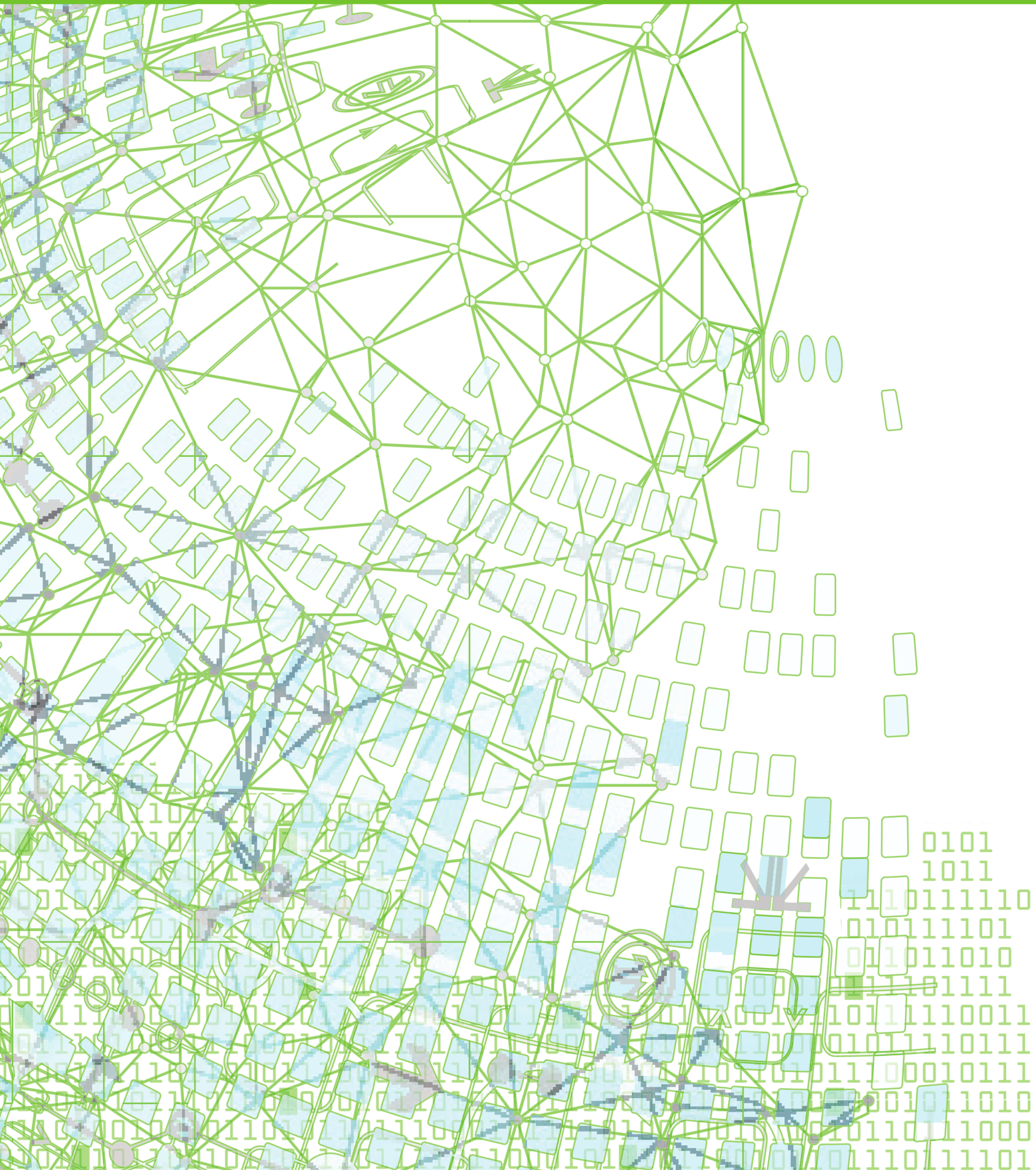
PGI® COMPILERS & TOOLS

PVF RELEASE NOTES

Version 2017



NVIDIA.

TABLE OF CONTENTS

LIST OF TABLES

Chapter 1. PVF RELEASE OVERVIEW

Welcome to Release 2017 of PGI Visual Fortran®, a set of Fortran compilers and development tools for Windows integrated with Microsoft® Visual Studio.

This document describes the new features of the PVF IDE interface, differences in the PVF 2017 compilers and tools from previous releases, and late-breaking information not included in the standard product documentation.

PGI Release 2016 version 16.4 and newer includes FlexNet license daemons updated to version 11.13.1.3. This update addresses a FlexNet security vulnerability, https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-8277. These new license daemons also work with older PGI releases. We recommend all users update their license daemons—see the licensing FAQ, https://www.pgicompilers.com/support/faq.htm for more information. This FlexNet update also requires you to update your PGI FlexNet license keys, https://www.pgicompilers.com/license/pin_manage.php?view=keys to a new format. Older keys are incompatible.

1.1. Product Overview

PVF is integrated with two versions of Microsoft Visual Studio. Currently, Visual Studio 2013 and 2015 are supported. Throughout this document, "PGI Visual Fortran" refers to PVF integrated with either of the two supported versions of Visual Studio. Similarly, "Microsoft Visual Studio" refers to Visual Studio 2013 and VS 2015. 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 2017 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.
- 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 2015 shell.

1.2. Microsoft Build Tools

PVF on all Windows systems includes Microsoft Open Tools. These files are required in addition to the files Microsoft provides in the Windows 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 PGI online glossary located at https://www.pgroup.com/support/definitions.htm.

These two terms are used throughout the documentation to reflect groups of processors:

Intel 64
A 64-bit Intel Architecture 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.

AMD64
A 64-bit processor from AMD™ incorporating 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.

Chapter 2. NEW AND MODIFIED FEATURES

This section provides information about the new and modified features of Release 2017 of PGI Visual Fortran.

2.1. What's New in Release 2017

17.7 Updates and Additions

- PGI Accelerator Compilers
  - Added support for Tesla V100 GPUs in PGI OpenACC and CUDA Fortran. Based on the new NVIDIA Volta GV100 GPU, Tesla V100 offers more memory bandwidth, more streaming multiprocessors, next generation NVLink and new microarchitectural features that add up to better performance and programmability. For OpenACC and CUDA Fortran programmers, Tesla V100 offers improved hardware support and performance for CUDA Unified Memory features on both x86-64 and OpenPOWER processor-based systems. Use the sub-option `cc70` with the `-ta=tesla` or `-Mcuda` compile- and link-time options to generate code for the Tesla V100 GPUs; the CUDA 9 toolkit must be used when targeting Volta GPUs.
  - Added initial support for the CUDA 9 toolkit. Requirements for using CUDA 9 with the PGI 17.7 compilers and profiler:
    - Install the release candidate (RC) or early access (EA) version of the CUDA 9 toolkit.
    - Specify the location of the CUDA 9 toolkit using the compiler option CUDAROOT; for example, if CUDA 9 is installed in `/opt/cuda-9.0`, add **`CUDAROOT=/opt/cuda-9.0`** when compiling, linking, or invoking the profiler.
    - Use the sub-option `cuda9.0` with the `-ta=tesla` or `-Mcuda` compile- and link-time options.

  - Added Beta support for automatic deep copy of Fortran derived types. This feature allows you to port applications with modern deeply nested data structures to Tesla GPUs using OpenACC. The PGI 17.7 compilers allow you to list aggregate Fortran data objects in OpenACC **`copy`**, **`copyin`**, **`copyout`** and **`update`** directives to move them between host and device memory including traversal and management of pointer-based objects within the aggregate data object. When enabled, full deep copy ensures that, when moving Fortran variables of derived type from host to device or device to host, the entire data structure, including pointers and allocatable arrays, is copied back and forth between host and device, or device and host, memory. To enable deep copy, use the `deepcopy` sub-option to `-ta=tesla`. Two things to note: polymorphic data types are not supported, and the presence of overlapping pointers may cause runtime errors.
- All PGI Compilers
  - Improved inlining with the `-Minline` option. You may see different functions inlined with this release than you observed in prior releases. In some cases, compilation may take noticeably longer to complete because of an increase in the number of functions inlined. Added the `smallsize:number` sub-option tomanage `-Minline` which you can use to always inline functions of size smaller than number regardless of other size limits.
  - Added a new feature for program exploration called Program Analysis Summary Output (PASO). PASO enables the exporting of information gathered from the internal representation (IR) of various compilation stages in an organized, human readable format. This data forms a detailed summary of the program's internal structure, spanning not just one but all the files that make up a project. To enable PASO, compile and link with the `-Msummary` option. To find out more about how PASO works and how you might leverage it, refer to Program Analysis Using Summary Output, https://www.pgicompilers.com/blogs/posts/paso.htm. PASO is available on all platforms supported by PGI except macOS.
- Profiler

  The following new profiler features are enabled only when invoking the profiler with CUDA 9. To use CUDA 9 with PGI 17.7, start the profiler with the **`CUDAROOT`** option set to the CUDA 9 installation location. For example, if CUDA 9 is installed in `/opt/cuda-9.0`, add **`CUDAROOT=/opt/cuda-9.0`** when launching the profiler.

  - Enhanced unified memory profiling:
    - Added association between unified memory events/CPU page faults and the source location of memory allocation.
    - Added new unified memory profiling events for page thrashing, throttling and remote map.
    - Added support for switching between segment and non-segment timelines.

        - Added filtering of events based on the virtual address, migration reason or page fault access type.
    - Added an OpenACC details table summarizing each recorded event.
    - Added support for the profiling of cooperative kernel launches.
    - Added NVLink events to the timeline.
    - Added memory throughput to the NVLink topology diagram.
    - Increased the options available for multi-hop remote profiling.
    - Added support for OpenACC profiling on all multicore systems.
- Documentation

  Added HTML-formatted documentation for all PGI compilers and tools online at https://www.pgicompilers.com. The new format should make it easier to use any Internet-connected system or device to search or reference PGI documentation. PDF versions of all documents also remain available.

17.5 Updates and Additions

- CUDA Toolkit
    - This version includes an updated nvlink linker utility that addresses an issue with the link ordering of object files on Linux.
- Operating Systems
    - Added support for Ubuntu 17.04 and RHEL 6.9 Linux distributions.

17.4 Updates and Additions

- PGI Accelerator Compilers
    - Added support for atomic add and atomic subtract in device code for Fortran's single precision complex data types.
    - Added device versions of isfinite, isnan, isinf, and round.
    - Added support for CUDA Fortran warp-vote operations.
    - Changed the impact of `-g` on the optimization level used when compiling device code. For host code, `-g` sets the compiler's optimization level to zero unless another optimization level is specified using a `-O` option. For device code, the same is now true. For details about generating debug information for device code, refer to the PGI Compiler Reference Guide's section on DWARF Debugging Formats.
    - Updated the version of nvlink to 8.0.73; this version of the device linker allows an arbitrary ordering of object files on its command line.

17.3 Updates and Additions

The 17.3 release was the first PGI Visual Fortran release in 2017; all updates and additions listed for PGI 17.1 apply to 17.3 as well unless otherwise noted.

IMPORTANT: The PGI Visual Fortran 2017 release no longer supports 32-bit development. PVF 2017 only supports 64-bit operating systems and does not include the ability to compile 32-bit applications for execution on either 32-bit or 64-bit operating systems. With PGI 2017, you can still view the Win32 solution platform within a PVF project, but you cannot build or debug using the Win32 solution platform. If you have not yet migrated your projects from the Win32 solution platform to the x64 solution platform, consider doing so by using PVF 16.10 or earlier versions which support building both the Win32 and x64 versions for comparison.

17.1 Updates and Additions

- All PGI Compilers
  - Updated how floating point divides are computed to guarantee results will be uniform for scalar and vector operations. This change can cause numerical differences between PGI 17.1 and previous PGI releases. In rare cases, this change can cause an increase in execution time.
  - Improved inlining with the `-Minline` option. You may see different functions inlined with this release than you observed in prior releases. In some cases, compilation may take longer to complete because of an increase in the number of functions inlined. Some of the `-Minline` sub-options, which you can use to control inlining, have also changed from previous releases:
    - Added `totalsize:n` to limit inlining to the total size of n where n is the size of the combined program units on a per file basis.
    - Changed `size:n` to `maxsize:n` which allows inlining only of functions smaller than approximately n lines. The compilers silently convert the previous `size:n` to `maxsize:n`.
    - Dropped `levels:n` which limited inlining to n levels of functions. The compilers silently ignore `levels:n`.
  - Added the `-cpp` option as an alias for the `-Mpreprocess` option.
- PGI Accelerator OpenACC Compilers
  - Dropped support for CUDA 7.0.
  - Changed the default version of the CUDA Toolkit used by the compilers from CUDA 7.0 to 7.5. The CUDA Toolkit 8.0 can be used instead of 7.5 by adding the sub-option `cuda8.0` to the `-ta=tesla` or `-Mcuda` compile- and link-time options.

    - Removed compute capability 2.0 (Fermi) from the compilers' default set of compute capabilities. The `cc20` sub-option to the `-ta=tesla` and `-Mcuda` options is deprecated.
    - Improved data management within the OpenACC **`cache`** directive.
    - Added support for additional OpenACC 2.5 features:
        - Changed the behavior of the **`exit data`** directive to decrement the dynamic reference count.
        - Added the new optional **`finalize`** clause to set the dynamic reference count to zero.
        - Added the **`if_present`** clause to the **`update`** directive which changes the behavior when data is not present from a runtime error to a no-op.
        - Added new **`init`**, **`shutdown`**, and **`set`** directives.
        - Added new API routines to get and set the default async queue value.
        - Added support for the new definition of **`routine bind`** clause.
        - Updated the value of **`_OPENACC`** to 201510.

      With the exception of nested parallelism, declare link, and adding restrictions to **`cache`** clause variable references to variables within a cached region, OpenACC 2.5 feature support is complete in PGI 2017.
- Other Features and Additions
    - Added support for Microsoft Windows Server 2016.
- Deprecations and Eliminations
    - PGI 2017 supports 64-bit operating systems only. Compiling 32-bit applications for execution on either 32-bit or 64-bit operating systems is no longer supported on any platform.
    - Dropped support for the CUDA 7.0 toolkit.

2.2. New and Modified Compiler Options

Release 2017 supports new and updated command line options and keyword suboptions.

Added the following options:

- -cpp is now an alias for -Mpreprocess.
- [dis]allows variadic macros.

Changed the following -Minline sub-options:

- Added totalsize:n to limit inlining to the total size of n.
- maxsize:n replaces size:n to prevent inlining of functions bigger than n. The compilers silently convert the previous size:n to maxsize:n.
- Removed levels:n which limited inlining to n levels of functions. The compilers silently ignore levels:n.

2.3. Updates to CUDA Toolkit Support

As of PGI 17.1, CUDA 7.5 and the CUDA 8.0 are supported on Linux and Windows. CUDA 7.5 is the default.

Support for PGI Accelerator features for macOS including CUDA Fortran, OpenACC and CUDA-x86 is no longer available as of the PGI 2017 release.

Targeting a CUDA Toolkit Version

- The CUDA 7.5 Toolkit is set as the default in PGI 17.7. To use the CUDA 7.5 Toolkit, first download the CUDA 7.5 driver from NVIDIA at http://www.nvidia.com/cuda.
- You can compile with the CUDA 8 Toolkit either by adding the option -ta=tesla:cuda8.0 to the command line or by adding **`set DEFCUDAVERSION=8.0`** to the `siterc` file. To use the CUDA 8.0 Toolkit, you must download and install the CUDA 8.0 driver from NVIDIA.
- `pgaccelinfo` prints the driver version as the first line of output. For a 7.5 driver, it prints:
  CUDA Driver Version 7050

2.4. OpenACC

CUDA Unified Memory

In the PGI 17.7 release, the use of CUDA Unified Memory for allocatable data moved from a Beta feature to production. This feature, described in detail in the OpenACC and CUDA Unified Memory, https://www.pgroup.com/lit/articles/insider/v6n2a4.htm *PGInsider* article, is available with the Linux/x86-64 and Linux/OpenPOWER compilers. It is supported on Linux/x86-64 using both the default PGI code generator and the Beta LLVM-based code generator. To enable this feature, add the option `-ta=tesla:managed` to the compiler and linker command lines.

In the presence of `-ta=tesla:managed`, all C/C++/Fortran explicit allocation statements in a program unit are replaced by equivalent "managed" data allocation calls that place the data in CUDA Unified Memory. Managed data share a single address for CPU/GPU and data movement between CPU and GPU memories is implicitly handled by the CUDA driver. Therefore, OpenACC data clauses and directives are not needed for "managed" data. They are essentially ignored, and in fact can be omitted.

When a program allocates managed memory, it allocates host pinned memory as well as device memory thus making allocate and free operations somewhat more expensive and data transfers somewhat faster. A memory pool allocator is used to mitigate the overhead of the allocate and free operations. The pool allocator is enabled by default for `-ta=tesla:managed` or `-ta=tesla:pinned`. In the PGI 17.7 release, the presence of `-Mcuda` disables the pool allocator; we are working on lifting that restriction in an upcoming release.

Data movement of managed data is controlled by the NVIDIA CUDA GPU driver; whenever data is accessed on the CPU or the GPU, it could trigger a data transfer if the last time it was accessed was not on the same device. In some cases, page thrashing may occur and impact performance. An introduction to CUDA Unified Memory is available on Parallel Forall.

This feature has the following limitations:

- Use of managed memory applies only to dynamically-allocated data. Static data (C static and extern variables, Fortran module, common block and save variables) and function local data is still handled by the OpenACC runtime. Dynamically allocated Fortran local variables and Fortran allocatable arrays are implicitly managed but Fortran array pointers are not.
- Given an allocatable aggregate with a member that points to local, global or static data, compiling with `-ta=tesla:managed` and attempting to access memory through that pointer from the compute kernel will cause a failure at runtime.
- C++ virtual functions are not supported.
- The `-ta=tesla:managed` compiler option must be used to compile the files in which variables are allocated, even if there is no OpenACC code in the file.

This feature has the following additional limitations when used with NVIDIA Kepler GPUs:

- Data motion on Kepler GPUs is achieved through fast pinned asynchronous data transfers; from the program's perspective, however, the transfers are synchronous.
- The PGI runtime enforces synchronous execution of kernels when `-ta=tesla:managed` is used on a system with a Kepler GPU. This situation may result in slower performance because of the extra synchronizations and decreased overlap between CPU and GPU.
- The total amount of managed memory is limited to the amount of available device memory on Kepler GPUs.

This feature is not supported on NVIDIA Fermi GPUs.

CUDA Unified Memory Pool Allocator

Dynamic memory allocations are made using cudaMallocManaged(), a routine which has higher overhead than allocating non-unified memory using cudaMalloc(). The more calls to cudaMallocManaged(), the more significant the impact on performance.

To mitigate the overhead of cudaMallocManaged() calls, both `-ta=tesla:managed` and `-ta=tesla:pinned` use a CUDA Unified Memory pool allocator to minimize the number of calls to cudaMallocManaged(). The pool allocator is enabled by default. It can be disabled, or its behavior modified, using these environment variables:

Table 1 Pool Allocator Environment Variables

Environment Variable	Use
PGI_ACC_POOL_ALLOC	Disable the pool allocator. The pool allocator is enabled by default; to disable it, set PGI_ACC_POOL_ALLOC to 0.
PGI_ACC_POOL_SIZE	Set the size of the pool. The default size is 1GB but other sizes (i.e., 2GB, 100MB, 500KB, etc.) can be used. The actual pool size is set such that the size is the nearest, smaller number in the Fibonacci series compared to the provided or default size. If necessary, the pool allocator will add more pools but only up to the PGI_ACC_POOL_THRESHOLD value.
PGI_ACC_POOL_ALLOC_MAXSIZE	Set the maximum size for allocations. The default maximum size for allocations is 64B but another size (i.e., 100KB, 10MB, 250MB, etc.) can be used as long as it is greater than or equal to 16B.
PGI_ACC_POOL_ALLOC_MINSIZE	Set the minimum size for allocation blocks. The default size is 16B but other sizes can be used. The size must be greater than or equal to 16B.
PGI_ACC_POOL_THRESHOLD	Set the percentage of total device memory that the pool allocator can occupy. The default is set to 50% but other percentages can be used.

Multicore Support

PGI Accelerator OpenACC compilers support the option `-ta=multicore`, to set the target accelerator for OpenACC programs to the host multicore. This will compile OpenACC compute regions for parallel execution across the cores of the host processor or processors. The host multicore will be treated as a shared-memory accelerator, so the data clauses (**`copy`**, **`copyin`**, **`copyout`**, **`create`**) will be ignored and no data copies will be executed.

By default, `-ta=multicore` will generate code that will use all the available cores of the processor. If the compute region specifies a value in the **`num_gangs`** clause, the minimum of the **`num_gangs`** value and the number of available cores will be used. At runtime, the number of cores can be limited by setting the environment variable **`ACC_NUM_CORES`** to a constant integer value. If an OpenACC compute construct appears lexically within an OpenMP parallel construct, the OpenACC compute region will generate sequential code. If an OpenACC compute region appears dynamically within an OpenMP region or another OpenACC compute region, the program may generate many more threads than there are cores, and may produce poor performance.

The ACC_BIND environment variable is set by default with `-ta=multicore`; **`ACC_BIND`** has similiar behavior to **`MP_BIND`** for OpenMP.

The `-ta=multicore` option differs from the `-ta=host` option in that `-ta=host` generates sequential code for the OpenACC compute regions.

Default Compute Capability

The default compute capability list for OpenACC and CUDA Fortran compilation for NVIDIA Tesla targets is cc30, cc35, and cc50. If CUDA 8.0 is specified using the `cuda8.0` sub-option, cc60 is added to the default compute capability list. The generation of device code can be time consuming, so you may notice an increase in compile time. You can override the default by specifying one or more compute capabilities using either command-line options or an `rcfile`.

To change the default with a command-line option, provide a comma-separated list of compute capabilities to `-ta=tesla:` for OpenACC or `-Mcuda=` for CUDA Fortran.

To change the default with an `rcfile`, set the **`DEFCOMPUTECAP`** value to a blank-separated list of compute capabilities in the siterc file located in your installation's bin directory:

```
set DEFCOMPUTECAP=30 35 50;
```

Alternatively, if you don't have permissions to change the `siterc` file, you can add the **`DEFCOMPUTECAP`** definition to a separate `.mypgirc` file (`mypgi_rc` on Windows) in your home directory.

OpenACC 2.5

The PGI compilers implement OpenACC 2.5 as defined in *The OpenACC Application Programming Interface*, Version 2.5, August 2013, http://www.openacc.org, with the exception that these features are not yet supported:

- nested parallelism
- declare link
- enforcement of the new **`cache`** clause restriction that all references to listed variables must lie within the region being cached

Support for Profiler/Trace Tool Interface

PGI compilers support the OpenACC 2.5 version of the OpenACC Profiler/Trace Tools Interface. This is the interface used by the PGI profiler to collect performance measurements of OpenACC programs.

2.5. Runtime Library Routines

PGI 2017 supports runtime library routines associated with the PGI Accelerator compilers. For more information, refer to *Using an Accelerator* in the PGI Compiler User's Guide.

Chapter 3. SELECTING AN ALTERNATE COMPILER

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

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

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

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

The method to use depends on the situation.

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.

Warning: The debug engine in PVF 2017 is not compatible with previous releases. If you use `Tools | Options` to target a release prior to 2017, 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 Linux and Windows. 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 2017 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 64-bit Windows system, the redistributable directory is:

```
C:\Program Files\PGI\win64\17.7\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 2017 installed for Microsoft Visual Studio 2015 includes the latest version, version 14.0, of the Microsoft

Open Tools; PVF 2017 installed for Microsoft Visual Studio 2013 includes the previous version, version 12.0, of the Microsoft Open Tools in order to maintain compatibility with Visual C++ 2013.

The Microsoft Open Tools directory contains a subdirectory named `REDIST`. PGI 2017 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, please see the PGI frequently asked questions (FAQ) webpage at https://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.

- 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.

5.4. OpenACC Issues

This section includes known limitations in PGI's support for OpenACC directives. PGI plans to support these features in a future release.

ACC routine directive limitations

- Fortran assumed-shape arguments are not yet supported.

Clause Support Limitations

- Not all clauses are supported after the `device_type` clause.

5.5. Corrections

A number of problems are corrected in this release. Refer to the Technical Problem Reports webpage at https://www.pgroup.com/support/release_tprs.htm for a complete and up-to-date table 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: mailto: sales@pgroup.com
WWW: https://www.pgicompilers.com

The PGI User Forum, https://www.pgicompilers.com/userforum/index.php is monitored by members of the PGI engineering and support teams as well as other PGI customers. The forums contain answers to many commonly asked questions. Log in to the PGI website, https://www.pgicompilers.com/account/login.php" to access the forums.

Many questions and problems can be resolved by following instructions and the information available in the PGI frequently asked questions (FAQ), https://www.pgicompilers.com/support/faq.htm.

Submit support requests using the PGI Technical Support Request form, https://www.pgicompilers.com/support/support_request.php .