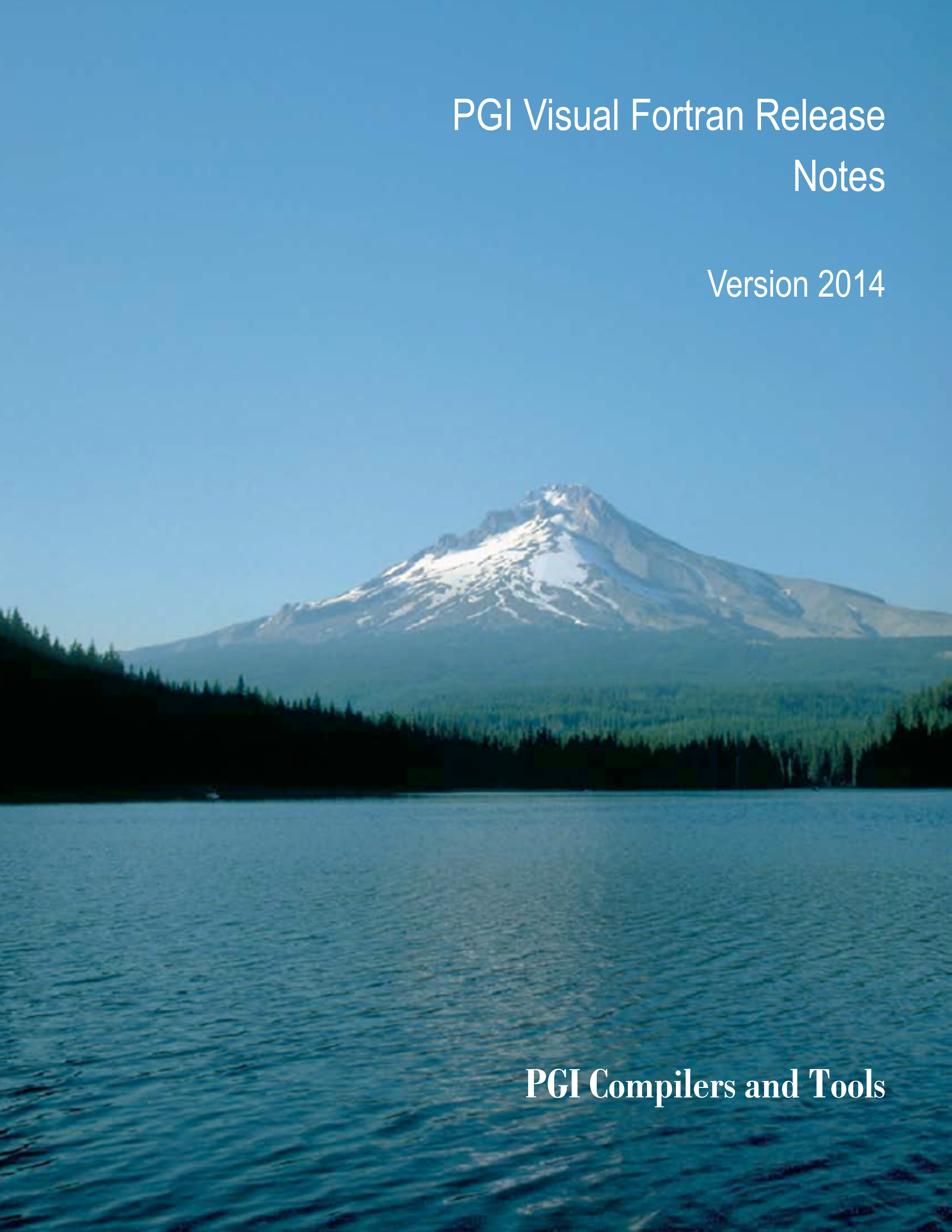

# PGI Visual Fortran Release Notes

Version 2014

PGI Compilers and Tools

# TABLE OF CONTENTS

# LIST OF TABLES

# Chapter 1.
# PVF RELEASE OVERVIEW

Welcome to Release 2014 of PGI Visual Fortran®, a set of Fortran compilers and development tools for 32-bit and 64-bit Windows integrated with Microsoft® Visual Studio.

This document describes the new features of the PVF IDE interface, differences in the PVF 2014 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 available in the PVF Installation Guide for this release.

## 1.1. Product Overview

PVF is integrated with several versions of Microsoft Visual Studio. Currently, Visual Studio 2008, 2010, 2012, and 2013 are supported. Throughout this document, "PGI Visual Fortran" refers to PVF integrated with any of the four supported versions of Visual Studio. Similarly, "Microsoft Visual Studio" refers to Visual Studio 2008, VS 2010, VS 2012, and VS 2013. When it is necessary to distinguish among the products, the document does so.

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 license 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, Fortran extensions to the Visual Studio 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 2014 of PGI Visual Fortran includes the following components:

- PGFORTRAN OpenMP and auto-parallelizing Fortran 2003 compiler.

- PGF77 OpenMP and auto-parallelizing FORTRAN 77 compiler.
- PVF Visual Studio integration components.
- AMD Core Math Library (ACML), version 5.24.0 for Windows x64 and version 4.4.0 for 32-bit Windows.
- OpenACC and CUDA Fortran tools and libraries necessary to build executables for Accelerator GPUs, when the user's license supports these optional features.
- PVF documentation.

If you do not already have Microsoft Visual Studio on your system, be sure to get the PVF installation package that contains the Visual Studio 2013 Shell.

## 1.2. Microsoft Build Tools

PVF on all Windows systems includes Microsoft Open Tools. On some systems (Windows XP, Windows Server 2003, Windows Server 2008), these files are all the additional tools and libraries required to compile, link, and execute programs on Windows. On other systems (Windows 2008 R2, Windows 7, Windows 8, Windows 8.1, Windows Server 2012), these files are required in addition to the files Microsoft provides in the Windows 8.1 SDK.

## 1.3. Terms and Definitions

This document contains a number of terms and definitions with which you may or may not be familiar. If you encounter an unfamiliar term in these notes, please refer to the online glossary at http://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, AMD Bulldozer, and AMD Piledriver 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, as well as Ivy Bridge and Haswell processors.

# Chapter 2.
# NEW AND MODIFIED FEATURES

This section provides information about the new and modified features of Release 2014 of PGI Visual Fortran.

## 2.1. What's New in Release 2014

### 14.9 Updates and Additions

- PGI Accelerator Features and Enhancements:
    - Integrated CUDA 6.5 Toolkit for Windows (Server 2008 R2 and later). Refer to CUDA Fortran Toolkit Issues for more details.
    - An **`acc routine`** directive with no parallelism clause (**`gang`**, **`worker`** or **`vector`**) will be treated as if the **`seq`** clause were present.
    - When compiling for NVIDIA Tesla targets (using **`-ta=tesla`** or **`-acc`** without the **`-ta`** flag) with the **`-Mcuda`** flag on the link line, the program will use the default CUDA stream zero for synchronous OpenACC data transfers and kernel launches. This allows OpenACC synchronous data transfers and kernels to interact properly with CUDA synchronous operations. Without the **`-Mcuda`** option, the program will create a CUDA stream even for synchronous operations, avoiding the serialization associated with CUDA stream zero.
- A number of problems are corrected in this release. Refer to http://www.pgroup.com/support/release_tprs.htm for a complete and up-to-date table of technical problem reports fixed in recent releases of PGI compilers and tools. This table contains a summary description of each problem as well as the version in which it was fixed.

### 14.7 Updates and Additions

- PGI Accelerator Features and Enhancements:
    - Support for CUDA managed data in CUDA Fortran; refer to the CUDA Fortran Programming Guide and Reference for details.

  - Expanded OpenACC 2.0 Features
    - Fortran module variables in OpenACC declare directives
    - Full support for the `atomic` directive
    - The `wait` clause on OpenACC directives is now supported.
    - The `async` clause on the `wait` directive is now supported.
    - When specifying a particular CUDA toolkit version on the command line, if that version is not available in the compiler installation, the compiler will now fail with an error message instead of giving a warning and compiling only for the host.
  - Improved accelerator code generation for nested loops
  - Support for debugging module scope variable in CUDA Fortran
- New Language Features:
  - New F90 pointer optimizations
- Other Features and Enhancements:
  - CPU code vectorization enhancements

## 14.6 Updates and Additions

- A number of problems are corrected in this release. Refer to http://www.pgroup.com/support/release_tprs.htm for a complete and up-to-date table of technical problem reports fixed in recent releases of PGI compilers and tools. This table contains a summary description of each problem as well as the version in which it was fixed.

## 14.4 Updates and Additions

- PGI Accelerator Features and Enhancements:
  - Expanded OpenACC 2.0 Features
    - Loop directive collapse clause on deeply nested loops
    - Parallel directive firstprivate clause
    - Fortran derived type member arrays in data clauses
    - Partial support for Fortran atomic directives
    - Fortran common block names in OpenACC data clauses
  - CUDA Fortran support for CUDA 5.5 batched cuBLAS routines
  - Integrated CUDA 6.0 Toolkit
- PGI Multi-core Features and Enhancements:
  - Support for new AVX2 instructions available on the latest Haswell CPUs from Intel
  - Updated Windows assembler

### 14.2 and 14.3 Updates and Additions

- A number of problems are corrected in these releases. Refer to http://www.pgroup.com/support/release_tprs.htm for a complete and up-to-date table of technical problem reports fixed in recent releases of PGI compilers and tools. This table contains a summary description of each problem as well as the version in which it was fixed.

### 14.1 Updates and Additions

- PGI Visual Fortran fully integrated with Visual Studio 2013, supported on Windows 8.1, including support for OpenACC and CUDA Fortran on NVIDIA Tesla GPUs, and full native OpenACC on AMD Radeon GPUs.
- These Windows releases are supported in PGI 2014, but will be deprecated in PGI 2015.
  - Windows XP
  - Windows Server 2003
  - Windows Server 2008
- Updates to PGI OpenACC Fortran/C/C++ compilers, include:
  - Support for CUDA 5.5 and NVIDIA Kepler K40 GPUs
  - Support for AMD Radeon GPUs and APUs
  - Native compilation for NVIDIA and AMD GPUs
  - Ability within CUDA Fortran to generate dwarf information and debug on the host, device, or both
  - Additional OpenACC 2.0 features supported, including procedure calls (routine directive), unstructured data lifetimes; create and device_resident clauses for the Declare directive; ability to call CUDA Fortran atomic functions on NVIDIA; and complete run-time API support.
  - PGI Unified Binary for OpenACC programs across NVIDIA and AMD GPUs

  For more information, refer to PGI Accelerator Enhancements.
- Full Fortran 2003 and incremental Fortran 2008 features including long integers, recursive I/O, type statement for intrinsic types, ISO_FORTRAN_ENV and ISO_C_BINDING module updates as well as support for F2008 **`contiguous`** attribute and keyword.

  For more information, refer to New or Modified Fortran Functionality.
- LAPACK linear algebra math library for shared-memory vector and parallel processors, version 3.4.2, supporting Level 3 BLACS (Basic Linear Algebra Communication Subroutines) for use with PGI compilers. This library is provided in both 64-bit and 32-bit versions for AMD64 or Intel 64 CPU-based installations running Linux, OS X, or Windows.
- Support for the latest Operating Systems including Windows 8.1.
- The `–ta` and `–acc` flags include additional options and functionality. The `–tp` flag functionality is now primarily for processor selection.

  For more information, refer to New or Modified Compiler Options.

- PGI Visual Fortran fully integrated with Visual Studio 2013, supported on Windows 8.1, including support for OpenACC and CUDA Fortran on NVIDIA Tesla GPUs, and full native OpenACC on AMD Radeon GPUs.

## 2.2. New and Modified Compiler Options

Release 2014 supports a number of new command line options as well as new keyword suboptions for existing command line options.

### 2.2.1. Required Suboption

The default behavior of the OpenACC compilers has changed in 14.1 from previous releases. The OpenACC compilers now issue a compile-time error if accelerator code generation fails. You can control this behavior with the `required` suboption.

In previous releases, the compiler would issue a warning when accelerator code generation failed. Then it would generate code to run the compute kernel on the host. This previous behavior generates incorrect results if the compute kernels are inside a data region and the host and device memory values are inconsistent.

`-acc=required`, `-ta=tesla:required`, and `-ta=radeon:required` are the defaults.

You can enable the old behavior by using the `norequired` suboption with either of the `-ta` or `-acc` flags.

### 2.2.2. Accelerator Options

The `-ta=nvidia` option is deprecated in PGI 2014. Users are urged to change their build commands and makefiles to use `-ta=tesla` in place of `-ta=nvidia`.

The `-acc` option enables the recognition of OpenACC directives. In the absence of any explicit `-ta` option, `-acc` implies `-ta=tesla,host`.

#### `-ta` Option

The `-ta` option defines the target accelerator and the type of code to generate. This flag is valid for Fortran, C, and C++ on supported platforms.

**Syntax**

```
-ta=tesla(:tesla_suboptions),radeon(:radeon_suboptions),host
```

There are three major suboptions:

`tesla(:tesla_suboptions)`

`radeon(:radeon_suboptions)`

`host`

**Default**

The default is `-ta=tesla,host`.

### Select Tesla Accelerator Target

Use the `tesla(:tesla_suboptions)` option to select the Tesla accelerator target and, optionally, to define the type of code to generate.

In the following example, Tesla is the accelerator target architecture and the accelerator generates code for compute capability 3.0:

```
$ pgfortran -ta=tesla:cc30
```

The following table lists and briefly defines the suboptions for the `-ta=tesla` flag.

Table 1 –ta=tesla Suboptions

| Use this suboption... | To indicate this... |
|---|---|
| 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. |
| cc1x | Generate code for the lowest 1.x compute capability possible. |
| cc1+ | Is equivalent to cc1x, cc2x, cc3x. |
| cc20 | Generate code for compute capability 2.0. |
| cc2x | Generate code for the lowest 2.x compute capability possible. |
| cc2+ | Is equivalent to cc2x, cc3x. |
| cc30 | Generate code for compute capability 3.0. |
| cc35 | Generate code for compute capability 3.5. |
| cc3x | Generate code for the lowest 3.x compute capability possible. |
| cc3+ | Is equivalent to cc3x. |
| [no]debug | Enable[disable] debug information generation in device code. |
| [no]lineinfo | Enable[disable] line information generation in device code. |
| fastmath | Use routines from the fast math library. |
| fermi | Is equivalent to cc2x. |
| fermi+ | Is equivalent to cc2+. |
| [no]flushz | Enable[disable] flush-to-zero mode for floating point computations in the GPU code. |
| keep | Keep the kernel files. |

| Use this suboption... | To indicate this... |
|---|---|
| kepler | Is equivalent to cc3x. |
| kepler+ | Is equivalent to cc3+. |
| llvm | Generate code using the llvm-based back-end. |
| maxregcount:n | Specify the maximum number of registers to use on the GPU. |
| nofma | Do not generate fused multiply-add instructions. |
| noL1 | Prevent the use of L1 hardware data cache to cache global variables. |
| pin | Set default to pin host memory. |
| [no]rdc | Generate [do not generate] relocatable device code. |
| [no]required | Generate [do not generate] a compiler error if accelerator device code cannot be generated. |

### Select Radeon Accelerator Target

Use the `radeon(:radeon_suboptions)` option to select the Radeon accelerator target and, optionally, to define the type of code to generate.

In the following example, Radeon is the accelerator target architecture and the accelerator generates code for Radeon Cape Verde architecture:

```
$ pgfortran -ta=radeon:capeverde
```

The following table lists and briefly defines the suboptions for the `–ta=radeon` flag.

## Table 2 –ta=radeon Suboptions

| Use this suboption... | To indicate this... |
|---|---|
| buffercount:n | Set the maximum number of OpenCL buffers in which to allocate data. |
| capeverde | Generate code for Radeon Cape Verde architecture. |
| keep | Keep the kernel files. |
| llvm | Generate code using the llvm-based back-end. |
| [no]required | Generate [do not generate] a compiler error if accelerator device code cannot be generated. |
| spectre | Generate code for Radeon Spectre architecture. |
| tahiti | Generate code for Radeon Tahiti architecture. |

### Host Option

Use the `host` option to generate code to execute OpenACC regions on the host.

The `–ta=host` flag has no suboptions.

#### Multiple Targets

Specifying more than one target, such as `-ta=tesla,radeon` generates code for multiple targets. When host is one of the multiple targets, such as `-ta=tesla,host`, the result is generated code that can be run with or without an attached accelerator.

### 2.2.3. Relocatable Device Code

An `rdc` option is available for the `-ta=tesla` and `-Mcuda` flags that specifies to generate relocatable device code. Starting in PGI 14.1 on Linux and in PGI 14.2 on Windows, the default code generation and linking mode for Tesla-target OpenACC and CUDA Fortran is `rdc`, relocatable device code.

You can disable the default and enable the old behavior and non-relocatable code by specifying any of the following: `-ta=tesla:nordc`, `-Mcuda=nordc`, or by specifying any 1.x compute capability or any Radeon target.

### 2.2.4. –tp Modifications

The `-tp` switch now truly indicates the target processor. In prior releases a user could use the `-tp` flag to also indicate use of 32-bit or 64-bit code generation. For example, the `-tp shanghai-32` flag was equivalent to the two flags: `-tp shanghai` and `-m32`.

The `-tp` flag interacts with the `-m32` and `-m64` flags to select a target processor and 32-bit or 64-bit code generation. For example, specifying `-tp shanghai -m32` compiles 32-bit code that is optimized for the AMD Shanghai processor, while specifying `-tp shanghai -m64` compiles 64-bit code.

Specifying `-tp shanghai` without a `-m32` or `-m64` flag compiles for a 32-bit target if the PGI 32-bit compilers are on your path, and for a 64-bit target if the PGI 64-bit compilers are on your path.

## 2.3. New and Modified Fortran Functionality

PVF 2014 contains additional Fortran functionality such as full Fortran 2003 and incremental Fortran 2008 features including long integers, recursive I/O, type statement for intrinsic types, as well as ISO_FORTRAN_ENV and ISO_C_BINDING module updates and support for F2008 `contiguous` attribute and keyword.

### 2.3.1. Contiguous Pointers

PGI 2014 supports the `contiguous` attribute as well as the `is_contiguous` intrinsic inquiry function.

#### contiguous Attribute

Here is an example of a declaration using the contiguous keyword:

```
real*4, contiguous, pointer, dimension(:,:) :: arr1_ptr, arr2_ptr, arr3_ptr
```

It is the responsibility of the programmer to assure proper assignment and use of contiguous pointers. Contiguous pointers can result in improved performance, such as this example of using contiguous pointers as the arguments to the matmul intrinsic function.

```
    arr3_ptr = matmul(arr1_ptr,arr2_ptr)
```

### is_contiguous Intrinsic Inquiry Function

The `is_contiguous()` intrinsic function takes a pointer argument and returns a value of type logical. It returns true if the pointer is associated with a contiguous array section, false otherwise.

## 2.4. New and Modified Runtime Library Routines

PGI 2014 supports new runtime library routines associated with the PGI Accelerator compilers.

For more information, refer to *Using an Accelerator* in the User's Guide.

## 2.5. PGI Accelerator Enhancements

### 2.5.1. OpenACC Directive Summary

PGI now supports the following OpenACC directives:

#### Parallel Construct

Defines the region of the program that should be compiled for parallel execution on the accelerator device.

#### Kernels Construct

Defines the region of the program that should be compiled into a sequence of kernels for execution on the accelerator device.

#### Data Directive

Defines data, typically arrays, that should be allocated in the device memory for the duration of the data region, whether data should be copied from the host to the device memory upon region entry, and copied from the device to host memory upon region exit.

#### Enter Data and Exit Data Directives

The Enter Data directive defines data, typically arrays, that should be allocated in the device memory for the duration of the program or until an exit data directive that deallocates the data, and whether data should be copied from the host to the device memory at the enter data directive.

The Exit Data directive defines data, typically arrays, that should be deallocated in the device memory, and whether data should be copied from the device to the host memory.

### Host_Data Construct

Makes the address of device data available on the host.

### Loop Directive

Describes what type of parallelism to use to execute the loop and declare loop-private variables and arrays and reduction operations. Applies to a loop which must appear on the following line.

### Combined Parallel and Loop Directive

Is a shortcut for specifying a loop directive nested immediately inside an accelerator parallel directive. The meaning is identical to explicitly specifying a parallel construct containing a loop directive.

### Combined Kernels and Loop Directive

Is a shortcut for specifying a loop directive nested immediately inside an accelerator kernels directive. The meaning is identical to explicitly specifying a kernels construct containing a loop directive.

### Cache Directive

Specifies array elements or subarrays that should be fetched into the highest level of the cache for the body of a loop. Must appear at the top of (inside of) the loop.

### Declare Directive

Specifies that an array or arrays are to be allocated in the device memory for the duration of the implicit data region of a function, subroutine, or program.

Specifies whether the data values are to be transferred from the host to the device memory upon entry to the implicit data region, and from the device to the host memory upon exit from the implicit data region.

Creates a visible device copy of the variable or array.

### Update Directive

Used during the lifetime of accelerator data to update all or part of a host memory array with values from the corresponding array in device memory, or to update all or part of a device memory array with values from the corresponding array in host memory.

### Routine Directive

Used to tell the compiler to compile a given procedure for an accelerator as well as the host. In a file or routine with a procedure call, the routine directive tells the implementation the attributes of the procedure when called on the accelerator.

As of PGI 14.9, an **`acc routine`** directive with no parallelism clause (**`gang`**, **`worker`** or **`vector`**) will be treated as if the **`seq`** clause were present.

### Wait Directive

Specifies to wait until all operations on a specific device async queue or all async queues are complete.

For more information on each of these directives and which clauses they accept, refer to the *Using an Accelerator* section in the PGI Visual Fortran User's Guide.

## 2.5.2. CUDA Toolkit Version

The PGI Accelerator x64+accelerator compilers with OpenACC and CUDA Fortran compilers support the CUDA 6.0 toolkit as the default. The compilers and tools also support the CUDA 6.5 Toolkit.

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

### For OpenACC Directives

Use the property: `Fortran | Target Accelerators | Tesla: CUDA Toolkit`

When Target NVIDIA Tesla 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, which is 6.0 for this release.
>
> `6.0`: Specifies use of toolkit version 6.0.
>
> `6.5`: Specifies use of toolkit version 6.5.

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

```
-ta=tesla[:cuda6.0 | cuda6.5]
```

### For CUDA Fortran Construct

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, which is 6.0 for this release.
>
> `6.0`: Specifies use of toolkit version 6.0. This is the default.
>
> `6.5`: Specifies use of toolkit version 6.5.

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

```
-Mcuda[=cuda6.0 | cuda6.5]
```

### 2.5.3. Fortran Derived Types in OpenACC

Static and allocatable arrays of derived type have long been supported with the PGI Accelerator compilers.

```
module mpoint
type point
    real :: x, y, z
end type
type(point) :: base(1000)
end module

subroutine vecaddgpu( r, n )
 use mpoint
 type(point) :: r(:)
 integer :: n
 !$acc parallel loop present(base) copyout(r(:))
 do i = 1, n
  r(i)%x = base(i)%x
  r(i)%y = sqrt( base(i)%y*base(i)%y + base(i)%z*base(i)%z )
  r(i)%z = 0
 enddo
end subroutine
```

PGI 14.4 and later releases include support for array members of derived types, including static arrays and allocatable arrays within a derived type. In either case, the entire derived type must be placed in device memory, by putting the derived type itself in an appropriate data clause. For this release, the derived type variable itself must appear in a data clause, at least a **`present`** clause, for any compute construct that directly uses the derived type variable.

```
module mpoint
type point
    real :: base(1000)
    integer :: n
    real, allocatable, dimension(:) :: x, y, z
end type

type(point) :: A
end module

subroutine vecaddgpu()
 integer :: i
 !$acc parallel loop copyin(A) copyout(A%x,A%y,A%z)
 do i = 1, n
  A%x(i) = A%base(i)
  A%y(i) = sqrt( A%base(i) )
  A%z(i) = 0
 enddo
end subroutine
```

In this example, the derived type **`A`** is copied to the device, which copies the static array member **`A%base`** and the scalar **`A%n`**. The allocatable array members **`A%x`**, **`A%y`** and **`A%z`** are then copied to the device. The derived type variable **`A`** should be copied before its allocatable array members, either by placing the derived type in an earlier data clause, or by copying or creating it on the device in an enclosing data region or dynamic data lifetime. If the derived type is not present on the device when the allocatable array members are copied, the accesses to the allocatable members, such as **`A%x(i)`**, on the device will be invalid, because the hidden pointer and descriptor values in the derived type variable will not get updated.

Be careful when copying derived types containing allocatable members back to the host. On the device, the allocatable members will get updated to point to device memory. If the whole derived type gets copied back to the host, the allocatable members will be invalid on the host.

When creating or copying a derived type on the device, the whole derived type is allocated. There is no support for allocating a subset of a derived type, or only allocating space for a single member.

Derived types and allocatable members can be managed using dynamic data directives as well:

```
module mpoint
 type point
    integer :: n
    real, dimension(:), allocatable :: x, y, z
 end type
contains
 subroutine move_to_device( A )
  type(point) :: A
  !$acc enter data copyin(A)
  !$acc enter data create(A%x, A%y, A%z)
 end subroutine

 subroutine move_off_device( A )
  type(point) :: A
  !$acc exit data copyout(A%x, A%y, A%z)
  !$acc exit data delete(A)
 end subroutine
end module

subroutine vecaddgpu( A, base )
 use mpoint
 type(point) :: A
 real :: base(:)
 integer :: i
 !$acc parallel loop present(A,base)
 do i = 1, n
  A%x(i) = base(i)
  A%y(i) = sqrt( base(i) )
  A%z(i) = 0
 enddo
end subroutine
```

Arrays of derived type, where the derived type contains allocatable members, have not been tested and should not be considered supported for this release. That important feature will be included in an upcoming release.

## 2.5.4. OpenACC declare data directive for global and Fortran module variables

The 14.7 release supports the OpenACC **`declare`** directive with the **`copyin`**, **`create`** and **`device_resident`** clauses for Fortran module variables, for Tesla-target GPUs. This is primarily for use with the OpenACC **`routine`** directive and separate compilation. The data in the **`declare`** clauses are statically allocated on the device when the program attaches to the device. Data in a **`copyin`** clause will be initialized from the host data at that time. A program attaches to the device when it reaches its first data or compute construct, or when it calls the OpenACC **`acc_init`** routine.

In Fortran, module fixed-size variables and arrays, and module allocatable arrays which appear in **`declare`** directives at module scope will be available globally on the CPU as well as in device code. Module allocatable arrays that appear in a **`declare create`**, **`declare copyin`** or **`declare device_resident`** will be allocated in host memory as well as in device memory when they appear in an allocate statement. The compiler manages the actual pointer to the data and a descriptor that contains array lower and upper bounds for each dimension, and the device copy of the pointer will be set to point to the array in device memory.

The following example module contains one fixed size array and an allocatable array, both appearing in a **`declare create`** clause. The static array **`xstat`** will be available at any time inside accelerator compute regions or routines.

```
module staticmod
 integer, parameter :: maxl = 100000
 real, dimension(maxl) :: xstat
 real, dimension(:), allocatable :: yalloc
 !$acc declare create(xstat,yalloc)
end module
```

This module may be used in another file that allocates the **`yalloc`** array. When the allocatable array **`yalloc`** is allocated, it will be allocated both in host and device memory, and will then be available at any time in accelerator compute regions or routines.

```
subroutine allocit(n)
 use staticmod
 integer :: n
 allocate( yalloc(n) )
end subroutine
```

In another module, these arrays may be used in a compute region or in an accelerator routine:

```
module useit
 use staticmod
contains
 subroutine computer( n )
  integer :: n
  integer :: i
  !$acc parallel loop
   do i = 1, n
    yalloc(i) = iprocess( i )
   enddo
 end subroutine
 real function iprocess( i )
  !$acc routine seq
  integer :: i
  iprocess = yalloc(i) + 2*xstat(i)
 end function
end module
```

# 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.

## 3.1. 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, –V13.8 causes the compiler driver to invoke the 13.8 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.

## 3.2. 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 2014 is not compatible with previous releases. If you use `Tools | Options` to target a release prior to 2014, you cannot use PVF to debug. Instead, use the `–V` method described earlier in this section 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 section addresses how to effectively distribute applications built using PGI compilers and tools.

## 4.1. 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.

### 4.1.1. PGI Redistributables

PGI Visual Fortran includes redistributable directories which contain all of the PGI dynamically linked libraries that can be re-distributed by PVF 2014 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\14.9\REDIST
  ```

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

  ```
  C:\Program Files\PGI\win64\14.9\REDIST
  C:\Program Files(x86)\PGI\win32\14.9\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.

### 4.1.2. Microsoft Redistributables

PGI Visual Fortran includes Microsoft Open Tools, the essential tools and libraries required to compile, link, and execute programs on Windows. PVF 2014 installed on Windows 7, 8, 8.1, and Server 2012 includes the latest version, version 12, of the Microsoft Open Tools. PVF 2014 installed on Windows XP, Server 2003, and Server 2008 includes the Microsoft Open Tools version 10.

The Microsoft Open Tools directory contains a subdirectory named `REDIST`. PGI 2014 licensees may redistribute the files contained in this directory in accordance with the terms of the associated license agreements.

> 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 section contains information about known limitations, documentation errors, and corrections. Wherever possible, a work-around 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/faq.htm

## 5.1. PVF IDE Limitations

The issues in this section are related to IDE limitations.

- 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.
- 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.

## 5.2. 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, refer to 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.

## 5.3. PGI Compiler Limitations

- 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`.
- 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.
- The `–i8` option can make programs incompatible with the ACML library; use of any INTEGER*8 array size argument can cause failures with these libraries. Visit developer.amd.com to check for compatible ACML libraries.
- ACML 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.

## 5.4. CUDA Toolkit Issues

### Targeting a CUDA Toolkit Version

- The CUDA 6.0 Toolkit is set as the default in PGI 14.9. To use the CUDA 6.0 Toolkit, first download the CUDA 6.0 driver from NVIDIA at www.nvidia.com/cuda.
- You can compile with the CUDA 6.5 Toolkit either by adding the option `-ta=tesla:cuda6.5` to the command line or by adding `set CUDAVERSION=6.5` to the `siterc` file.
- `pgaccelinfo` prints the driver version as the first line of output. For a 6.5 driver, it prints:

```
CUDA Driver Version 6050
```

### CUDA 6.5 Toolkit Limitations

- PGI installation packages for the Windows 7/8/2012 operating systems contain both the CUDA 6.0 and 6.5 Toolkits but the packages for the Windows XP/2003/2008 operating systems contain CUDA 6.0 only.
- The CUDA 6.5 Toolkit is not supported on Windows Server 2012 although it is supported on Windows Server 2012 R2.

- The CUDA 6.5 Toolkit is not supported on Windows Vista.

## 5.5. OpenACC Issues

This section includes known limitations in PGI's support for OpenACC directives.

PGI plans to support these features in a future release, though separate compilation and extern variables for Radeon will be deferred until OpenCL 2.0 is released.

### ACC routine directive limitations

- The `routine` directive has limited support on AMD Radeon. Separate compilation is not supported on Radeon, and selecting `-ta=radeon` disables `rdc` for `-ta=tesla`.
- The `bind` clause on the `routine` directive is not supported.
- The `nohost` clause on the `routine` directive is not supported.
- Reductions in procedures with `acc routine` are not fully supported.
- Fortran assumed-shape arguments are not yet supported.

### Clause Support Limitations

- The `device_type` clause is not supported on any directive.

## 5.6. Corrections

A number of problems are corrected in this release. Refer to www.pgroup.com/support/release_tprs.htm for a complete and up-to-date table of technical problem reports, TPRs, fixed in recent releases of the PGI compilers and tools. This table contains a summary description of each problem as well as the version in which it was fixed.

# Chapter 6.
# CONTACT INFORMATION

You can contact PGI at:

20400 NW Amberwood Drive Suite 100
Beaverton, OR 97006

Or electronically using any of the following means:

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

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:

http://www.pgroup.com/userforum/index.php

Many questions and problems can be resolved by following instructions and the information available at our frequently asked questions (FAQ) site:

http://www.pgroup.com/support/faq.htm

All technical support is by e-mail or submissions using an online form at:

http://www.pgroup.com/support

Phone support is not currently available.

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

## Notice

## Trademarks

## Copyright