PGI Visual Fortran®
Release Notes
Release 2010

The Portland Group®

PGI Visual Fortran®

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: 10250133

Contents

Chapter 1. PVF® Release Overview

Welcome to Release 2010 of PGI Visual Fortran®, a set of Fortran compilers and development tools for 32-bit and 64-bit x86-compatible processor-based workstations and servers running versions of the Windows operating system.

This document describes the new features of the PVF IDE interface, differences in the PVF 2010 compilers and tools from previous releases, and late-breaking information not included in the standard product documentation.

PGI Visual Fortran (PVF®) is licensed using FLEXnet, the flexible license management system from Flexera Software*. Instructions for obtaining a permanent license are included in your order confirmation. More information on licensing is in the PVF Installation Guide for this release.

Product Overview

There are three products in the PVF product family. Each product is integrated with a particular version of Microsoft Visual Studio:

- PGI Visual Fortran 2010

  This product is integrated with Microsoft Visual Studio 2010 (VS 2010).

- PGI Visual Fortran 2008

  This product is integrated with Microsoft Visual Studio 2008 (VS 2008).

- PGI Visual Fortran 2005

  This product is integrated with Microsoft Visual Studio 2005 (VS 2005).

Throughout this document, "PGI Visual Fortran" and “PVF” refer to all PVF products collectively. Similarly, "Microsoft Visual Studio" refers to VS 2010, VS 2008 and VS 2005. When it is necessary to distinguish between the products, the document uses the full product name.

Single-user node-locked and multi-user network floating license options are available for all PVF products. When a node-locked license is used, one user at a time can use PVF on the single system where it is installed. When a network floating license is used, a system is selected as the server and it controls the licensing,

and users from any of the client machines connected to the server can use PVF. Thus multiple users can simultaneously use PVF, up to the maximum number of users allowed by the license.

PVF provides a complete Fortran development environment fully integrated with Microsoft Visual Studio 2010, 2008, or 2005. It includes a custom Fortran Build Engine that automatically derives build dependencies, a Fortran-aware editor, a custom PGI Debug Engine integrated with the Visual Studio debugger, PGI Fortran compilers, and PVF-specific property pages to control the configuration of all of these.

Release 2010 of PGI Visual Fortran includes the following components:

- PGFORTRAN OpenMP and auto-parallelizing Fortran 90/95 compiler
- PGF77 OpenMP and auto-parallelizing FORTRAN 77 compiler
- PVF Visual Studio integration components
- AMD Core Math Library 4.4.0 (ACML)
- PVF documentation

PGI Visual Fortran 2010 and 2008 are available with the Microsoft Visual Studio Shell. PVF 2010 ships with the VS 2010 Shell and PVF 2008 ships with the VS 2008 Shell SP1. Use these packages if you do not already have Visual Studio installed on your system. Otherwise, download the versions of PVF without the VS Shell, since these are much smaller.

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.

Chapter 2. Compiler Features

This chapter contains the new or modified features of this release of PGI Visual Fortran as compared to prior releases.

What's New in PVF Release 2010

10.9 Additions

- Additional **PGI Visual Fortran** features in 10.9 include a new property page that facilitates building and linking against specific libraries, such as **Intel's Math Kernel Library, AMD's Core Math Library**, or **Rogue Wave's IMSL numerical library**.

10.8 Additions

- Additional **PGI Visual Fortran** features in 10.8 include a property that supports specifying the CUDA 3.1 toolkit. When `Enable CUDA Fortran` is set to `Yes`, you can now use the `Fortran|Language|CUDA Fortran Toolkit` property to specify version 3.1 of the CUDA toolkit as the one targeted by the compilers. You can also access this version of the toolkit from the `Fortran|Target Accelerators|NVIDIA:CUDA Toolkit` property.
- 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.

10.6 Additions

- **Support for Visual Studio 2010**. PGI Visual Fortran is completely integrated with Microsoft's recent release of Visual Studio 2010.
- Three additional **PVF Samples** are available:

  gpu/AccelPM_Matmul
  gpu/CUDAFor_Matmul
  interlanguage/vcmain_calling_pvfdll

  For more detailed information on these samples, refer to "New or Modified PVF Samples," on page 16.
- 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 15.
  - 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 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.
- **Enhanced Variable Rollover, Watch** and **Quick Watch** support. Fortran-specific support for user-defined type members and array elements is improved in this release.
- Additional **PVF build macros** are available. The following lists summarizes lists and briefly describes these macros.

$(Configuration)	the name of the current project configuration (for example, "Debug")
$(ConfigurationType)	the type of the current project configuration - one of the following: "Application" , "StaticLibrary", or "DynamicLibrary"
$(OutputPath)	the path to the directory for output files, relative to the project directory, as set by the Output Directory property
$(OutputType)	the type of the current project output - one of the following: "exe", "staticlibrary", or "library"
$(Platform)	the name of the current project platform (for example, "x64").
$(PlatformArchitecture)	the name of the current project platform architecture: for Win32: 32; for x64: 64
$(PlatformShortName)	the description of the architecture ABI for the current project platform: for Win32: x86; for x64: amd64

- **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.

10.4 Additions

- Full support for the **PGI Accelerator programming model** in PGI Visual Fortran
- Complete support for **PGI CUDA Fortran** on NVIDIA CUDA-enabled GPUs.
- Support for **CUDA 3.0 Toolkit** in the PGI Accelerator x64+GPU native Fortran 95/03 compilers and in PGI CUDA Fortran. To specify the version of the CUDA toolkit that is targeted by compilers, use one of the following properties:
  - Set Fortran | Language | Enable CUDA Fortran to *Yes* and then use CUDA Fortran Toolkit to specify the version.
  - Set Fortran | Target Accelerator | Target NVIDIA Accelerator to *Yes* and then use NVIDIA: CUDA Toolkit to specify the version.

Note

Compiling with the CUDA 3.0 toolkit 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 31.

- **Support for compute capability 2.0** in the PGI Accelerator x64+GPU native Fortran 95/03 compilers and in PGI CUDA Fortran. To specify CUDA compute capability 2.0, use one of the following properties:
  - Set Fortran | Language | CUDA Fortran Compute Capability to *Manual* and the CC 2.0 property to *Yes*.
  - Set Fortran | Target Accelerator | Target NVIDIA Accelerator to *Yes*, NVIDIA Compute Capability to *Manual*, and NVIDIA: CC 2.0 property to *Yes*.

  By default, the compiler targets these three compute capabilities: 1.0, 1.3, and 2.0.

  ### Important

  The user can now choose whether to manually or automatically determine the compute capabilities for both CUDA Fortran and Target Accelerator NVIDIA.
- The new **NVIDIA: Synchronous Kernel Launch** property, available from the Fortran | Target Accelerator property page when Fortran | Target Accelerator | Target NVIDIA Accelerator is set to *Yes*, specifies to wait for each kernel to finish before continuing in the host program.
- The new Fortran | Language | **CUDA Fortran Use Fused Multiply-Adds** property allows the user to control the generation of fused multiply-add instructions with CUDA Fortran.
- The new Fortran | Language | **CUDA Fortran Use Fast Math Library** property allows the user to specify to use routines from the fast math library.
- 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**:

  The default compute capabilities in 10.3 target both compute capability 1.0 and 1.3. Further, the user can specify multiple compute capabilities to be targeted. For example, to target all the compute capabilities 1.0, 1.1, 1.2, and 1.3: from the Fortran | Language property page, first set the CUDA Fortran Compute Capability to `Yes`, and then set each of the properties CUDA Fortran CC1.0, CUDA Fortran CC1.1, CUDA Fortran CC1.2, and CUDA Fortran CC1.3 to `Yes`.
- 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 10.

Prior to 10.3

New features in PVF 2010 prior to 10.3 are:

- Includes the standalone **PGPROF** performance profiler with CCFF support.
- **PGI Accelerator x64+GPU native Fortran 95/03 compilers** now support the full PGI Accelerator programming model v1.0 standard for directive-based GPU programming and optimization.
  - Device-resident data using the UPDATE directive
  - COMPLEX and DOUBLE COMPLEX data, Fortran derived types
  - 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
  - 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.

  For more information on these features, refer to "Fortran Enhancements," on page 10.
- Expanded Operating Systems Support including **Windows 7**.
- **Compiler optimizations and enhancements** including:
  - OpenMP support for up to 256 cores
  - AVX code generation

- Executable size improvements

Compiler Optimizations and Features

Additional Compiler Optimizations and features available in Release 2010 include:

- Computation and reporting of compute intensity of loops in all languages
- Packed SSE code generation for unrolled loops
- SSE vectorization of generalized reduction loops
- Improved scalar prefetching, spill tuning and live range splitting
- Improved static estimation of block execution frequencies
- Auto-generation of DWARF for improved tools interoperability
- Enhanced Fortran 95 DWARF generation

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	Wait for each kernel to finish before continuing in the host program.

- `-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.
- 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.

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 the *Interoperability with C* chapter of the *PGI Fortran Reference for PVF*.

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 for PVF*.

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 BLANK, PAD, and POS are now available.

WRITE
: New specifiers of 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 *Intrinsics* chapter of the *PGI Fortran Reference for PVF*.

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` and `ieee_exceptions`. The following sections provide more detail about these modules.

Note

For specific information on these intrinsic modules and more details related to the types, functions, and subroutines to which they provide access, refer to the *PGI Fortran Reference for PVF*.

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 for PVF*.

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 for PVF*.

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 to the derived types, functions, and subroutines to which they provide access, refer to the *Intrinsics Modules* section of the *PGI Fortran Reference for PVF*.

IEEE_EXCEPTIONS

The `ieee_exceptions` intrinsic 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 for PVF*.

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 for PVF*.

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 Runtime Library Routines

PGI 2010 supports new runtime library routines associated with the PGI Accelerator compilers. For more information, refer to "PGI Accelerator Compilers Runtime Libraries," on page 22.

New or Modified PVF Samples

Additional **PVF Samples** are available from the PVF installation directory, typically in a directory called `Samples`:

```
$(VSInstallDir)\PGI Visual Fortran\Samples\
```

In the `gpu` subdirectory of the `Samples` directory, you find these new sample programs which require a PGI Accelerator License to compile and a GPU to run.

`AccelPM_Matmul`
: Uses directives from the PGI Accelerator Programming Model to offload a matmul computation to a GPU.

`CUDAFor_Matmul`
: Uses CUDA Fortran to offload a matmul computation to a GPU.

In the `interlanguage` subdirectory of the `Samples` directory, you find this new sample program which requires that Visual C++ be installed to build and run:

`vcmain_calling_pvfdll`
: Calls a routine in a PVF DLL from a main program compiled by VC++.

For a complete list of sample projects available in PVF, refer to Chapter 1 of the PVF User's Guide.

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.
- 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 Windows
- Updated graphical user interface

MPI Support

Message Passing Interface (MPI) is a set of function calls and libraries that are used to send messages between multiple processes. These processes can be located on the same system or on a collection of distributed servers. Unlike OpenMP, the distributed nature of MPI allows it to work in almost any parallel environment. Further, distributed execution of a program does not necessarily mean that you must run your MPI job on many machines.

In this release, PVF provides built-in support for Microsoft's version of MPI: MSMPI. Inside PVF you can build, run, and debug MSMPI applications with ease. For information on how to compile, run, and debug your MPI application, refer to Chapter 4, *Using MPI in PVF* of the *PVF User's Guide*.

Important

To use PVF's MPI features, you must first install additional Microsoft software which you can download from Microsoft. For the specific corequirements and how to obtain the software, refer to the "*MPI Corequirements*" section of the *PVF Installation Guide*.

Chapter 3. Selecting an Alternate Compiler

Each release of PGI Visual Fortran contains two components - the newest release of PVF and the newest release of the PGI compilers and tools that PVF targets.

When PVF is installed onto a system that contains a previous version of PVF, the previous version of PVF is replaced. The previous version of the PGI compilers and tools, however, remains installed side-by-side with the new version of the PGI compilers and tools. By default, the new version of PVF will use the new version of the compilers and tools. Previous versions of the compilers and tools may be uninstalled using Control Panel | Add or Remove Programs.

There are two ways to use previous versions of the compilers:

- Use a different compiler release for a single project.
- Use a different compiler release for all projects.

The method to use depends on the situation.

For a Single Project

To use a different compiler release for a single project, you use the compiler flag -V<ver> to target the compiler with version <ver>. This method is the recommended way to target a different compiler release.

For example, -V10.1 causes the compiler driver to invoke the 10.1 version of the PGI compilers if these are installed.

To use this option within a PVF project, add it to the Additional options section of the Fortran | Command Line and Linker | Command Line property pages.

For All Projects

You can use a different compiler release for all projects. The Tools | Options dialog within PVF contains entries that can be changed to use a previous version of the PGI compilers. Under Projects and Solutions | PVF Directories, there are entries for Executable Directories, Include and Module Directories, and Library Directories.

- For the x64 platform, each of these entries includes a line containing $(PGIToolsDir). To change the compilers used for the x64 platform, change each of the lines containing $(PGIToolsDir) to contain the path to the desired bin, include, and lib directories.
- For the Win32 platform, these entries include a line containing $(PGIToolsDir) on Win32 systems or $(PGIToolsDir32) on Windows x64 systems. To change the compilers used for the Win32 platform, change each of the lines containing $(PGIToolsDir) or $(PGIToolsDir32) to contain the path to the desired bin, include, and lib directories.

Warning

The debug engine in PVF 2010 is not compatible with previous releases. If you use Tools | Options to target a release prior to 2010, you cannot use PVF to debug. Instead, use the -V method described in section 3.1 to select an alternate compiler.

Chapter 4. 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 10, *Using an Accelerator* and Chapter 20, *PGI Accelerator Compilers Reference*, in the *PVF 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
- PVF Target Accelerators property page.
- 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. For a list of supported GPUs, refer to the Accelerator Installation and Supported Platforms list in the latest *PVF 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 Target Processors property page as documented in this release.

You can also use the `-ta=nvidia` flag to enable the accelerator directives and target the NVIDIA GPU. This flag is available from PVF's Target Accelerators property page. 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 PVF 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 PVF license key. For specific information related to installation, refer to the *PVF Installation Guide.*

Running an Accelerator Program

In PVF you can use the PVF Target Accelerators property page to enable accelerator compilation. For more information on the properties, refer to *Tips on Running Accelerator Programs* section of the *PVF User's Guide.*

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 10 in the *PVF 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. When you use the Target Accelerator properties page, this flag is automatically added to both compilation and linking.

There are runtime library files for Fortran.

- 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 *PVF 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 PVF 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. The options are available from the property pages in PVF.

Applicable PVF Property Pages

The following property pages are applicable specifically when working with accelerators.

Fortran | Target Accelerators
Use the `-ta` option to enable recognition of Accelerator directives.

Fortran | Target Processors
Use the `-tp` option to specify the target host processor architecture.

Fortran | Diagnostics
Use the `-Minfo` option to see messages about the success or failure of the compiler in translating the accelerator region into GPU kernels.

For more information about the many suboptions available with these options, refer to the *Fortran Property Pages* section of the *PVF User's Guide*.

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 Target Accelerators properties page to select `Yes` for both the Target NVIDIA Accelerator and the Target Host properties.

These properties tell the compiler to generate two versions of functions that have valid accelerator regions.

If you enable the Unified Binary Information property on the Diagnostics property page, 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 Target Accelerator property NVIDIA: Enable Profiling (`-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 10* in the *PVF 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 *PVF User's Guide.* PGI plans to support additional intrinsics in future releases.

Chapter 5. 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 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

PGI Visual Fortran includes redistributable directories which contain all of the PGI 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 copy of the PGI EULA in PDF form is included in the release.

The following paths for the redistributable directories assume 'C:' is the system drive.

- On a Win32 system, there are two redistributable directories:

```
C:\Program Files\PGI\win32\10.9\REDIST
C:\Program Files\PGI\win32\10.9\REDIST-RLR
```

- On a Win64 system, there are four redistributable directories:

```
C:\Program Files\PGI\win64\10.9\REDIST
C:\Program Files\PGI\win64\10.9\REDIST-RLR
C:\Program Files (x86)\PGI\win32\10.9\REDIST
C:\Program Files (x86)\PGI\win32\10.9\REDIST-RLR
```

The redistributable directories contain the PGI runtime library DLLs for all supported targets. This enables users of the PGI compilers to create packages of executables and PGI runtime libraries that execute

successfully on almost any PGI-supported target system, subject to the requirement that end-users of the executable have properly initialized their environment to use the relevant version of the PGI DLLs.

Microsoft Redistributables

PGI Visual Fortran includes Microsoft Open Tools, the essential tools and libraries required to compile, link, and execute programs on Windows. PVF 2010 includes the latest version, version 10, of the Microsoft Open Tools.

The Microsoft Open Tools directory contains a subdirectory names REDIST. PGI 2010 licensees may redistribute the files contained in this directory in accordance with the terms of the associated license agreements.

Chapter 6. Troubleshooting Tips and Known Limitations

This chapter contains information about known limitations, documentation errors, and corrections that have occurred to PVF 2010. Whenever possible, a workaround is provided.

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

Visual Studio Shell Limitations

- The 2005 and 2008 versions of the Visual Studio Shell do not support the `Project From Existing Code` menu option on the `File | New` menu. If you select this menu option when it is not supported, you will see a dialog that reads:

  ```
  Class not registered. Looking for object with CLSID: {81A1ED1C-76B9-4363-A838-F413FCCBB606}.
  ```

  You can safely ignore this message.

Use MPI in PVF Limitations

- The multi-process debug style known as "Run One At a Time" is not supported in this release.

PVF IDE Limitations

The issues in this section are related to IDE limitations.

- Integration with source code revision control systems is not supported.
- When moving a project from one drive to another, all .d files for the project should be deleted and the whole project should be rebuilt. When moving a solution from one system to another, also delete the solution's Visual Studio Solution User Options file (.suo).
- The Resources property pages are limited. Use the Resources | Command Line property page to pass arguments to the resource compiler. Resource compiler output must be placed in the intermediate directory for build dependency checking to work properly on resource files.

- There are several properties that take paths or pathnames as values. In general, these may not work as expected if they are set to the project directory $(ProjectDir) or if they are empty, unless empty is the default. Specifically:

  General | *Output Directory* should not be empty or set to $(ProjectDir).
  General | *Intermediate Directory* should not be empty or set to $(ProjectDir).
  Fortran | *Output* | *Object File Name* should not be empty or set to $(ProjectDir).
  Fortran | *Output* | *Module Path* should not be empty or set to include $(ProjectDir).

- Dragging and dropping files in the Solution Explorer that are currently open in the Editor may result in a file becoming "orphaned." Close files before attempting to drag-and-drop them.

PVF Debugging Limitations

The following limitations apply to PVF debugging:

- Debugging of unified binaries is not fully supported. The names of some subprograms are modified in the creation of the unified binary, and the PVF debug engine does not translate these names back to the names used in the application source code. For more information on debugging a unified binary, see www.pgroup.com/support/tools.htm.
- In some situations, using the Watch window may be unreliable for local variables. Calling a function or subroutine from within the scope of the watched local variable may cause missed events and/or false positive events. Local variables may be watched reliably if program scope does not leave the scope of the watched variable.
- Rolling over Fortran arrays during a debug session is not supported when Visual Studio is in Hex mode. This limitation also affects Watch and Quick Watch windows.

  Workaround: deselect Hex mode when rolling over arrays.

PGI Compiler Limitations

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

- If an executable is linked with any PVF-compiled DLL, the PVF runtime library DLLs must be used (in particular the static libraries cannot be used). To accomplish this, use the compiler option `-Bdynamic` when creating the executable.
- Do not use `-Mprof` with PVF runtime library DLLs. To build an executable for profiling, use the static libraries. The static libraries will be used by default in the absence of `-Bdynamic`.
- The `–i8` option can make programs incompatible with MPI; use of any INTEGER*8 array size argument can cause failures with these libraries.
- The `–i8` option can make programs incompatible with the bundled ACML library. Visit developer.amd.com to check for compatible libraries.
- Using `-Mprof=func` and `–mp` together with any of the PGI compilers can result in segmentation faults by the generated executable. These options should not be used together.

- Using `-Mpfi` and `-mp` together is not supported. The `-Mpfi` flag disables `-mp` at compile time, which can cause run-time errors in programs that depend on interpretation of OpenMP directives or pragmas. Programs that do not depend on OpenMP processing for correctness can still use profile feedback. Using the `-Mpfo` flag does not disable OpenMP processing.
- ACML 4.4.0 is built using the `-fastsse` compile/link option, which includes `-Mcache_align`. When linking with ACML on Win32, all program units must be compiled with `-Mcache_align`, or an aggregate option such as `-fastsse`, which incorporates `-Mcache_align`. This process is not an issue on 64-bit targets where the stack is 16-byte aligned by default. You can use the lower-performance, but fully portable, `blas` and `lapack` libraries on CPUs that do not support SSE instructions.

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 the following statement to the `siterc` file:

```
set CUDAVERSION=3.1;
```

Using the CUDA 3.1 toolkit generates binaries that may not work on machines with a CUDA 2.3 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

Refer to www.pgroup.com/support/release_tprs.htm for a complete, up-to-date table of technical problem reports, TPRs, fixed in recent releases of the PGI compilers and tools. The 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.