be required. In addition to enabling recognition of CUDA syntax, this option pulls the required libraries into the link process.

`--no_using_std`
Use this option to disable implicit use of the standard namespace in C++. This option is important for consistent behavior between pgCC and g++.

Sample Code

Here is an example of building and running the CUDA SDK MonteCarlo example on Linux:

```
% pgcpp --no_using_std -Mcudax86 -DUNIX -O2 \
        -I. -I../../common/inc -I../../shared/inc \
        MonteCarlo.cpp MonteCarlo_gold.cpp MonteCarlo_SM10.cu MonteCarlo_SM13.cu \
        -L../../lib -L../../common/lib/linux -L../../../shared/lib \
        -lcutil_x86_64 -lshrutil_x86_64
```

```
% ./a.out
[Monte Carlo]
Generating input data...
Allocating memory...
Generating normally distributed samples...
Running GPU Monte Carlo...
Options         : 256
Simulation paths: 262144
Time (ms.)      : 524.278015
GPU options per sec.: 488.290549
GPU Monte Carlo vs. Black-Scholes statistics
L1 norm        : 2.971674E-06
Average reserve: 387.263539
CPU Monte Carlo vs. Black-Scholes statistics...
L1 norm: 2.970427E-06
Average reserve: 386.847322
CPU vs. GPU Monte Carlo statistics...
L1 norm: 3.964267E-08
[Monte Carlo] - Test summary
PASSED
```

Debugging CUDA-x86 Applications with PGDBG

Developers can use PGDBG to debug their CUDA device code. When setting breakpoints in device code, `OMP_NUM_THREADS` threads, each running one task in emulation of a CUDA thread, hits the breakpoint in parallel.

Unsupported Features

Some features of CUDA C++ for GPUs are not supported in this release.

- Warp-synchronous programming.

  There is no current plan to support this feature. Many CUDA SDK examples use warp-synchronous programming techniques, for example in reductions; and these examples need to be rewritten for CUDA x86. For example, the reduction in the MonteCarlo code should be rewritten like this:

```
template<class T, unsigned int blockSize>
__device__ void sumReduceSharedMem(volatile T *sum, volatile T *sum2, int tid)
{
    // do reduction in portable, non-warp-synchronous manner
    for(unsigned int s=256; s>0; s>>=1)
    {
        if (blockSize >= (s+s)) {
            if (tid < s) {
                sum[tid] += sum[tid+s]; sum2[tid] += sum2[tid+s];
            }
            __syncthreads();
        }
    }
}
```

- Textures, and API calls in support of textures such as the CUDA Array functions. Support is planned for this feature in an upcoming release.
- The CUDA driver-level API. This feature may be supported in a future release.
- OpenGL interoperability. This feature may be supported in a future release.

Optimizing Code Generator

In this initial release, the optimizing code generator for CUDA device code is not enabled. Each CUDA thread is emulated on the x86 using an OpenMP task. The tasks run balanced across the number of threads specified by the OMP_NUM_THREADS environment variable, or by default, the number of cores on the system as determined by the `-mp=allcores` compiler flag support. PGI plans to release an update with an Optimizing Code Generator late in 2011.

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 2011 release includes these directories:

$PGI/linux86/11.10/REDIST
$PGI/linux86-64/11.10/REDIST
$PGI/osx86/11.10/REDIST
$PGI/win32/11.10/REDIST
$PGI/win64/11.10/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 2011 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 2011 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 2011 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 2011 and should be recompiled.
- The `-i8` option can make programs incompatible with the bundled ACML library, MPI, and ACML; use of any INTEGER*8 array size argument can cause failures with these libraries. 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:

- If you experience poor performance in the *PGDBG* or *PGPROF* GUI, try upgrading the X library `libxcb` to the latest version. The version number varies depending on your distribution. You can obtain a patch from your Linux distributor.

- 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 Mac OS X

The following are known issues on Mac OS X:

- On MacOS platform, the PGI 2011 compilers do not support static linking of user binaries. For compatibility with future Apple updates, the compilers only support dynamic linking of user binaries.
- Using `–Mprof=func` or `–Mprof=lines` is not supported.
- To begin an OpenMPI debugging session with *PGDBG* on Mac OS X 10.5 Leopard or later, use the following steps:

  1. Invoke the debugger using the full pathname of the executable. For example, you might use a command similar to this one:

     ```
     pgdbg -mpi:mpirun -np 4 /home/user1/a.out
     ```

  2. Set a breakpoint on main.
  3. Continue to the breakpoint.
  4. Begin your debugging session.

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 unified binary. 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, 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.

- 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.
- Due to problems in *PGDBG* in shared library load recognition on Fedora Core 6 or RHEL5, 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*:

- 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

Note

The CUDA 3.1 Toolkit is set as the default in PGI 2011.

- To use the CUDA 3.2 Toolkit, first download the CUDA 3.2 driver from NVIDIA at www.nvidia.com/cuda.

  You can compile with the CUDA 3.2 Toolkit either by adding the `-ta=nvidia:cuda3.2` option to the command line or by adding `set CUDAVERSION=3.2` to the `siterc` file.

- To use the CUDA 4.0 Toolkit, first download the CUDA 4.0 driver from NVIDIA at www.nvidia.com/cuda.

  You can compile with the CUDA 4.0 Toolkit either by adding the `-ta=nvidia:cuda4.0` option to the command line or by adding `set CUDAVERSION=4.0` to the `siterc` file.

`pgaccelinfo` prints the driver version as the first line of output.

For a 3.1 driver: `CUDA Driver Version 3010`
For a 3.2 driver: `CUDA Driver Version 3020`
For a 4.0 driver: `CUDA Driver Version 4000`

Corrections

A number of problems have been corrected in the PGI 2011 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
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:

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.