PGI Visual Fortran®
Release Notes

Version 11.8

The Portland Group®

PGI Visual Fortran®

Printed in the United States of America

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

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

ID: 11221115

Contents

Chapter 1. PVF® Release Overview

Welcome to Release 2011 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 2011 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 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).

Throughout this document, "PGI Visual Fortran" and "PVF" refer to all PVF products collectively. Similarly, "Microsoft Visual Studio" refers to VS 2010 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 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. 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 2011 of PGI Visual Fortran includes the following components:

- PGFORTRAN™ native OpenMP and auto-parallelizing Fortran 90/95/2003 compiler.
- PGF77® native OpenMP and auto-parallelizing FORTRAN 77 compiler.
- PGPROF® Performance Profiler
- PVF Visual Studio integration components
- AMD Core Math Library 4.4.0 (ACML)
- PVF documentation

PGI Visual Fortran is available with the Microsoft Visual Studio Shell. Use this package 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, AMD Istanbul, and AMD Bulldozer processors.
- Intel 64 – a 64-bit IA32 processor with Extended Memory 64-bit Technology extensions designed to be binary compatible with AMD64 processors. This includes Intel Pentium 4, Intel Xeon, Intel Core 2, Intel Core 2 Duo (Penryn), Intel Core (i3, i5, i7) both first generation (Nehalem) and second generation (Sandy Bridge) processors.

Chapter 2. New or Modified 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 2011

11.8 Updates and Additions

- PGI 11.8 includes an update of Cygwin to version 1.7.9-1.
- **PGI Accelerator** compilers and **CUDA Fortran** include these modifications and enhancements:
  - CUDA Fortran supports the option `-mcmodel=medium`.
  - PGI provides a module which defines interfaces to the CUBLAS Library from PGI CUDA Fortran. These interfaces are made accessible by placing the following statement in the CUDA Fortran host-code program unit:

    ```
    use cublas
    ```

11.7 Updates and Additions

- Fortran 2003 now supports final subroutines that allow the programmer to specify subroutine(s) that get called when a variable is deallocated or ceases to exist (i.e., no longer in scope). z

  This allows the program to clean up, perhaps release some resources, when the variable no longer exists. For more information, refer to "Object-Oriented Programming Enhancements," on page 18.
- **PGDBG** has a number of enhancements in this release:
  - New graphical presentation of stack trace information in PGDBG's Call Stack tab.
  - Enhanced support for command-line editing in PGDBG text mode and the PGDBG GUI's Command Prompt tab. Includes support for arrow and control keys and, in the GUI, text selection and replacement.

11.6 Updates and Additions

- The ALLOCATE statement now correctly accepts the `source=` qualifier for a polymorphic source type with allocatable members.

- Support is available for Intel's Sandy Bridge processor. From the `Fortran|Target Processors` property page, use the property `Intel Sandy Bridge`, which adds the `–tp=sandybridge-64` or the `–tp=sandybridge-32` switch to the compilation line.
- The option `-Mvect` now supports the suboption `simd[{128|256}]` which specifies to vectorize using SIMD instructions and data, either 128 bits or 256 bits wide, on processors where there is a choice.

  From the `Fortran|Optimization|Vectorization` property page select the appropriate vectorization type. For more details, refer to "New or Modified Properties," on page 14.
- **PGI Accelerator** compilers and **CUDA Fortran** include these modifications and enhancements:
  - The default CUDA Toolkit version is now 3.2; the previous default was 3.1. For how to target a specific toolkit, refer to "PGI Accelerator and CUDA Fortran Enhancements," on page 9
  - Added support for the CUDA 4.0 Toolkit. Some features require this toolkit.

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

    ```
    CUDA Driver Version 4000
    ```
  - A `cublas` emulation library is now provided fro CUDA C++ for x86. For details on linking in this library, see the simple CUBLAS SDK example. A `cublas` module is also provided for PGI CUDA Fortran and PGI Accelerator Fortran use.
  - In CUDA Fortran, the print statement is allowed in attributes(global) and attributes(device) routines.

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

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

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

    If `SUB` is an overloaded interface to two or more routines, one of which has a matching dummy argument with the CUDA Fortran DEVICE attribute and the other does not, the generic procedure matching process will prefer the routine with the argument that has the DEVICE attribute.
  - In CUDA Fortran global routines, shared memory arrays may now have nonstatic size. The size may come from the blockdim members, from arguments or device variables. The launch of the kernel must specify enough space to hold all automatic shared memory arrays.

    Example:

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

11.5 Updates and Additions

- **PGI Accelerator** and **CUDA Fortran** include functionality that controls flush-to-zero mode for floating point computations in GPU code. PVF has added these properties to support this functionality:

  Fortran | Language | CUDA Fortran Flush to Zero
  : Use this property to control flush-to-zero mode for floating point computations in the GPU code generated for CUDA Fortran kernels.

  Fortran | Target Accelerators | NVIDIA: Flush to Zero
  : When Target NVIDIA Accelerator is set to `Yes`, use this property to control flush-to-zero mode for floating point computations in the GPU code generated for PGI Accelerator model compute regions.

  For more information on these properties, refer to "New or Modified Properties," on page 14.

11.4 Updates and Additions

- Support for Visual Studio 2010 SP1 is included in this release.
- Microsoft Open Tools 10 remains the default as it has been since the 10.6 release. Microsoft Open Tools 9 is no longer included as a backup.
- Additional Fortran modules are now available, including:
  - A Fortran Module to declare interfaces to many of the routines in the standard C math library `libm`.
  - A Fortran module to declare interfaces to many of the CUDA device builtin routines.

  For more information on these modules, refer to "Fortran Modules," on page 9.
- **PGPROF** has a number of enhancements in this release:
  - Reorganized menu items are now more closely aligned with standard menu functionality.
  - User preference settings are now saved on a per-user basis on Linux and Mac, and a per-user per-system basis on Windows.
  - User preference settings are now saved on exit by default.
  - Changing font size is now fully supported in all tabs. Changing font type is supported in all tabs except Compiler Feedback.
- The **PGI Accelerator** and **CUDA Fortran** include these enhancements:
  - New API routines access minimal profiling information for accelerator regions and kernels.
  - A new memory management routine for CUDA Fortran, `cudaMemGetInfo`, returns the amount of free and total memory available (in bytes) for allocation on the device.

  For more information on these routines, refer to "PGI Accelerator and CUDA Fortran Enhancements," on page 9

11.3 Updates and Additions

- New PVF properties include runtime properties that are compilation-only flags.
  - Use the `Fortran|Runtime|Check Array Bounds` property to enable array bounds checking at runtime.
  - Use the `Fortran|Runtime|Check Pointers` property to perform runtime checks for pointers that are dereferenced while initialized to null.
  - Use the `Fortran|Runtime|Check Stack` property to perform runtime stack checks for available space in the prologue of a function and before the start of a parallel region.
- New PVF properties include linker properties that specify stack allocation size.
  - Use the `Linker|General|Stack Reserve Size` property to specify the total number of bytes for stack allocation in virtual memory.
  - Use the `Linker|General|Stack Commit Size` property to specify the total number of bytes for stack allocation in physical memory.
- **PGPROF** has a number of enhancements in this release:
  - The PGPROF option `-text` is re-enabled in this release.
  - Reorganized menu items are now more closely aligned with standard menu functionality.
  - User preference settings are now saved on a per-user basis on Linux and Mac, and a per-user per-system basis on Windows.
  - User preference settings are now saved on exit by default.
  - Changing font size is now fully supported in all tabs. Changing font type is supported in all tabs except Compiler Feedback.

Updates and Additions prior to 11.3

PGI Visual Fortran Additions

PGI Visual Fortran (PVF) 2011 includes these enhancements:

- PVF 2011 includes a property that supports showing PTXAS informational messages. Use the `Fortran|Language|CUDA Fortran Keep PTXAS Info` property to show PTXAS informational messages during compilation.
- PVF 2011 includes a property that supports specifying the CUDA 3.2 Toolkit. Use the `Fortran|Language|CUDA Fortran Toolkit` property to specify version 3.2 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.

General Enhancements

- A new option `-nomp` is available which disables the `-mp` option that is enabled by default for linking.

- Behavior of the `-tp` flags and -m32/-m64 flags without a bit-length suffix. These flags now use the current bit width rather than defaulting to 32-bit.
- Enhanced error checks for mixing 32-bit and 64-bit targets, multiple 32-bit targets, and whether the 32-bit or 64-bit compilers are not installed.
- Added support for PTXAS informational messages. To add `-Mcuda=ptxinfo` to the compilation line and show PTXAS informational messages during compilation, use the property: `Fortran | Language | CUDA Fortran Keep PTXAS`.
- Added support for the CUDA 3.2 Toolkit.
- The PGPROF option `-text` has been disabled in this release. It may be re-enabled in a future release.

Fortran 2003 features

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

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

where *tbp-name-list* is a list of the specific type-bound procedures to be included in the generic set.

You can use these statements for named generics as well as for operators and assignments.

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

Associate Construct
: Associates a name either with a variable or with the value of an expression for the duration of a block. When the block is executed, the `associate-name` remains associated with the variable or retains the value specified, taking its type, type parameters, and rank from the association.

For more information and an example, refer to the PGI Visual Fortran User's Guide.

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

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

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

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

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

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

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

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

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

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

SIGN= specifier
: Provides the ability to control the optional plus characters in formatted numeric output.

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

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

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

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

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

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

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

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

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

Fortran Modules

PGI 2011 includes additional Fortran modules:

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

```
use libm
```

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

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

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

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

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

PGI Accelerator and CUDA Fortran Enhancements

PGI Accelerator x64+GPU native Fortran 95/03 and C99 compilers, and **CUDA Fortran** now support the CUDA 4.0 Toolkit. PGI continues to ship CUDA 3.2 and CUDA 3.1 Toolkit with the PGI compilers and tools, and you may leave it on your system if it exists from a previous installation. The default is CUDA 3.2.

The default toolkit for 11.8 is CUDA 3.2. Thus, CUDA Fortran host programs should be recompiled, as the PGI 11.8 CUDA Fortran runtime libraries are not compatible with previous CUDA Fortran releases.

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

In PGI Accelerator:

Use the property: Fortran | Target Accelerators | NVIDIA: CUDA Toolkit

When Target NVIDIA Accelerator is set to `Yes`, you can specify the version of the NVIDIA CUDA Toolkit targeted by the compilers.

`Default`: The compiler selects the default CUDA Toolkit version.
`4.0`: Specifies use of toolkit version 4.0.
`3.2`: Specifies use of toolkit version 3.2.
`3.1`: Specifies use of toolkit version 3.1.
`3.0`: Specifies use of toolkit version 3.0.
`2.3`: Specifies use of toolkit version 2.3.

Selecting one of these properties is equivalent to adding the associated switch to the PVF compilation and link lines:

```
-ta=nvidia[:cuda2.3 | cuda3.0 | cuda3.1 | cuda3.2 | cuda4.0]
```

For CUDA Fortran:

Use the property: `Fortran | Language | CUDA Fortran Toolkit`

When Enable CUDA Fortran is set to `Yes`, you can specify the version of the CUDA Toolkit targeted by the compilers.

`Default`: The compiler selects the default CUDA Toolkit version.
`4.0`: Specifies use of toolkit version 4.0.
`3.2`: Specifies use of toolkit version 3.2.
`3.1`: Specifies use of toolkit version 3.1.
`3.0`: Specifies use of toolkit version 3.0.
`2.3`: Specifies use of toolkit version 2.3.

Selecting one of these properties is equivalent to adding the associated switch to the PVF compilation and link lines:

```
-Mcuda[=cuda2.3 | cuda3.0 | cuda3.1 | cuda3.2 | cuda4.0]
```

Additional PGI Accelerator Runtime Routines

Table 2.1 contains a list of the PGI Accelerator model runtime routines available in Release 11.5. For a complete description of these routines, refer to Chapter 4, "PGI Accelerator Compilers Reference" of the *PGI Compiler Reference Manual*.

Table 2.1. Accelerator Runtime Library Routines

This Runtime Library Routine...	Does this...
acc_allocs	Returns the number of arrays allocated in data or compute regions.
acc_bytesalloc	Returns the total bytes allocated by data or compute regions.
acc_bytesin	Returns the total bytes copied in to the accelerator by data or compute regions.
acc_bytesout	Returns the total bytes copied out from the accelerator by data or compute regions.

This Runtime Library Routine...	Does this...
acc_copyins	Returns the number of arrays copied in to the accelerator by data or compute regions.
acc_copyouts	Returns the number of arrays copied out from the accelerator by data or compute regions.
acc_disable_time	Tells the runtime to start profiling accelerator regions and kernels, if it is not already doing so.
acc_enable_time	Tells the runtime to start profiling accelerator regions and kernels, if it is not already doing so.
acc_exec_time	Returns the number of microseconds spent on the accelerator executing kernels.
acc_free	Frees memory allocated on the attached accelerator. [C only]
acc_frees	Returns the number of arrays freed or deallocated in data or compute regions.
acc_get_device	Returns the type of accelerator device used to run the next accelerator region, if one is selected.
acc_get_device_num	Returns the number of the device being used to execute an accelerator region.
acc_get_free_memory	Returns the total available free memory on the attached accelerator device.
acc_get_memory	Returns the total memory on the attached accelerator device.
acc_get_num_devices	Returns the number of accelerator devices of the given type attached to the host.
acc_init	Connects to and initializes the accelerator device and allocates control structures in the accelerator library.
acc_kernels	Returns the number of accelerator kernels launched since the start of the program.
acc_malloc	Allocates memory on the attached accelerator. [C only]
acc_on_device	Tells the program whether it is executing on a particular device.
acc_regions	Returns the number of accelerator regions entered since the start of the program.
acc_set_device	Tells the runtime which type of device to use when executing an accelerator compute region.
acc_set_device_num	Tells the runtime which device of the given type to use among those that are attached.
acc_shutdown	Tells the runtime to shutdown the connection to the given accelerator device, and free up any runtime resources.
acc_total_time	Returns the number of microseconds spent in accelerator compute regions and in moving data for accelerator data regions.

Memory Management in CUDA

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

The syntax for `cudaMemGetInfo` is:

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

Declaring Interfaces to CUDA Device Builtin Routines

A Fortran module is available to declare interfaces to many of the CUDA device builtin routines.

To access this module, add this line to your Fortran program:

```
use cudadevice
```

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

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

New or Modified Compiler Options

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

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

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

  For example, if you have the 32-bit driver on your path, then `-tp shanghai` defaults to 32-bit; while if you are using the 64-bit driver, it defaults to 64-bit. For example, `-tp p7 -m64`, `-m64 -tp p7`, `-tp p7-64`, `-m64 -tp p7-64`, and `-tp p7-64 -m64` all mean the same thing.
- `–ta=nvidia(,nvidia_suboptions),host` is a switch associated with the PGI Accelerator compilers. `–ta` defines the target architecture.

  In release 2011, the `nvidia_suboptions` include:

analysis	Perform loop analysis only; do not generate GPU code.
cc10, cc11, cc12, cc13, cc20	Generate code for compute capability 1.0, 1.1, 1.2, 1.3, or 2.0 respectively.
cuda2.3 or 2.3	Specify the CUDA 2.3 version of the toolkit.

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

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

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

ptxinfo	Show PTXAS informational messages during compilation.

New or Modified Runtime Library Routines

PGI 2011 supports new runtime library routines associated with the PGI Accelerator compilers. For more information, refer to the "Using an Accelerator" chapter of the PGI Visual Fortran User's Guide.

New or Modified Properties

PVF 11.8 contains these new and modified properties:

Fortran | Optimization | Vectorization

Use this property to specify the type of vectorization to perform.

- **Default**: Accepts the default vectorization.
- **Enable Vectorization**: Enables vectorization by adding the `-Mvect` switch to the PVF compilation and link lines.
- **Vectorize using SSE instructions**: Enables vectorization using SSE instructions by adding the `-Mvect=sse` switch to the PVF compilation line.
- **Vectorize using SIMD instructions**: Enables vectorization using SIMD instructions and data, by adding the `-Mvect=simd` switch to the PVF compilation line.
- **Vectorize using 128-bit SIMD instructions**: Enables vectorization using SIMD 128-bit instructions and data, by adding the `-Mvect=simd:128` switch to the PVF compilation line.
- **Vectorize using 256-bit SIMD instructions**: Enables vectorization using SIMD 256-bit instructions and data, by adding the `-Mvect=simd:256` switch to the PVF compilation line.

Fortran | Language | CUDA Fortran Flush to Zero

Use this property to control flush-to-zero mode for floating point computations in the GPU code generated for CUDA Fortran kernels.

- **Default**: Accepts the default handling of floating point computations in the GPU code generated for CUDA Fortran kernels.
- **Yes**: Enables flush-to-zero mode by adding the `-Mcuda=flushz` switch to the PVF compilation and link lines.
- **No**: Disables flush-to-zero mode by adding the `-Mcuda=noflushz` switch to the PVF compilation and link lines.

Fortran | Target Accelerators | NVIDIA: Flush to Zero

When Target NVIDIA Accelerator is set to `Yes`, use this property to control flush-to-zero mode for floating point computations in the GPU code generated for PGI Accelerator model compute regions.

- **Default**: Accepts the default handling of floating point computations in the GPU code generated for CUDA Fortran kernels.
- **Yes**: Enables flush-to-zero mode by adding the `-ta=nvidia:flushz` switch to the PVF compilation and link lines.

- **No**: Disables flush-to-zero mode by adding the `-ta=nvidia:noflushz` switch to the PVF compilation and link lines.

New or Modified Tools Support

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

PGPROF

PGI 2011 includes several new and enhanced performance profiling features:

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

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

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

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

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

PGDBG

PGI 2011 includes several new and enhanced performance debugging features:

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

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

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

64-bit Java Runtime Environment requirement

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

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*.

Fortran 2003 Support Overview

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

I/O Support

These I/O capabilities are available in F2003:

- New specifier enhancements include:
  - SIGN= Specifier
  - NEXTREC, NUMBER, RECL, SIZE of all integer kinds
  - Round I/O specifier through the ROUND=specifier clause or through use of `RU`, `RD`, `RZ`, `RN`, `RC`, and `RP` edit descriptors.
  - SIZE of any integer kind in read/write statements
  - IOSTAT kind in all I/O statements
  - New specifier in WRITE statement: DELIMITER
  - New specifiers in READ statement: BLANK and PAD
  - New specifiers in INQUIRE statement: DECIMAL, POS, PENDING

- New I/O-related intrinsics include:
  - NEW_LINE
  - IS_IOSTAT_END
  - IS_IOSTAT_EOR
- Miscellaneous I/O capabilities include:
  - New decimal comma specifier descriptors, such as `DC` and `DP`
  - Asynchronous I/O
  - IMPORT and WAIT statements
  - ASYNCHRONOUS attribute and statement
  - I/O of Inf and NaN
  - I/O keyword encoding
  - Length of names and statements enhanced
  - Error message (ERRMSG) on ALLOCATE and DEALLOCATE
  - STOP statement warns about FP execution
  - Access = 'stream'

Data-related Enhancements

These data-related enhancements are available in F2003:

- allocatable scalars
- Generic and derived type may have the same name
- ABSTRACT types and interfaces
- VOLATILE attribute and statement
- MAX and MIN intrinsics take character parameter
- Structure and array Constructors
- Mixed component accessibility
- Deferred character length and deferred type parameters
- Typed allocation, including typed allocation for unlimited polymorphic entities
- Sourced allocation for deferred character, non-polymorphic, polymorphic types, and unlimited polymorphic entities
- Procedure pointers
- IEEE modules including IEEE_ARITHMETIC, IEEE_EXCEPTIONS, and IEEE_FEATURES. The IEEE_ARITHMETIC module allows operations on large arrays.

- IMPORT statement
- PROTECTED attribute and statement
- move_alloc() function
- Rename user-defined operators
- Pointer reshaping
- Square brackets

Object-Oriented Programming Enhancements

These object-oriented programming enhancements are available in F2003:

- Classes and inheritance association
- New intrinsics including EXTENDS_TYPE_OF and SAME_TYPE_AS
- Attributes including PASS, NOPASS, PRIVATE, PUBLIC and NON_OVERRIDABLE
- ASSOCIATE and SELECT TYPE constructs, including the SELECT TYPE construct for unlimited polymorphic entities
- Type-bound, deferred type-bound procedures and generic type-bound procedures
- Type extension (not polymorphic)
- Polymorphic entities and unlimited polymorphic entities
- Parameterized derived types
- ABSTRACT types
- PRIVATE statement for type bound procedures
- Type uses CONTAINS declaration
- Final Subroutines

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

  Final subroutines are only available in derived types and must be module procedures with a single dummy argument. These dummy arguments shall be nonoptional, nonpointer, nonallocatable, nonpolymorphic, and of the same type as the derived type being defined. Moreover, the dummy argument shall not have the `INTENT(OUT)` or `VALUE` attribute. All length type parameters must be assumed, that is, the length value must be '*'.

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

Interoperability with C Enhancements

These features for interoperability with C are available in F2003:

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

Miscellaneous F2003 Enhancements

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

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

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 32-bit Windows platform, these entries include a line containing $(PGIToolsDir) on 32-bit Windows systems or $(PGIToolsDir32) on 64-bit Windows systems. To change the compilers used for the 32-bit Windows 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 2011 is not compatible with previous releases. If you use Tools | Options to target a release prior to 2011, you cannot use PVF to debug. Instead, use the `-V` method described earlier in this chapter to select an alternate compiler.

Chapter 4. Distribution and Deployment

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

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 PVF 2011 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 32-bit Windows system, the redistributable directory is:

```
C:\Program Files\PGI\win32\11.8\REDIST
```

- On a 64-bit Windows system, there are two redistributable directories:

```
C:\Program Files\PGI\win64\11.8\REDIST
C:\Program Files (x86)\PGI\win32\11.8\REDIST
```

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 2011 includes the latest version, version 10, of the Microsoft Open Tools.

The Microsoft Open Tools directory contains a subdirectory named REDIST. PGI 2011 licensees may redistribute the files contained in this directory in accordance with the terms of the associated license agreements.

Note

On Windows, runtime libraries built for debugging (e.g. `msvcrtd` and `libcmtd`) are not included with PGI Visual Fortran. When a program is linked with `-g` for debugging, the standard non-debug versions of both the PGI runtime libraries and the Microsoft runtime libraries are always used. This limitation does not affect debugging of application code.

Chapter 5. Troubleshooting Tips and Known Limitations

This chapter contains information about known limitations, documentation errors, and corrections that have occurred to PVF 2011. 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 on pgroup.com 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.
- 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 on 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.

- Take extra care when using `-Mprof` with PVF runtime library DLLs. To build an executable for profiling, use of the static libraries is recommended. The static libraries are used by default in the absence of `-Bdynamic`.
- The `–i8` option can make programs incompatible with the bundled ACML library, MPI, and ACML; use of any INTEGER*8 array size argument can cause failures with these libraries. Visit developer.amd.com to check for compatible libraries.
- Using `–Mpfi` and `–mp` together is not supported. The `–Mpfi` flag disables `–mp` at compile time, which can cause runtime 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 32-bit Windows, 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 version in which it was fixed.

Chapter 6. Contact Information

You can contact The Portland Group at:

The Portland Group
STMicroelectronics, Inc.
Two Centerpointe Drive
Lake Oswego, OR 97035 USA

The PGI User Forum is monitored by members of the PGI engineering and support teams as well as other PGI customers. The forum newsgroups may contain answers to commonly asked questions. Log in to the PGI website to access the forum:

www.pgroup.com/userforum/index.php

Or contact us electronically using any of the following means:

Fax	+1-503-682-2637
Sales	sales@pgroup.com
Support	trs@pgroup.com
WWW	www.pgroup.com

All technical support is by email or submissions using an online form at www.pgroup.com/support. Phone support is not currently available.

Many questions and problems can be resolved at our frequently asked questions (FAQ) site at www.pgroup.com/support/faq.htm.

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