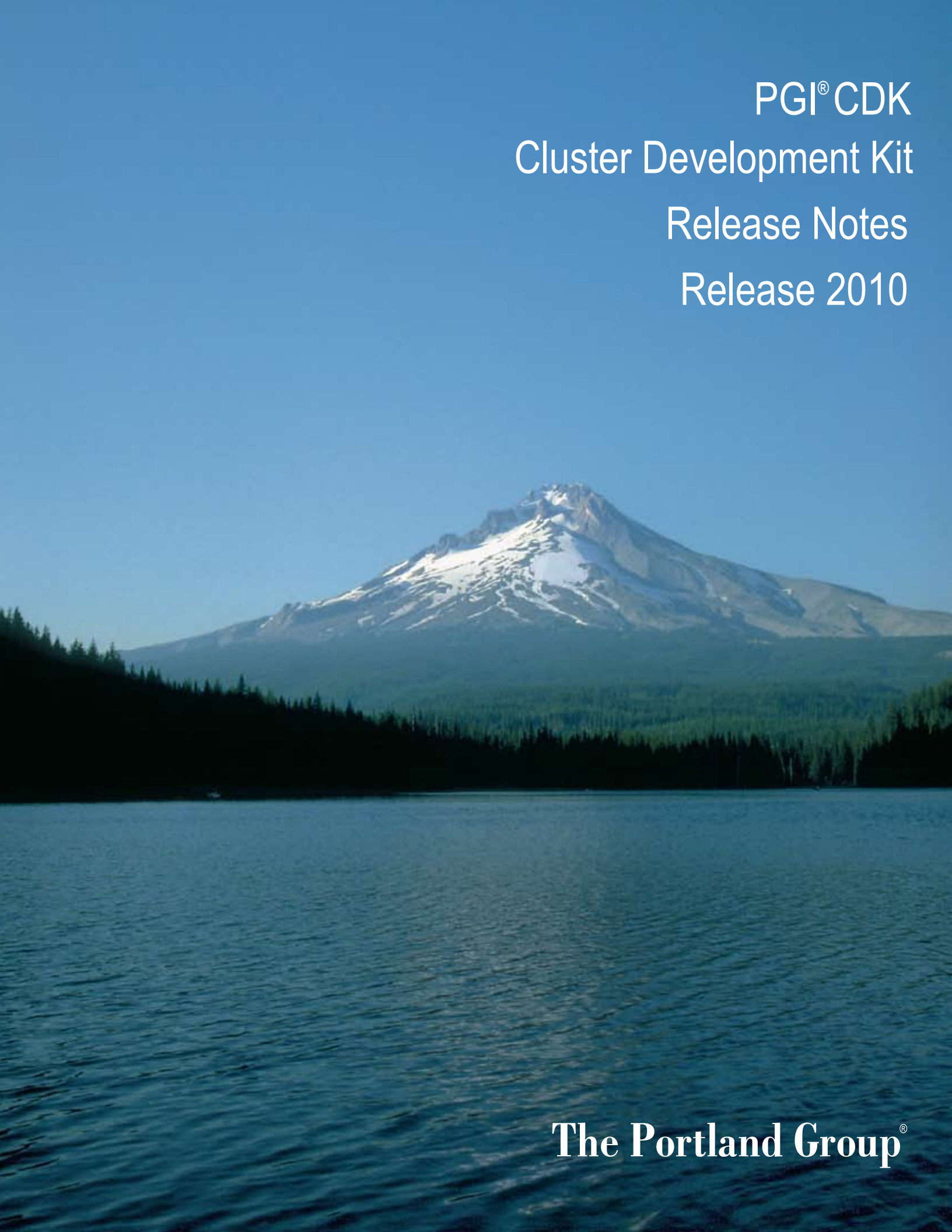

PGI® CDK
Cluster Development Kit
Release Notes
Release 2010

The Portland Group®

PGI CDK® Release Notes

Printed in the United States of America

First Printing: Release 2010, version 10.0, November 2009
Second Printing: Release 2010, version 10.1, January 2010
Third Printing: Release 2010, version 10.2, February 2010
Fourth Printing: Release 2010, version 10.3, March 2010
Fifth Printing: Release 2010, version 10.4, April 2010
Sixth Printing: Release 2010, version 10.5, May 2010
Seventh Printing: Release 2010, version 10.6, June 2010
Eighth Printing: Release 2010, version 10.8, August 2010
Ninth Printing: Release 2010, version 10.9, September 2010

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

ID: 10250112

Contents

1. Release Overview 1
Product Overview 1
Terms and Definitions 2
Supported Platforms 3

2. New or Modified Features 5
What's New in CDK Release 2010 5
10.9 Additions 5
10.8 Additions 5
10.6 Additions 6
10.5 Additions 7
10.4 Additions 8
10.3 Additions 9
2010 Additions Prior to 10.3 9
Getting Started 10
Using -fast, -fastsse, and Other Performance-Enhancing Options 11
New or Modified Compiler Options 11
C++ Compilation Requirements 13
Fortran Enhancements 13
Enhanced Fortran Interoperability with C 13
New or Modified Fortran Statements 14
New or Modified Fortran Intrinsic Functions 15
New Fortran Intrinsic Modules 15
Fortran I/O Enhancements 18
New or Modified Tools Support 19
Library Interfaces 19
Environment Modules 19

3. PGI Accelerator 21
Components 21
Availability 21
User-directed Accelerator Programming 21

Features Not Covered or Implemented 22
System Requirements 22
Supported Processors and GPUs 22
Installation and Licensing 22
Running an Accelerator Program 22
PGI Accelerator Compilers Runtime Libraries 22
Environment Variables 23
Applicable Command Line Options 24
PGI Unified Binary for Accelerators 25
Profiling Accelerator Kernels 26
Supported Intrinsics 26

4. Distribution and Deployment 27
Application Deployment and Redistributables 27
PGI Redistributables 27
Linux Redistributables 27

5. The PGI Windows CDK 29
Build MPI Applications with MSMPI 29
Using MSMPI libraries 29
Generate MPI Profile Data 29
Debug MSMPI Applications with PGDBG 30
Bash Shell Example 30
DOS Shell Example 31

6. Troubleshooting Tips and Known Limitations 33
General Issues 33
Platform-specific Issues 34
Linux 34
PGDBG-related Issues 34
PGPROF-related Issues 34
CUDA Fortran Toolkit Issues 34
Corrections 35

7. Contact Information 37

Chapter 1. Release Overview

Welcome to Release 2010 of *PGI Cluster Development Kit,* or *PGI CDK*, a set of Fortran, C, and C++ compilers and development tools for 32-bit and 64-bit x86-compatible processor-based workstations and servers running versions of the Linux operating systems.

A cluster is a collection of compatible computers connected by a network. The PGI CDK Cluster Development Kit 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.

This document describes changes between Release 2010 and previous releases of the PGI CDK, as well as late-breaking information not included in the current printing of the PGI User's Guide. There are two platforms supported by the PGI CDK compilers and tools:

- 32-bit Linux - supported on 32-bit Linux operating systems running on either a 32-bit x86 compatible or an x64 compatible processor.
- 64-bit/32-bit Linux - includes all features and capabilities of the 32-bit Linux version, and is also supported on 64-bit Linux operating systems running an x64 compatible processor.

These versions are distinguished in these release notes where necessary.

Product Overview

Release 2010 of *PGI CDK* includes the following components:

- PGFORTRAN native OpenMP and auto-parallelizing Fortran 90/95/2003 compiler.
- PGF77 native OpenMP and auto-parallelizing FORTRAN 77 compiler.
- PGHPF data parallel High Performance Fortran 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.

- MPICH2 MPI libraries, version 1.2.1p1, 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

  Note: linux86-64 versions are limited.

- FLEXnet license utilities.

The release contains the following documentation and tutorial materials:

- Online documentation in PDF, HTML and man page formats.
- Online HPF tutorials that provide insight into cluster programming considerations.

  Note

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

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 a term in these notes with which you are not familiar, 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 2, Intel Penryn, and Intel Core i7 (Nehalem) processors.

Supported Platforms

There are six platforms supported by the PGI Workstation and PGI Server compilers and tools:

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

Chapter 2. New or Modified Features

This chapter provides information about the new or modified features of Release 2010 of the PGI compilers and tools as compared to prior releases.

What's New in CDK Release 2010

10.9 Additions

- Updated the version of **MPICH2** used in the CDK to 1.2.1p1.

10.8 Additions

- Additional **Fortran 2003 features** in 10.8 include:
  - **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 Chapter 3 of the PGI User's Guide.
  - **Sourced Allocation** for polymorphic types (no unlimited polymorphics) - uses the `source=` clause in the `allocate` statement to take the type, type parameters, and values from another variable or expression instead of allocating explicitly specified types and type parameters. The allocated variable has the same dynamic type, parameters, and value as the source variable.

    ```
    subroutine example(x)
       class(t), allocate ::a
       class(t)           :: x
       allocate(a, source=x)
    ```
- **PGI Accelerator x64+GPU** native Fortran 95/03 and C99 compilers, and **PGI CUDA Fortran** now support the CUDA 3.1 Toolkit. This CUDA 3.1 Toolkit is an update to the CUDA 3.0 Toolkit, so CUDA 3.0 Toolkit is no longer shipped with the PGI compilers and tools, though you may leave it on your system if it exists from a previous installation.

  Due to changes in the CUDA 3.1 API, CUDA Fortran host programs should be recompiled, as the PGI 10.8 CUDA Fortran runtime libraries are not compatible with previous CUDA Fortran releases.

10.6 Additions

- Additional **Fortran 2003 features** in 10.6 include:
  - deferred type-bound procedures - procedures that are defined for the purpose of defining a base type for a future extension.
  - NON_OVERRIDABLE attribute - specifies that the type-bound procedure cannot be overridden during type extension.
  - PRIVATE statement for type-bound procedures - specifies the accessibility of the type-bound procedure. The accessibility of components within the derived types is separate from the accessibility of the type-bound procedures. By default, a type-bound procedure is `PUBLIC`, unless a `PRIVATE` statement is in the type's procedure section or it is explicitly declared to be `PRIVATE`.
  - PRIVATE and PUBLIC attributes - determine where the type-bound procedures can be referenced. The default is `PUBLIC`, which allows the procedures to be referenced anywhere in the program having that type of variable. If the procedure is `PRIVATE`, it can only be referenced from within the module in which it is defined.

    Mixed component accessibility allows some components of a type to be `PRIVATE` while others are `PUBLIC`. The `PRIVATE` attribute sets the default accessibility of the component, but can be overridden or confirmed in the component definition.
  - ASYNCHRONOUS statement and attribute - warn the compiler that incorrect results might occur for optimizations involving movement of code across wait statements, or statements that cause wait operations.
  - ABSTRACT and DEFERRED - are placeholders. When `abstract` is specified, the compiler warns if any variable is declared of that type. When `deferred` is specified for a procedure, the compiler warns if the procedure is not overridden.
  - The IEEE standard intrinsic module `ieee_features` supports specification of essential IEEE features. It provides access to one derived type and a collection of named constants of this type that affect the manner in which code is compiled in the scoping units.
  - Enhancements to I/O that include the following. For more details on these features, refer to "Fortran I/O Enhancements," on page 18.
    - I/O Decimal specifier - Ability to use a comma instead of a decimal point in input and output. Support for this feature is through the `DECIMAL=scalar_char` clause or through use of the `dp` and `dc` descriptors.
    - I/O Encoding specifier - Ability to specify input/output encoding using the `encoding=` specifier on the OPEN or INQUIRE statements.
  - Intrinsic functions `max`, `min`, `maxloc`, `minloc`, `maxval`, and `minval` now accept arguments of type `character`.
  - Statements `allocate` and `deallocate` now accept an `errmsg=` clause. The clause takes a scalar default character string variable. This variable is assigned an explanatory message if an error condition occurs.

- The `allocate` statement now accepts the `source=` clause. Instead of allocating a variable with an explicitly specified type and type parameter, it is now possible to take the type, type parameters, and value from another variable or expression.

  Specifying a polymorphic variable for the `source=` clause is not yet supported.

- **PGI Accelerator x64+GPU** native Fortran 95/03 and C99 compilers, and **PGI CUDA Fortran** now support the following CUDA built-in functions: **syncthreads_count**, **syncthreads_and**, **syncthreads_or**, **threadfence**, **threadfence_block**, **threadfence_system**, **ballot**. In addition, CUDA Fortran device code now supports the functions **popcnt()**, **poppar()**, and **leadz()**.
- **PGPROF** now supports PGI Accelerator model and CUDA Fortran profiling.

10.5 Additions

- Additional **Fortran 2003 features** in 10.5 include:
  - Type-bound procedures - procedures that are invoked through an object and the actual procedure executed depends on the dynamic type of the object.
  - PASS and NOPASS attributes - allow the procedure to specify to which argument, if any, the invoking object is passed. For example, `pass(x)` passes it to dummy argument x, while `nopass` indicates not to pass it at all.
- **PGI Accelerator x64+GPU native Fortran 95/03 and C99 compilers, and PGI CUDA Fortran** now support the runtime library routine **acc_get_device_num** which returns the number of the device being used to execute an accelerator region. For a complete list of supported routines, refer to "PGI Accelerator Compilers Runtime Libraries".
- **Accelerator Profiling**: If you are profiling a program that uses the PGI Accelerator Model or PGI CUDA Fortran, **pgcollect** automatically collects information about accelerator performance and includes that information in the profile output for the program.

  ### Note

  Inclusion of the accelerator performance information in the program's profile output occurs for both time-based sampling and, on Linux, for event-based sampling.

  On some Linux systems, initialization of the CUDA driver for accelerator hardware that is in a power-save state can take a significant amount of time. You can avoid this delay in one of these ways:

  - Run the `pgcudainit` program in the background, which keeps the GPU powered on and significantly reduces initialization time for subsequent programs. For more information on this approach, refer to Chapter 7 of the *PGI User's Guide*.
  - Use the **pgcollect** option `-accinit` to eliminate much of the initialization overhead and to provide a more accurate profile.

    ```
    pgcollect -time -accinit myaccelprog
    ```

10.4 Additions

- **PGI Accelerator x64+GPU native Fortran 95/03 and C99 compilers, and PGI CUDA Fortran** now support CUDA 3.0 Toolkit and compute capability 2.0.
  - To specify **CUDA compute capability 2.0**, use one of the following options:

In the Accelerator:	`-ta=nvidia:cc20`
In CUDA Fortran:	`-Mcuda=cc20`

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

    In the Accelerator:

In the Accelerator:	
For CUDA toolkit 3.0	`-ta=nvidia:cuda3.0` or `-ta=nvidia:3.0`
For CUDA toolkit 2.3	`-ta=nvidia:cuda2.3` or `-ta=nvidia:2.3`

For CUDA Fortran:	
For CUDA toolkit 3.0	`-Mcuda=cuda3.0` or `-Mcuda=3.0`.
For CUDA toolkit 2.3	`-Mcuda=cuda2.3` or `-Mcuda=2.3`

Note

Compiling with the CUDA 3.0 toolkits generates binaries that may not work on machines with a 2.3 CUDA driver. For more information, refer to "CUDA Fortran Toolkit Issues," on page 34.

- A **wait for kernel** option, `-ta=nvidia:[no]wait`, is available when targeting NVIDIA Accelerator; the user can specify whether to wait for the kernel to finish before continuing in the host program. The default is to wait.
- Added support for **fused multiply-add instructions in CUDA Fortran**. The user can now control the generation of fused multiply-add instructions in both the Accelerator and with CUDA Fortran. In addition to the existing `-ta=nvidia:nofma` option already available for the Accelerator, PGI 10.4 and higher support the equivalent option in CUDA Fortran: `-Mcuda=nofma`.
- Added ability to **use fast math library in CUDA Fortran**. The user can now specify to use routines from the fast math library. In addition to the existing `-ta=nvidia:fastmath` option available for the Accelerator, PGI 10.4 and higher support the equivalent option in CUDA Fortran: `-Mcuda=fastmath`.

- Added support in CUDA Fortran for using **allocatable device arrays in modules** which contain global subroutines, accessible from both the host code which uses the module, and the device code contained within the module.

10.3 Additions

- **Enhanced compute capabilities with CUDA in 10.3**:

  The default compute capabilities target both compute capability 1.0 and 1.3. Further, beginning with 10.3, the user can specify multiple compute capabilities to be targeted.

  For example, to target all four compute capabilities 1.0, 1.1, 1.2, and 1.3, use these options on the command line:

```
-Mcuda=cc10, -Mcuda=cc11, -Mcuda=cc12, -Mcuda=cc13
```

  or

```
-Mcuda=cc10,cc11,cc12,cc13
```

- Additional **Fortran 2003 features** in 10.3 include:
  - Abstract interfaces
  - IS_IOSTAT_END, IS_IOSTAT_EOR, and NEW_LINE intrinsics
  - Object-oriented features including classes, type extensions (non-polymorphic), polymorphic entities, typed allocation, inheritance association, as well as EXTENDS_TYPE_OF and SAME_TYPE_AS intrinsics.
  - New and modified statements, including: WAIT statement; blank, pad, and pos specifiers for the READ statement; delim and pos specifiers for the WRITE statement; and pending and pos specifiers for the INQUIRE statement

  For more information on these features, refer to "Fortran Enhancements," on page 13.

2010 Additions Prior to 10.3

- **PGI Accelerator x64+GPU native Fortran 95/03 and C99 compilers** support the full PGI Accelerator programming model v1.0 standard for directive-based GPU programming and optimization as well as several features from the v1.1 standard.
  - Supported on Linux, MacOS, and Windows
  - Device-resident data using the UPDATE directive
  - COMPLEX and DOUBLE COMPLEX data, Fortran derived types, C structs
  - Automatic GPU-side loop unrolling
  - Support for Accelerator regions nested within OpenMP parallel regions
- **PGI CUDA Fortran extensions** supported in the PGI 2010 Fortran 95/03 compiler enable explicit CUDA GPU programming
  - Declare variables in CUDA GPU device, constant or shared memory

  - Dynamically allocate page-locked pinned host memory, CUDA device main memory, constant memory and shared memory
  - Move data between host and GPU with Fortran assignment statements
  - Declare explicit CUDA grids/thread-blocks to launch GPU compute kernels
  - Support for CUDA Runtime API functions and features
  - Additional options for CUDA Fortran, such as `-Mcuda=keepgpu`, which keeps the generated GPU code for CUDA Fortran.
  - Efficient host-side emulation for easy CUDA Fortran debugging
- **Fortran 2003 incremental features** including: namelist I/O on internal files, IMPORT, pointer reshaping, procedure pointers and statement, iso_c_binding intrinsic module, c_associated, c_f_pointer, c_f_procpointer, enum, move_alloc(), iso_fortran_env module, optional kind to intrinsics, allocatable scalars, volatile attribute and statement, pass and nopass attributes, bind(c), value, command_argument_count, get_command, get_command_argument, get_environment_variable, ieee_exceptions module, and ieee_arithmetic module.
- **PGC++/ PGCC (2010 C++)** new features and enhancements include:
  - The latest EDG release 4.1, with enhanced GNU and Microsoft compatibility
  - `extern` inline support by default: Multiple copies of inline functions in an executable are now removed.
  - Extended internal tables for better support of large codes, including improved BOOST support.
  - C++ `-mp` thread safe exception handling.
- **Expanded Operating Systems Support** including RHEL 5, Fedora 11, SLES 11, SuSE 11.1, Ubuntu 9, Windows 7 and MacOS Snow Leopard
- **Compiler optimizations and enhancements** including:
  - OpenMP support for up to 256 cores
  - AVX code generation
  - Partial redundancy elimination
  - Executable size improvements
- **Updated Documentation** including the *PGI Users Guide*, *PGI Tools Guide*, and *PGI Fortran Reference*.

Getting Started

By default, the PGI 2010 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`.

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

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 detailed in the PGI User's Guide, such as `-Mvect`, `-Munroll`, `-Minline`, `-Mconcur`, `-Mpfi/-Mpfo` and so on. However, increased speeds using these options are typically application- and system-dependent. It is important to time your application carefully when using these options to ensure no performance degradations occur.

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 2010:

- `-m32` indicates to use the 32-bit compiler for the default processor type.
- `-m64` indicates to use the 64-bit compiler for the default processor type.
- `-ta=nvidia(,nvidia_suboptions),host` is a switch associated with the PGI Accelerator compilers. `-ta` defines the target architecture.

  In release 2010, 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.
fastmath	Use routines from the fast math library.
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.

- `-Mautoinline` has new suboptions:

levels:n	Instructs the compiler to perform *n* levels of inlining. The default number of levels is 10.
maxsize:n	Instructs the compiler not to inline functions of size > *n*. The default size is 100.
totalsize:n	Instructs the compiler not to stop inlining when the size equals *n*. The default size is 800.

- New options `-pre` and `-Mnopre` exist to enable/disable partial redundancy elimination.
- New options `-Meh_frame` and `-Mnoeh_frame` instruct the linker to keep eh_frame call frame sections in the executable.

Note

The eh_frame option is available only on newer Linux or Windows systems that supply the system unwind libraries.

- A new option `--gnu_version <num>` exists that sets the GNU C++ compatibility version. (C++ only)
- A new option `-Mcuda` tells the compiler to enable CUDA Fortran. In release 2010, `-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.
emu	Enable CUDA Fortran emulation mode.
fastmath	Use routines from the fast math library.
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.

C++ Compilation Requirements

Note

We break object code compatibility in this release for C++.

All C++ source files and libraries that were built with PGI 9.0 or prior releases must be recompiled to link with 10.1 or higher object files.

Fortran Enhancements

The following sections describe enhancements to Fortran related to interoperability with C, statements, assignments, intrinsics, modules, array-related allocation, and I/O operations.

Enhanced Fortran Interoperability with C

Fortran 2003 provides a mechanism for interoperating with C. Any entity involved must have equivalent declarations made in both C and Fortran. In this release, PGI has expanded Fortran interoperability with C by adding these components:

- Enumerators – a set of integer constants. The kind of enumerator corresponds to the integer type that C would choose for the same set of constants.
- c_f_pointer – a subroutine that assigns the C pointer target, cptr, to the Fortran pointer, fptr, and optionally specifies its shape, shape. The syntax is:

```
c_f_pointer (cptr, fptr [,shape])
```

- c_f_procpointer – a subroutine that associates the C pointer target, cptr, with the target of a C function pointer. The syntax is:

```
c_f_procpointer (cptr, fptr)
```

- c_associated – a subroutine that determines the status of the C pointer target, cptr1, or determines if one C_PTR, cptr1 is associated with a target C_PTR, cptr2. The syntax is:

```
c_associated (cptr1[,cptr2])
```

For more information on these components, refer to Chapter 9, *Interoperability with C* of the *PGI Fortran Reference*.

New or Modified Fortran Statements

The following Fortran statements are new. For complete descriptions, refer to chapter 3, *Fortran Statements* of the *PGI Fortran Reference*.

ASSOCIATE
: 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.

ASYNCHRONOUS
: Indicates to the compiler that incorrect results might occur for optimizations involving movement of code across wait statements, or statements that cause wait operations.

WAIT
: Performs a wait operation for specified pending asynchronous data transfer operations.

The following Fortran statements are enhanced in this release:

ALLOCATE
: New specifiers of ERRMSG and SOURCE are now available.

INQUIRE
: New specifiers of PENDING and POS are now available.

READ
: New specifiers of ASYNCHRONOUS, BLANK, DECIMAL, PAD, and POS are now available.

WRITE
: New specifiers of ASYNCHRONOUS, DECIMAL, DELIM and POS are now available.

New or Modified Fortran Intrinsic Functions

An intrinsic is a function available in a given language whose implementation is handled specifically by the compiler. Since the compiler has an intimate knowledge of the intrinsic function, it can better integrate it and optimize it for the situation. In this release, PGI implemented the following intrinsics. For detailed information about these intrinsics, refer to the chapter 6, *Intrinsics* of the *PGI Fortran Reference*.

EXTENDS_TYPE_OF(A,B)
: Determines whether the dynamic type of A is an extension type of the dynamic type of B. Returns either true or false.

GET_COMMAND_ARGUMENT(NUMBER [, VALUE, LENGTH, STATUS])
: Returns the specified command line argument of the command that invoked the program.

GET_COMMAND([COMMAND, LENGTH, STATUS])
: Returns the entire command line that was used to invoke the program.

GET_ENVIRONMENT_VARIABLE (NAME [,VALUE, LENGTH, STATUS, TRIM_NAME])
: Returns the value of the specified environment variable.

IS_IOSTAT_END(STAT)
: Tests whether a variable has the value of the I/O status: "end of file"; returns either true or false.

IS_IOSTAT_EOR(STAT)
: Tests whether a variable has the value of the I/O status: "end of record"; returns either true or false.

NEW_LINE(A)
: Returns the newline character.

SAME_TYPE_AS(A,B)
: Determines whether the dynamic type of A is the same as the dynamic type of B. Returns either true or false.

New Fortran Intrinsic Modules

PGI 2010 now supports the Fortran intrinsic modules `ieee_arithmetic`, `ieee_exceptions`, and `ieee_features`.

IEEE_ARITHMETIC

The `ieee_arithmetic` intrinsic module provides access to two derived types, named constants of these types, and a collection of generic procedures.

This module behaves as if it contained a `use` statement for the module `ieee_exceptions`, so all the features of `ieee_exceptions` are included.

Note

For specific information on these types, functions, and subroutines, refer to the *PGI Fortran Reference*.

Defined Elemental Operators

- ==

  For two values of one of the derived types, this operator returns true if the values are the same; false, otherwise.

- /=

  For two values of one of the derived types, this operator returns true if the values are different; false, otherwise.

Derived Types

- ieee_class_type - Identifies a class of floating point values.
- ieee_round_type - Identifies a particular round mode.

The following table shows the values that each of these class types can take:

This derived type...	Takes these values...
ieee_class_type	ieee_signaling_nan ieee_quiet_nan ieee_negative_inf ieee_negative_normal ieee_negative_denormal ieee_negative_zero ieee_postive_zero ieee_postive_denormal ieee_postive_normal ieee_postive_inf ieee_other_value (Fortran 2003 only)
ieee_round_type	ieee_nearest ieee_to_zero ieee_up ieee_down

Note

For specific information on these values refer to the *PGI Fortran Reference*.

Inquiry Functions

ieee_support_datatype([x])
ieee_support_denormal([x])
ieee_support_divide([x])
ieee_support_inf([x])
ieee_support_nan([x])
ieee_support_rounding (round_value[,x])
ieee_support_sqrt([x])
ieee_support_standard ([x])
ieee_support_underflow_control ([x]) Fortran 2003 only

Elemental Functions

ieee_class(x)
ieee_copy_sign(x,y)
ieee_is_finite(x)
ieee_is_nan(x)
ieee_is_negative(x)
ieee_is_normal(x)
ieee_is_logb(x)
ieee_next_after(x,y)
ieee_rem(x,y)
ieee_rint(x,y)
ieee_scaln(x,i)
ieee_unordered(x,y)
ieee_value(x,class)
ieee_support_datatype

Non-Elemental Subroutines

ieee_get_rounding_mode(round_value)
ieee_get_underflow_mode(gradual)
ieee_set_rounding_mode(round_value)
ieee_gst_underflow_mode(gradual)

Transformational Function

ieee_selected_real_kind([p] [,r])

For more information on these intrinsic modules, and on the derived types, functions, and subroutines to which they provide access, refer to the *Intrinsics Modules* section of the *PGI Fortran Reference*.

IEEE_EXCEPTIONS

The `ieee_exceptions` intrinsic module specifies accessibility of overflow and divide-by-zero flags as well as determines the level of support for other exceptions. This module provides access to two derived types, named constants of these types, and a collection of generic procedures.

Derived Types

ieee_flag_type - Identifies a particular exception flag.
ieee_status_type - Saves the current floating-point status.

Inquiry Functions

ieee_support_flag(flag [,x])
ieee_support_halting(flag)

Subroutines for Flags and Halting Modes

ieee_get_flag(flag, flag_value)
ieee_get_halting_mode(flag, halting)
ieee_set_flag(flag, flag_value)
ieee_set_halting_mode(flag, halting)

Subroutines for Floating-Point Status

ieee_get_status(status_value)
ieee_set_status(status_value)

For more information on this intrinsic module and the derived types, functions, and subroutines to which it provides access, refer to the *Intrinsics Modules* section of the *PGI Fortran Reference*.

IEEE_FEATURES

The `ieee_features` intrinsic module supports specification of essential IEEE features. It provides access to one derived type and a collection of named constants of this type.

Derived Type

ieee_features_type - Identifies a particular feature and may only take values that are those of named constants defined in the module.

Named Constants

ieee_datatype
ieee_denormal
ieee_divide
ieee_halting
ieee_inexact_flag
ieee_inf
ieee_invalid_flag
ieee_nan
ieee_rounding
ieee_sqrt
ieee_underflow_flag

For more information on this intrinsic module and the derived types, functions, and subroutines to which it provides access, refer to the *Intrinsics Modules* section of the *PGI Fortran Reference*.

Fortran I/O Enhancements

PGI 2010 implements these enhancements related to Input and Output:

- Ability to use a comma instead of a decimal point in input and output. Support for this feature is through the `DECIMAL=scalar_char` clause or through use of the `dp` and `dc` descriptors.
  - The `DECIMAL=scalar_char` clause is available for OPEN, READ, and WRITE statements, where scalar_char is a scalar character expression which takes the value 'point' or 'comma'. When the edit mode is *point*, decimal points appear in both input and output. When the edit mode is *comma*, commas rather than decimal points appear in both input and output.

    For OPEN statements, this value specifies the default decimal edit mode for the unit. For READ/WRITE statements, this decimal edit mode is the default mode only for the duration of that READ/WRITE statement.
  - The `dc` and `dp` descriptors, representing decimal comma and decimal point edit modes, respectively, are valid in format processing, such as in a FORMAT statement. The specific edit mode takes effect immediately when encountered in formatting, and stays in effect until either another descriptor is encountered or until the end of the current I/O statement.
- Ability to specify input/output encoding using the `encoding=` specifier on the OPEN statement. Further, the use of this specifier with the INQUIRE statement returns the encoding of the file:

  `UTF-8` specifies the file is connected for UTF-8 I/O or that the processor can detect this format in some way.

`UNKNOWN` specifies the processor cannot detect the format.
A processor-dependent value indicates the file is in another known format, such as `UTF-16LE`.

New or Modified Tools Support

The PGI Tools Guide describes the tools in detail as well as explains the new features highlighted in this section.

PGPROF graphical MPI/OpenMP/multi-thread performance analysis and tuning profiler has these enhancements in this release:

- PGI Accelerator and CUDA Fortran GPU-side performance statistics
- New data collection mechanism via `pgcollect` enables profiling without re-compiling or any special software co-installation requirements for OProfile. You can use pgcollect in standalone mode for time-based sampling using only PGI software – both on Linux and on Mac OS X 10.5 (Leopard).
- Support for profiling of code in shared object files – on Linux.

  Dynamic libraries are not yet supported on Mac OS X.
- Updated GUI for easier navigation with tabbed access to multiple source files and improved drill-down to assembly code
- Support for profiling of binaries compiled by non-PGI compilers.
- Uniform cross-platform performance profiling without re-compiling or any special software privileges on Linux, MacOS and Windows
- Updated graphical user interface

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 User's Guide*.

Environment Modules

On Linux, if you use the Environment Modules package (e.g., the module load command), PGI 2010 includes a script to set up the appropriate module files.

Chapter 3. PGI Accelerator

An accelerator is a special-purpose co-processor attached to a CPU and to which the CPU can offload data and executable kernels to perform compute-intensive calculations. This chapter describes the new PGI Accelerator compilers, including the collection of compiler directives used to specify regions of code in Fortran and C programs that can be offloaded from a host CPU to an attached accelerator.

Note

For more information and more details about the PGI Accelerator compilers, the programming model, directives, and how to run an accelerator program, refer to Chapter 7, *Using an Accelerator* and Chapter 18, *PGI Accelerator Compilers Reference*, in the *PGI User's Guide*.

Components

The PGI Accelerator compiler technology includes the following components:

- PGFORTRAN auto-parallelizing accelerator-enabled Fortran 95/2003 compiler
- NVIDIA CUDA Toolkit components
- A simple command-line tool to detect whether the system has an appropriate GPU or accelerator card.

No accelerator-enabled debugger is included with this release.

Availability

The PGI 2010 Fortran Accelerator compilers are available only on x86 processor-based workstations and servers with an attached NVIDIA CUDA-enabled GeForce, Quadro, Tesla, or Fermi card. These compilers target all platforms that PGI supports except 64-bit Mac OS X. For a list of supported GPUs, refer to the Accelerator Installation and Supported Platforms list in the latest *Installation Guide*.

User-directed Accelerator Programming

In user-directed accelerator programming the user specifies the regions of a host program to be targeted for offloading to an accelerator device. The bulk of a user's program, as well as regions containing constructs that are not supported on the targeted accelerator, are executed on the host. This chapter concentrates on specification of loops and regions of code to be offloaded to an accelerator.

Features Not Covered or Implemented

Currently the PGI Accelerator compilers do not include features for automatic detection and offloading of regions of code to an accelerator by a compiler or other tool. While future versions of the PGI compilers may allow for automatic offloading, multiple accelerators of the same type, or multiple accelerators of different types, these features are not currently supported.

System Requirements

To use the PGI Accelerator compiler features, you must install the NVIDIA CUDA component: NVIDIA Driver.

You may download this driver from the NVIDIA website at www.nvidia.com/cuda. These are not PGI products, and are licensed and supported by NVIDIA.

Supported Processors and GPUs

This PGI Accelerator compiler release supports all AMD64 and Intel 64 host processors supported by Release 2010 or higher of the PGI compilers and tools. Further, you can use the `-tp <target>` flag as documented in that release.

You can also use the `-ta=nvidia` flag to enable the accelerator directives and target the NVIDIA GPU. You can then use the generated code on any system with CUDA installed that has a CUDA-enabled GeForce, Quadro, Tesla, or Fermi card.

For more information on these flags as they relate to accelerator technology, refer to the PGI User's Guide. For a complete list of supported GPUs, refer to the NVIDIA website at:

www.nvidia.com/object/cuda_learn_products.html

Installation and Licensing

The PGI Accelerator compilers require a separate license key in addition to a normal CDK license key. For specific information related to installation, refer to the *CDK Installation Guide*.

Running an Accelerator Program

Launch a program that has accelerator directives and that was compiled and linked with the `-ta=nvidia` flag in the same way you would launch the program if it had been compiled without the `-ta=nvidia` flag. For more specific information, refer to Chapter 7 in the *PGI User's Guide*.

PGI Accelerator Compilers Runtime Libraries

PGI Accelerator Compilers provide user-callable functions and library routines that are available for use by programmers to query the accelerator features and to control behavior of accelerator-enabled programs at runtime. In Fortran, none of the PGI Accelerator compilers runtime library routines may be called from a PURE or ELEMENTAL procedure.

To access accelerator libraries, you must link an accelerator program with the same `–ta` flag used when compiling.

There are runtime library files for C and for Fortran.

- C Runtime Library Files - In C, prototypes for the runtime library routines are available in a header file named `accel.h`. All the library routines are extern functions with "C" linkage. This file defines:
  - The prototypes of all routines in this section.
  - Any data types used in those prototypes, including an enumeration type to describe types of accelerators.
- Fortran Runtime Library Files - In Fortran, interface declarations are provided in a Fortran include file named `accel_lib.h` and in a Fortran module named `accel_lib`. These files define:
  - Interfaces for all routines in this section.
  - Integer parameters to define integer kinds for arguments to those routines.
  - Integer parameters to describe types of accelerators.

    The integer parameter `accel_version` has a value `yyyymm`, where `yyyy` and `mm` are the year and month designations of the version of the Accelerator programming model supported. This value matches the value of the preprocessor variable `_ACCEL`.

The following list briefly describes the PGI Accelerator compilers runtime library routines that PGI currently supports. For a complete description of these routines, refer to the *PGI Accelerator Runtime Routines* section of the *PGI User's Guide*.

- **acc_get_device** - returns the type of accelerator device being used.
- **acc_get_device_num** - returns the number of the device being used to execute an accelerator region.
- **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_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.

Environment Variables

PGI supports environment variables that modify the behavior of accelerator regions. This section defines the user-setable environment variables used to control behavior of accelerator-enabled programs at execution. These environment variables must comply with these rules:

- The names of the environment variables must be upper case.
- The values assigned environment variables are case insensitive and may have leading and trailing white space.

- The behavior is implementation-defined if the values of the environment variables change after the program has started, even if the program itself modifies the values.

The following list briefly describes the PGI Accelerator environment variables that PGI supports. For more information on these variables, refer to the PGI User's Guide.

- ACC_DEVICE - controls which accelerator device to use when executing PGI Unified Binary for accelerators. The value of this environment variable may be the string NVIDIA or HOST.
- ACC_DEVICE_NUM - controls the default device number to use when executing accelerator regions. The value of this environment variable must be a nonnegative integer between zero and the number of devices attached to the host.
- ACC_NOTIFY - when set to a non-negative integer, indicates to print a message for each kernel launched on the device.

Applicable Command Line Options

There are command line options that apply specifically when working with accelerators.

- `-tp` - use this option to specify the target host processor architecture.
- `-Minfo` or `-Minfo=accel` - use either format of this option to see messages about the success or failure of the compiler in translating the accelerator region into GPU kernels.
- `-ta=nvidia(,nvidia_suboptions),host` - enables recognition of the !$ACC directives in Fortran, and #pragma acc directives in C. [C, Fortran only]

  It has these suboptions:

  - **nvidia** - Select NVIDIA accelerator target.

    This option has the following nvidia-suboptions:

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.
fastmath	Use routines from the fast math library.
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	Wait for each kernel to finish before continuing in the host program.

- **host** - Select the host as the target; when used in combination with the nvidia option, this option generates PGI Unified Binary Code.

The compiler automatically invokes the necessary CUDA software tools to create the kernel code and embeds the kernels in the Linux object file.

To access accelerator libraries, you must link an accelerator program with the same `-ta` flag used when compiling the program.

PGI Unified Binary for Accelerators

PGI compilers support the PGI Unified Binary feature to generate executables with functions optimized for different host processors, all packed into a single binary. This release extends the PGI Unified Binary technology for accelerators. Specifically, you can generate a single binary that includes two versions of functions:

- one version is optimized for the accelerator.
- one version runs on the host processor when the accelerator is not available or when you want to compare host to accelerator execution.

To enable this feature, use the extended –ta flag: `-ta=nvidia,host`

This flag tells the compiler to generate two versions of functions that have valid accelerator regions.

- A compiled version that targets the accelerator.
- A compiled version that ignores the accelerator directives and targets the host processor.

If you use the `-Minfo` flag, you get messages similar to the following during compilation:

```
s1:
     12, PGI Unified Binary version for -tp=barcelona-64 -ta=host
     18, Generated an alternate loop for the inner loop
         Generated vector sse code for inner loop
         Generated 1 prefetch instructions for this loop
s1:
     12, PGI Unified Binary version for -tp=barcelona-64 -ta=nvidia
     15, Generating copy(b(:,2:90))
         Generating copyin(a(:,2:90))
     16, Loop is parallelizable
     18, Loop is parallelizable
         Parallelization requires privatization of array t(2:90)
         Accelerator kernel generated
         16, !$acc do parallel
         18, !$acc do parallel, vector(256)  Using register for t
```

The PGI Unified Binary message shows that two versions of the subprogram s1 were generated:

- one for no accelerator (`-ta=host`)
- one for the NVIDIA GPU (`-ta=nvidia`)

At run time, the program tries to load the NVIDIA CUDA dynamic libraries and test for the presence of a GPU. If the libraries are not available or no GPU is found, the program runs the host version.

You can also set an environment variable to tell the program to run on the NVIDIA GPU. To do this, set ACC_DEVICE to the value NVIDIA or nvidia. Any other value of the environment variable causes the program to use the host version.

The only supported `-ta` targets for this release are `nvidia` and `host`.

Profiling Accelerator Kernels

This release supports the command line option:

```
-ta=nvidia,time
```

The `time` suboption links in a timer library, which collects and prints out simple timing information about the accelerator regions and generated kernels. For a specific example of accelerator kernel timing data, refer to *Chapter 7* in the *PGI User's Guide.*

Supported Intrinsics

PGI Accelerator compilers support Fortran intrinsics. For complete descriptions of these intrinsics, refer to the "*Supported Intrinsics*" section of the *Using an Accelerator* chapter of the *PGI User's Guide.* PGI plans to support additional intrinsics in future releases.

Chapter 4. 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. The application must be installed in such a way that it executes accurately on a system other than the one on which it was built, and which may be configured differently.

Application Deployment and Redistributables

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

PGI Redistributables

The PGI 2010 release includes these directories:

$PGI/linux86/10.9/REDIST
$PGI/linux86/10.9/REDIST-RLR
$PGI/linux86-64/10.9/REDIST
$PGI/linux86-64/10.9/REDIST-RLR

These directories contain all of the PGI Linux runtime library shared object files or Windows dynamically linked libraries that can be re-distributed by PGI 2010 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 2010 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.

Chapter 5. The PGI Windows CDK

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

- Building MPI applications with MSMPI
- Using PGPROF to do MPI profiling of MSMPI applications
- Using PGDBG to do MPI debugging of MSMPI applications

This chapter provides information on these tasks.

Build MPI Applications with MSMPI

Note

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

Using MSMPI libraries

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

Generate MPI Profile Data

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

The profile data generated by running an application built with the option `-Mprof=msmpi` contains information about the number of sends and receives, as well as the number of bytes sent and received, correlated with the source location associated with the sends and receives. You must use `-Mprof=msmpi` in conjunction with either the option `-Mprof=func` or `-Mprof=lines`.

When invoked using this type of profile data, PGPROF automatically displays MPI statistics.

Debug MSMPI Applications with PGDBG

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

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

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

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

To make pgserv.exe available, copy it from the PGI installation directory, such as C:\Program Files\PGI \win64\10.9\bin\ pgserv.exe, into a directory or directories such that the path to pgserv.exe is the same on all nodes in the debug session. Then you can start the debug session as follows:

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

If you use a command similar to the following one, it copies pgserv.exe to the current directory and also sets the path to pgserv.exe to this copy.

Bash Shell Example

Suppose you wanted to debug the following job invoked in a bash shell:

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

You use this command:

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

Important

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

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

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

Note

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

DOS Shell Example

Suppose you wanted to debug the following job invoked in a DOS shell:

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

You use this command:

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

Chapter 6. Troubleshooting Tips and Known Limitations

This chapter contains information about known limitations, documentation errors, and corrections that have occurred to PGI Server and Workstation.

The frequently asked questions (FAQ) section of the pgroup.com web page provides more up-to-date information about the state of the current release. The location is:

`http://www.pgroup.com/support/index.htm`

General Issues

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

- Object and module files created using PGI Workstation 2010 compilers are incompatible with object files from PGI Workstation 5.x and prior releases.
- Object files compiled with `-Mipa` using PGI Workstation 6.1 and prior releases must be recompiled with PGI Workstation 2010.
- The `-i8` option can make programs incompatible with the bundled ACML library. Visit developer.amd.com to check for compatible libraries.
- The `-i8` option can make programs incompatible with MPI and ACML; use of any INTEGER*8 array size argument can cause failures with these 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.

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 unified binaries, that is, programs built with the `-tp=x64` option, is not fully supported. The names of some subprograms are modified in the creation of the unified binary, and PGDBG does not translate these names back to the names used in the application source code. For detailed information on how to debug a unified binary, see www.pgroup.com/support/tools.htm.

PGPROF-related Issues

The following are known issues on PGDBG:

- Using `-Mprof=func`, `-mcmodel=medium` and `-mp` together on any of the PGI compilers can result in segmentation faults by the generated executable. These options should not be used together.
- Programs compiled and linked for gprof-style performance profiling using -pg can result in segmentation faults on system running version 2.6.4 Linux kernels.
- Times reported for multi-threaded sample-based profiles, that is, profiling invoked with `-pg` or `-Mprof=time` options, are for the master thread only. PGI-style instrumentation profiling with `-Mprof={lines | func}` or hardware counter-based profiling using `-Mprof=hwcts` or pgcollect must be used to obtain profile data on individual threads.

CUDA Fortran Toolkit Issues

Note

You can compile with the CUDA 3.1 toolkit, either by adding the `-ta=nvidia:cuda3.1` option to the command line or by adding `set CUDAVERSION=3.1` to the `siterc` file. Using Cuda 3.1 generates binaries that may not work on machines with a 2.3 CUDA driver.

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

For a 2.3 driver: `CUDA Driver Version 2030`
For a 3.0 driver: `CUDA Driver Version 3000`
For a 3.1 driver: `CUDA Driver Version 3010`

Corrections

A number of problems have been corrected in the PGI 2010 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 release in which it was fixed.

Chapter 7. 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.