

PGI Visual Fortran[®]

Release Notes 2010

The Portland Group[®]
STMicroelectronics
Two Centerpointe Drive
Lake Oswego, OR 97035

While every precaution has been taken in the preparation of this document, The Portland Group® (PGI®), a wholly-owned subsidiary of STMicroelectronics, Inc., makes no warranty for the use of its products and assumes no responsibility for any errors that may appear, or for damages resulting from the use of the information contained herein. The Portland Group retains the right to make changes to this information at any time, without notice. The software described in this document is distributed under license from STMicroelectronics and/or The Portland Group and may be used or copied only in accordance with the terms of the license agreement ("EULA").

No part of this document may be reproduced or transmitted in any form or by any means, for any purpose other than the purchaser's or the end user's personal use without the express written permission of STMicroelectronics and/or The Portland Group.

PGI Visual Fortran®

Copyright © 2010 The Portland Group® and STMicroelectronics, Inc.-

All rights reserved.

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



Technical support: trs@pgroup.com

Sales: sales@pgroup.com

Web: www.pgroup.com

Contents

1. PVF[®] Release Overview	1
Product Overview	1
Terms and Definitions	2
2. Compiler Features	3
What's New in PVF Release 2010	3
Compiler Optimizations and Features	4
New or Modified Compiler Options	5
Fortran Enhancements	6
Enhanced Fortran Interoperability with C	6
New or Modified Fortran Statements	6
New or Modified Fortran Intrinsic Functions	7
New Fortran Intrinsic Modules	7
Additional Fortran Enhancements	10
New or Modified Runtime Library Routines	10
New or Modified Tools Support	11
MPI Support	11
3. Alternate Compiler Release Selection	13
For a Single Project	13
For All Projects	13
4. PGI Accelerator	15
Components	15
Availability	15
User-directed Accelerator Programming Line Flag	15
Features Not Covered or Implemented	16
System Requirements	16
Supported Processors and GPUs	16
Installation and Licensing	16
Running an Accelerator Program	16
PGI Accelerator Compilers Runtime Libraries	17

Environment Variables	18
Applicable Command Line Options	18
Applicable PVF Property Pages	18
PGI Unified Binary for Accelerators	19
Multiple Accelerators	20
Profiling Accelerator Kernels	20
Supported Ininsics	21
5. Distribution and Deployment	23
Application Deployment and Redistributables	23
PGI Redistributables	23
Microsoft Redistributables	24
6. Troubleshooting Tips and Known Limitations	25
Use MPI in PVF Limitations	25
PVF IDE Limitations	25
PVF Debugging Limitations	26
PGI Compiler Limitations	26
Corrections	27
7. Contact Information	29

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

- PGI Visual Fortran 2005

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

- PGI Visual Fortran 2008

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

Throughout this document, "PGI Visual Fortran" and "PVF" refer to both PVF products collectively. Similarly, "Microsoft Visual Studio" refers to both VS 2005 and VS 2008. 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 both 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 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 4.3-0 (ACML)
- PVF documentation fully integrated with Visual Studio help

PGI Visual Fortran 2008 also includes:

- Microsoft Visual Studio 2008 Shell SP1 (Integrated Development Environment - IDE)

A version of PVF 2008 is also available without the Visual Studio Shell. Use this package if you already have Visual Studio 2008 installed on your system.

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

- **PGI Visual Fortran MSMPI Support** enables users to build, run, and debug MSMPI Fortran programs on either a local Windows laptop or workstation, or a distributed-memory cluster running Microsoft Windows HPC Server 2008 - all from within the Visual Studio 2005 or 2008 integrated development environment (IDE).
- **PGI Visual Fortran full support for the PGI Accelerator programming model**
- Support for **PGI CUDA Fortran** on NVIDIA CUDA-enabled GPUs.
- Includes the standalone **PGPROF** performance profiler with CCFE support.
- **Expanded and reorganized property pages**
- **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
 - Enhanced compute capabilities with CUDA in 10.3:

The default compute capabilities now target both computer capability 1.0 and 1.3. Further, the user can specify multiple compute capabilities to be targeted. For example, to target the 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
```

- 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
- **Visual Studio 2008 Shell SP1 bundled into PVF 2008** - For users with Visual Studio 2008 already installed on their systems, a version of PVF 2008 without the Visual Studio Shell is also available.
- **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.

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 6](#).

- **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
- **Continued support for Visual Studio 2005**, including support for concurrent use of Visual Studio 2005 and 2008 on the same system.
- **Updated Documentation** including the *PVF User’s Guide*, *PGI Tools Guide*, and *PVF Fortran Reference*.

Compiler Optimizations and Features

- **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` have expanded to include:

<code>fastmath</code>	Use routines from the fast math library.
<code>keepgpu</code>	Keep the binary (.bin) files.
<code>keepgpu</code>	Keep the kernel source (.gpu) files.
<code>keepptx</code>	Keep the portable assembly (.ptx) file for the GPU code.
<code>maxregcount:n</code>	Specify the maximum number of registers to use on the GPU. Leaving this blank indicates no limit.
<code>mul24</code>	Use 24-bit multiplication for subscripting.

- `-Mautoinline` has new suboptions:

<code>levels:n</code>	Instructs the compiler to perform n levels of inlining. The default number of levels is 10.
<code>maxsize:n</code>	Instructs the compiler not to inline functions of size $> n$. The default size is 100.
<code>totalsize:n</code>	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	Generate code for compute capability 1.0.
cc11	Generate code for compute capability 1.1.
cc12	Generate code for compute capability 1.2.
cc13	Generate code for compute capability 1.3.
emu	Enable CUDA Fortran emulation mode.
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.

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 *PVF Fortran Reference*.

New or Modified Fortran Statements

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

WAIT

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

The following Fortran statements are enhanced in this release:

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 *Fortran Reference Guide*.

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 PVF Fortran Reference.

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.

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...
<code>ieee_class_type</code>	<code>ieee_signaling_nan</code> <code>ieee_quiet_nan</code> <code>ieee_negative_inf</code> <code>ieee_negative_normal</code> <code>ieee_negative_denormal</code> <code>ieee_negative_zero</code> <code>ieee_postive_zero</code> <code>ieee_postive_denormal</code> <code>ieee_postive_normal</code> <code>ieee_postive_inf</code> <code>ieee_other_value</code> (Fortran 2003 only)
<code>ieee_round_type</code>	<code>ieee_nearest</code> <code>ieee_to_zero</code> <code>ieee_up</code> <code>ieee_down</code>

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 PVF Fortran Reference.

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.

Additional Fortran Enhancements

PGI 2010 partially implements **Fortran 2003 Asynchronous Input/Output** in PGF77 and PGFORTRAN compilers.

- For external files opened with `ASYNCHRONOUS='YES'` in the `OPEN` statement, asynchronous I/O is allowed.
- Asynchronous I/O operations are indicated by `ASYNCHRONOUS='YES'` in `READ` and `WRITE` statements.
- The compilers do not implement the `ASYNCHRONOUS` attribute or `ASYNCHRONOUS` statement.

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 17.](#)

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:

- 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
- PGI Accelerator and CUDA Fortran GPU-side performance statistics
- 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. Alternate Compiler Release Selection

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>`.

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 and directives, 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 90/95 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 or profiler 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 GPU or Tesla 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 Release Notes.

User-directed Accelerator Programming Line Flag

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 the Target Accelerators property page. You can then use the generated code on any system with CUDA installed that has a CUDA-enabled GeForce, Quadro, or Tesla 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 PGI 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.

Running a program that has accelerator directives and was compiled and linked with the `-ta=nvidia` flag is the same as running the program compiled without the `-ta=nvidia` flag.

- The program looks for and dynamically loads the CUDA libraries. If the libraries are not available, or if they are in a different directory than they were when the program was compiled, you may need to append the CUDA library directory to your PATH environment variable on Windows.

- If you run an accelerated program on a system without a CUDA-enabled NVIDIA GPU, or without the CUDA software installed in a directory where the runtime library can find it, the program fails at runtime with an error message.
- If you set the environment variable `ACC_NOTIFY` to a nonzero integer value, the runtime library prints a line to standard error every time it launches a kernel.

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 separate runtime library files for C and 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 supported 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_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-settable 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 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 PVE.

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.

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

If you enable the Unified Binary Information property on the Diagnostics property page 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`.

Multiple Accelerators

You can use multiple NVIDIA GPUs in your application by writing a parallel MPI or OpenMP program.

MPI

In an MPI program where each MPI rank is actually running in parallel on the same node, you can use the MPI rank value to select a different GPU to use with the `acc_set_device_num` procedure.

OpenMP

In an OpenMP program, you can use the `omp_get_thread_num` function result to select a different GPU to use for each thread.

In an OpenMP parallel region, you can include an accelerator region with loops such that each thread computes a different subset of the iterations.

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.

Example 4.1. Sample Accelerator Kernel Timing Data

```
Accelerator Kernel Timing data
bb04.f90
s1
  15: region entered 1 times
time(us): total=1490738
           init=1489138 region=1600
           kernels=155 data=1445
w/o init: total=1600 max=1600
           min=1600 avg=1600
  18: kernel launched 1 times
time(us): total=155 max=155 min=155 avg=155
```

In this example, a number of things are occurring:

- For each accelerator region, the file name `bb04.f90` and subroutine or function name `s1` is printed, with the line number of the accelerator region, which in the example is 15.
- The library counts how many times the region is entered (1 in the example) and the microseconds spent in the region (in this example 1490738), which is split into initialization time (in this example 1489138) and execution time (in this example 1600).
- The execution time is then divided into kernel execution time and data transfer time between the host and GPU.
- For each kernel, the line number is given, (18 in the example), along with a count of kernel launches, and the total, maximum, minimum, and average time spent in the kernel, all of which are 155 in this example.

Supported Intrinsic

PGI Accelerator compilers support Fortran intrinsics. For complete descriptions of these intrinsics, refer to the "*Supported Intrinsic*" 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 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

PGI Visual Fortran 2010 includes directories named REDIST. These directories 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 REDIST directories assume 'C:' is the system drive.

- On a Win32 system, there are two redistributable directories:
 - C:\Program Files\PGI\win32\10.3\REDIST
 - C:\Program Files\PGI\win32\10.3\REDIST-RLR
- On a Win64 system, there are four REDIST directories:
 - C:\Program Files\PGI\win64\10.3\REDIST and
 - C:\Program Files\PGI\win64\10.3\REDIST-RLR and
 - C:\Program Files (x86)\PGI\win32\10.3\REDIST
 - C:\Program Files (x86)\PGI\win32\10.3\REDIST-RLR

The REDIST 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 9, of the Microsoft Open Tools.

The Microsoft Open Tools directory contains two subdirectories with names that begin with 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

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.
- PVF on Vista or Server 2008 may be unable to attach to an application running on a mapped network drive. If this occurs, run the application to which you want to attach on a local drive.
- 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.

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 `-Mphi` and `-mp` together is not supported. The `-Mphi` 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 `-Mpf0` flag does not disable OpenMP processing.
- ACML 4.3.0 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.

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.

