PGI® COMPILERS &TOOLS

PGI VISUAL FORTRAN RELEASE NOTES

Version 2019

NVIDIA

TABLE OF CONTENTS

Chapter 1.
PVF RELEASE OVERVIEW

> 💬 **PGI command-level compilers for Windows will continue to be enhanced and supported, but PGI Visual Fortran (PVF) is being deprecated. Existing PVF licensees can continue to use it indefinitely, but no new licenses will be issued. PVF technical support and new features will be discontinued at the end of the 2019 calendar year.**

> 💬 **Important** The PGI 2018 Release includes updated FlexNet license management software to address a security vulnerability. Users of any previous PGI release must update their FlexNet license daemons to enable PGI 18.1 and subsequent releases. See Third-Party Software Security Updates in What's New in PGI 2018 and our FlexNet Update FAQ for more information.

This chapter provides an overview of Release 2019 of PGI Visual Fortran®, a set of Fortran compilers and development tools for Windows integrated with Microsoft® Visual Studio.

1.1. Product Overview

PVF is integrated with Microsoft Visual Studio 2015. Throughout this document, "PGI Visual Fortran" refers to PVF integrated with VS 2015. Similarly, "Microsoft Visual Studio" refers to Visual Studio 2015. When it is necessary to distinguish further, 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 2019 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.

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 pgicompilers.com/definitions.

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

Intel 64
    64-bit Intel x86-64 CPUs including Intel Core processors, Intel Xeon Nehalem, Sandy Bridge, Ivy Bridge, Haswell, Broadwell and Skylake processors, and Intel Xeon Phi Knights Landing.

AMD64
    64-bit AMD™ x86-64 CPUs including Opteron and EPYC processors.

Chapter 2.
WHAT'S NEW IN PGI 2019

> PGI command-level compilers for Windows will continue to be enhanced and supported, but PGI Visual Fortran (PVF) is being deprecated. Existing PVF licensees can continue to use it indefinitely, but no new licenses will be issued. PVF technical support and new features will be discontinued at the end of the 2019 calendar year.

Welcome to Release 2019 of PGI Visual Fortran!

If you read only one thing about this PGI release, make it this chapter. It covers all the new, changed, deprecated, or removed features in PGI products released this year. It is written with you, the user, in mind.

Every PGI release contains user-requested fixes and updates. We keep a complete list of these fixed Technical Problem Reports online for your reference.

2.1. What's New in 19.7

OpenACC

The compiler will now issue an error message for OpenACC **data**, **enter data**, and **exit data** constructs with no data clauses. Previously, these had no effect. These are almost certainly programming errors, and the error message will help avoid such errors.

Tightened the conditions under which OpenACC **kernels** loops containing pointer references can be parallelized. This change may prevent the compiler from parallelizing a **kernels** loop it previously parallelized unsafely, which could result in a loss of performance. If you are certain the loop can be parallelized safely, changing **kernels** to **parallel** or adding the **independent** loop clause will instruct the compiler to override the safety check and parallelize the loop.

Changed how the compilers treat aggregate members (struct and derived type members) used in loop bounds or loop limits. Previously, the compiler would generally read the aggregate member from host memory. In 19.7, the compiler follows the OpenACC spec

more carefully by reading the device memory value of the aggregate member, except for values used in the limits of the outermost loop or loops collapsed with the outermost loop. This means that the value in device memory of any aggregate members used in loop limits must be up-to-date with the host memory values, or the generated code will behave differently.

Improved support for OpenACC **`host_data use_device`** constructs. The compilers now accept references to aggregate data types in the variable-list as defined in the OpenACC specification.

The CUDA 10.1 version bundled with PGI has been upgraded to CUDA 10.1 update 1. More information can be found in the CUDA 10.1 release notes.

CUDA Fortran

Added support to CUDA Fortran for the half precision floating point data type, real(2), as a first class type. This type, represented as IEEE binary16, gives the programmer the ability to declare and use half precision data for programming the Tensor Cores in select NVIDIA GPUs using CUDA Fortran. Real(2) support is available in Fortran compilers on Linux for x86 and OpenPOWER.

Improved performance of CUDA Fortran data transfers between host and device memory using assignment statements, for array sections that can be mapped onto cudaMemcpy2D.

Fortran

Added support for the Fortran 2008 **`g0`** editor descriptor.

Improved generation of DWARF debug information for allocatable arrays within modules.

Added support for allocations from multiple sources via the **`SOURCE`** argument (e.g. **`ALLOCATE(a, b, SOURCE=c)`**).

Extended **`MAXLOC`** and **`MINLOC`** intrinsics to accept the optional **`BACK`** argument.

Implemented **`NORM2`** intrinsic to calculate magnitude of a Euclidean vector.

Libraries

We have added two new optimized intrinsics: **`atan`** and **`atan2`** to our intrinsic math library for x86 processors. This includes both single and double precision as well as scalar and vector versions.

Deprecations

The PGI Debugger (pgdbg) is now deprecated. It will be discontinued at the end of the 2019 calendar year. For Linux users, PGI compilers are interoperable with the Allinea

DDT and Rogue Wave TotalView debuggers including their support for OpenACC, OpenMP and MPI debugging. On MacOS and Linux, the GNU debugger (gdb) can be used for basic debugging of PGI-generated code.

2.2. What's New in 19.5

All Compilers

The PGI 19.5 release contains all the new features found in PGI 19.4 and a few key updates for important user-reported problems.

2.3. What's New in 19.4

All Compilers

Version 19.4 of the PGI Compilers & Tools is a Community Edition release. New, refined and improved support for new features introduced in 2019 include: Tensor Core support in CUDA Fortran, LLVM-based code generation, full C++17 language support, support for printf() in OpenACC regions, support for CUDA 10.0 and 10.1, improved SIMD vectorization, GNU interoperability through GCC 8.1, pre-compiled Open MPI 3.1.3 libraries, support for the latest operating systems, and more. Please refer to the TPR list located at https://www.pgroup.com/support/release-tprs-2019.htm for a detailed summary of bug fixes.

2.4. What's New in 19.3

OpenACC and CUDA Fortran

Added support for the CUDA Toolkit version 10.1. Read more about support for CUDA Toolkits in CUDA Toolkit Versions.

2.5. What's New in 19.1

All Compilers

Upgraded FlexNet Publisher licensing to version 11.16.2 from 11.14.1.3 for x86-64. The newer daemons are backwards-compatible with previous versions of the PGI compilers. The reverse is not true. This release, and future releases, of the PGI compilers are not compatible with the older daemons and require version 11.16.2 at a minimum.

PCAST (PGI Compiler-Assisted Software Testing) now supports C and Fortran directives that are equivalent to the existing library API calls.

Added support for Windows Server 2019.

Fortran

Added support for the **`ERROR STOP`** statement.

Added run-time checks to enforce **`CONTIGUOUS`** behavior. These checks only occur in code running on the CPU; in particular, they do not occur in code running on a GPU.

Improved Fortran reshape performance, and handling of derived type c-binding for C++ objects.

OpenACC and CUDA Fortran

Added initial support for use of NVIDIA GPU Tensor Cores in CUDA Fortran programs.

The C and C++ compilers now include support for formatted output using **`printf()`** statements in OpenACC compute regions. Most common format specifiers for flag, width, precision, size and type are supported. Using **`printf()`** in OpenACC regions is useful for basic debugging and programmer-driven tracing during development and tuning of OpenACC applications on both multicore CPUs and GPUs.

Added support for Turing architecture with compute capability version 7.5.

CUDA 9.2 is now the default target version if there is no GPU or CUDA driver found on the system.

Removed support for legacy PGI Accelerator Directives. Attempting to compile code with such directives will now give an error instead of a warning.

Renamed the **`[no]llvm`** sub-option to **`-Mcuda`** and **`-ta=tesla`** to **`[no]nvvm`**.

Various compiler improvements, including bug fixes related to subkernels when compiling with **`-Mallocatable=03`** (default allocation behavior since 18.7), and improved scalar replacement optimizations.

Consolidated the accelerator runtime libraries by removing non-thread-safe versions previously used for serial execution.

Performance

Improved performance of several intrinsic functions for AVX2 and newer processors: `sin, cos, log, log10, cexp, acos, ains, atan, tanh`.

Improved performance of memory idiom functions: `mset, mcopy, mzero`.

Enhanced vectorizer to recognize more opportunities for SIMD code generation across all CPU targets.

Known Issues and Limitations

Passing **–M[no]llvm** to MPI wrappers (mpicc, mpifort, etc.) is not supported. Doing so will cause unresolved symbol errors and segmentation faults when compiling.

Using **–V\<version\>** with the PGI 2019 compilers to target PGI 2018 or earlier releases is not supported. This is a known limitation that is the result of the switch to using LLVM compilers as the default code generator for the PGI 2019 compilers.

Programs built with OpenMPI 3.1.3 that use only one MPI rank will hang if invoked directly. You must now run the program with **mpirun -np 1 ./executable** or set the environment variable **OMPI_MCA_ess_singleton_isolated=1**. This is a known and intended behavior change in the 3.1.3 version of OpenMPI.

OpenMP profiling is not supported for Fortran applications that use both OpenACC and OpenMP, and use critical sections, OpenMP lock API calls, or performs I/O. OpenACC profiling of these applications is still possible by disabling OpenMP profiling with the '**--openmp-profiling off**' option to pgprof.

Deprecations and Eliminations

CUDA 9.1 is no longer included as part of the PGI compilers installation package.

The PGI compilers installation packages no longer include pre-compiled versions of MPICH or MVAPICH.

The **pgf77** driver is deprecated. Use **pgfortran** to compile F77 Fortran.

2.6. New and Modified Compiler Options

Release 2019 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.

2.7. OpenACC

The following sections contain details about updates to PGI compiler support for OpenACC.

2.7.1. OpenACC Error Handling

The OpenACC specification provides a mechanism to allow you to intercept errors triggered during execution on a GPU and execute a specific routine in response before the program exits. For example, if an MPI process fails while allocating memory on the GPU, the application may want to call `MPI_Abort` to shut down all the other processes before the program exits. This section explains how to take advantage of this feature.

To intercept errors the application must give a callback routine to the OpenACC runtime. To provide the callback, the application calls `acc_set_error_routine` with a pointer to the callback routine.

The interface is the following, where `err_msg` contains a description of the error:

```
typedef void (*exitroutinetype)(char *err_msg);
extern void acc_set_error_routine(exitroutinetype callback_routine);
```

When the OpenACC runtime detects a runtime error, it will invoke the `callback_routine`.

> 💬 This feature is not the same as error recovery. If the callback routine returns to the application, the behavior is decidedly undefined.

Let's look at this feature in more depth using an example.

Take the MPI program below and run it with two processes. Process 0 tries to allocate a large array on the GPU, then sends a message to the second process to acknowledge the success of the operation. Process 1 waits for the acknowledgment and terminates upon receiving it.

```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#define N 2147483648

int main(int argc, char **argv)
{
  int rank, size;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  int ack;
  if(rank == 0) {
    float *a = (float*) malloc(sizeof(float) * N);

#pragma acc enter data create(a[0:N])
```

```
#pragma acc parallel loop independent
    for(int i = 0; i < N; i++) {
        a[i] = i *0.5;
    }
#pragma acc exit data copyout(a[0:N])
    printf("I am process %d, I have initialized a vector of size %ld bytes on
 the GPU. Sending acknowledgment to process 1.", rank, N);
    ack = 1;
    MPI_Send(&ack, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  } else if(rank == 1) {
    MPI_Recv(&ack, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("I am process %d, I have received the acknowledgment from process 0
 that data in the GPU has been initialized.\n", rank, N);
    fflush(stdout);
  }

  // do some more work

  MPI_Finalize();

  return 0;
}
```

We compile the program with:

```
$ mpicc -ta=tesla -o error_handling_mpi error_handling_mpi.c
```

If we run this program with two MPI processes, the output will look like the following:

```
$ mpirun -n 2 ./error_handling_mpi
Out of memory allocating -8589934592 bytes of device memory
total/free CUDA memory: 11995578368/11919294464
Present table dump for device[1]:
NVIDIA Tesla GPU 0, compute capability 3.7, threadid=1
...empty...
call to cuMemAlloc returned error 2: Out of memory

-------------------------------------------------------
Primary job terminated normally, but 1 process returned
a non-zero exit code.. Per user-direction, the job has been aborted.
-------------------------------------------------------
--------------------------------------------------------------------------
mpirun detected that one or more processes exited with non-zero status,
thus causing the job to be terminated.
```

Process 0 failed while allocating memory on the GPU and terminated unexpectedly with an error. In this case mpirun was able to identify that one of the processes failed, so it shut down the remaining process and terminated the application. A simple two-process program like this is straightforward to debug. In a real world application though, with hundreds or thousands of processes, having a process exit prematurely may cause the application to hang indefinitely. Therefore it would be ideal to catch the failure of a process, control the termination of the other processes, and provide a useful error message.

We can use the OpenACC error handling feature to improve the previous program and correctly terminate the application in case of failure of an MPI process.

In the following sample code, we have added an error handling callback routine that will shut down the other processes if a process encounters an error while executing on the GPU. Process 0 tries to allocate a large array into the GPU and, if the operation is successful, process 0 will send an acknowledgment to process 1. Process 0 calls the OpenACC function acc_set_error_routine to set the

function `handle_gpu_errors` as an error handling callback routine. This routine prints a message and calls `MPI_Abort` to shut down all the MPI processes. If process 0 successfully allocates the array on the GPU, process 1 will receive the acknowledgment. Otherwise, if process 0 fails, it will terminate itself and trigger the call to `handle_gpu_errors`. Process 1 is then terminated by the code executed in the callback routine.

```c
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#define N 2147483648


typedef void (*exitroutinetype)(char *err_msg);
extern void acc_set_error_routine(exitroutinetype callback_routine);

void handle_gpu_errors(char *err_msg) {
  printf("GPU Error: %s", err_msg);
  printf("Exiting...\n\n");
  MPI_Abort(MPI_COMM_WORLD, 1);
  exit(-1);
}


int main(int argc, char **argv)
{
  int rank, size;

  MPI_Init(&argc, &argv);

  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  int ack;
  if(rank == 0) {
    float *a = (float*) malloc(sizeof(float) * N);


    acc_set_error_routine(&handle_gpu_errors);


#pragma acc enter data create(a[0:N])
#pragma acc parallel loop independent
    for(int i = 0; i < N; i++) {
      a[i] = i *0.5;
    }
#pragma acc exit data copyout(a[0:N])
    printf("I am process %d, I have initialized a vector of size %ld bytes on
 the GPU. Sending acknowledgment to process 1.", rank, N);
    fflush(stdout);
    ack = 1;
    MPI_Send(&ack, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
  } else if(rank == 1) {
    MPI_Recv(&ack, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    printf("I am process %d, I have received the acknowledgment from process 0
 that data in the GPU has been initialized.\n", rank, N);
    fflush(stdout);
  }

  // more work

  MPI_Finalize();
```

```
  return 0;
}
```

Again, we compile the program with:

```
$ mpicc -ta=tesla -o error_handling_mpi error_handling_mpi.c
```

We run the program with two MPI processes and obtain the output below:

```
$ mpirun -n 2 ./error_handling_mpi
Out of memory allocating -8589934592 bytes of device memory
total/free CUDA memory: 11995578368/11919294464
Present table dump for device[1]:
NVIDIA Tesla GPU 0, compute capability 3.7, threadid=1
...empty...
GPU Error: call to cuMemAlloc returned error 2: Out of memory
Exiting...

--------------------------------------------------------------------
MPI_ABORT was invoked on rank 0 in communicator MPI_COMM_WORLD
with errorcode 1.
```

This time the error on the GPU was intercepted by the application which managed it with the error handling callback routine. In this case the routine printed some information about the problem and called `MPI_Abort` to terminate the remaining processes and avoid any unexpected behavior from the application.

2.7.2. Performance Impact of Fortran 2003 Allocatables

In the PGI 18.7 release, use of Fortran 2003 semantics for assignments to allocatables was made the default. This change applies to host and device code alike. Previously, Fortran 1995 semantics were followed by default. The change to Fortran 2003 semantics may affect performance in some cases where the `kernels` directive is used.

When the following Fortran allocatable array assignment is compiled using the Fortran 2003 specification, the compiler cannot generate parallel code for the array assignment; lack of parallelism in this case may negatively impact performance.

```
  real, allocatable, dimension(:) :: a, b
  allocate(a(100), b(100))
  a = 3.14

  !$acc kernels
  a = b
  !$acc end kernels
```

The example code can be modified to use an array section assignment instead; the compiler can parallelize the array section assignment and the lost performance is regained.

```
  a(:) = b(:)
```

2.8. CUDA Toolkit Versions

The PGI compilers use NVIDIA's CUDA Toolkit when building programs for execution on an NVIDIA GPU. Every PGI installation package puts the required CUDA Toolkit components into a PGI installation directory called `2019/cuda`.

An NVIDIA CUDA driver must be installed on a system with a GPU before you can run a program compiled for the GPU on that system. PGI products do not contain CUDA Drivers. You must download and install the appropriate CUDA Driver from NVIDIA. The CUDA Driver version must be at least as new as the version of the CUDA Toolkit with which you compiled your code.

The PGI tool `pgaccelinfo` prints the driver version as its first line of output. You can use it to find out which version of the CUDA Driver is installed on your system.

PGI 19.7 includes the following versions of the CUDA Toolkit:

▶ CUDA 9.2
▶ CUDA 10.0
▶ CUDA 10.1

You can let the compiler pick which version of the CUDA Toolkit to use or you can instruct it to use a particular version. The rest of this section describes all of your options.

If you do not specify a version of the CUDA Toolkit, the compiler uses the version of the CUDA Driver installed on the system on which you are compiling to determine which CUDA Toolkit to use. This auto-detect feature was introduced in the PGI 18.7 release; auto-detect is especially convenient when you are compiling and running your application on the same system. In the absence of any other information, the compiler will look for a CUDA Toolkit version in the PGI `2019/cuda` directory that matches the version of the CUDA Driver installed on the system. If a match is not found, the compiler searches for the newest CUDA Toolkit version that is not newer than the CUDA Driver version. If there is no CUDA Driver installed, the PGI 19.7 compilers fall back to the default of CUDA 9.2.

If the only PGI compiler you have installed is PGI 19.7, then:

▶ If your CUDA Driver is 10.1, the compilers use CUDA Toolkit 10.1.
▶ If your CUDA Driver is 10.0, the compilers use CUDA Toolkit 10.0.
▶ If your CUDA Driver is 9.2, the compilers use CUDA Toolkit 9.2.
▶ If your CUDA Driver is 9.1, the compilers will issue an error that CUDA Toolkit 9.1 was not found; CUDA Toolkit 9.1 is not bundled with PGI 19.7
▶ If you do not have a CUDA driver installed on the compilation system, the compilers use the default CUDA Toolkit version 9.2.
▶ If your CUDA Driver is newer than CUDA 10.1, the compilers will still use the CUDA Toolkit 10.1. The compiler selects the newest CUDA Toolkit it finds that is not newer than the CUDA Driver.

You can change the compiler's default selection for CUDA Toolkit version using one of the following methods:

▶ Use a compiler option. Add the `cudaX.Y` sub-option to `-Mcuda` or `-ta=tesla` where `X.Y` denotes the CUDA version. For example, to compile a C file with the CUDA 9.2 Toolkit you would use:
```
pgcc -ta=tesla:cuda9.2
```
Using a compiler option changes the CUDA Toolkit version for one invocation of the compiler.

▸ Use an rcfile variable. Add a line defining `DEFCUDAVERSION` to the `siterc` file in the installation `bin/` directory or to a file named `.mypgirc` in your home directory. For example, to specify the CUDA 9.2 Toolkit as the default, add the following line to one of these files:

```
set DEFCUDAVERSION=9.2;
```

Using an rcfile variable changes the CUDA Toolkit version for all invocations of the compilers reading the rcfile.

When you specify a CUDA Toolkit version, you can additionally instruct the compiler to use a CUDA Toolkit installation different from the defaults bundled with the current PGI compilers. While most users do not need to use any other CUDA Toolkit installation than those provided with PGI, situations do arise where this capability is needed. Developers working with pre-release CUDA software may occasionally need to test with a CUDA Toolkit version not included in a PGI release. Conversely, some developers might find a need to compile with a CUDA Toolkit older than the oldest CUDA Toolkit installed with a PGI release. For these users, PGI compilers can interoperate with components from a CUDA Toolkit installed outside of the PGI installation directories.

PGI tests extensively using the co-installed versions of the CUDA Toolkits and fully supports their use. Use of CUDA Toolkit components not included with a PGI install is done with your understanding that functionality differences may exist.

The ability to compile with a CUDA Toolkit other than the versions installed with the PGI compilers is supported on all platforms; on the Windows platform, this feature is supported for CUDA Toolkit versions 9.2 and newer.

To use a CUDA toolkit that is not installed with a PGI release, such as CUDA 9.1 with PGI 19.7, there are three options:

▸ Use the rcfile variable `DEFAULT_CUDA_HOME` to override the base default

```
set DEFAULT_CUDA_HOME = /opt/cuda-9.1;
```

▸ Set the environment variable `CUDA_HOME`

```
export CUDA_HOME=/opt/cuda-9.1
```

▸ Use the compiler compilation line assignment CUDA_HOME=

```
pgfortran CUDA_HOME=/opt/cuda-9.1
```

The PGI compilers use the following order of precedence when determining which version of the CUDA Toolkit to use.

1. If you do not tell the compiler which CUDA Toolkit version to use, the compiler picks the CUDA Toolkit from the PGI installation directory `2019/cuda` that matches the version of the CUDA Driver installed on your system. If the PGI installation directory does not contain a direct match, the newest version in that directory which is not newer than the CUDA driver version is used. If there is no CUDA driver installed on your system, the compiler falls back on an internal default; in PGI 19.7, this default is CUDA 9.2.
2. The rcfile variable `DEFAULT_CUDA_HOME` will override the base default.
3. The environment variable `CUDA_HOME` will override all of the above defaults.
4. The environment variable `PGI_CUDA_HOME` overrides all of the above; it is available for advanced users in case they need to override an already-defined `CUDA_HOME`.

5. A user-specified `cudaX.Y` sub-option to `-Mcuda` and `-ta=tesla` will override all of the above defaults and the CUDA Toolkit located in the PGI installation directory `2019/cuda` will be used.

6. The compiler compilation line assignment `CUDA_HOME=` will override all of the above defaults (including the `cudaX.Y` sub-option).

2.9. Compute Capability

The compilers can generate code for NVIDIA GPU compute capabilities 3.0 through 7.5. The compilers construct a default list of compute capabilities that matches the compute capabilities supported by the GPUs found on the system used in compilation. If there are no GPUs detected, the compilers select cc35, cc50, cc60, and cc70.

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=60 70;
```

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.

The generation of device code can be time consuming, so you may notice an increase in compile time as the number of compute capabilities increases.

2.10. OpenMP

OpenMP 3.1

The PGI Fortran, C, and C++ compilers support OpenMP 3.1 on all platforms.

2.11. Runtime Library Routines

PGI 2019 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 2019 is not compatible with previous releases. If you use `Tools | Options` to target a release prior to 2019, 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 2019 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\19.7\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 2019 installed for Microsoft Visual Studio 2015 includes version 14.0 of the Microsoft Open Tools.

The Microsoft Open Tools directory contains a subdirectory named `REDIST`. PGI 2019 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.

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

ACC routine directive limitations

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

▸ Fortran assumed-shape arguments are not yet supported.

Clause Support Limitations

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

OpenACC Profiling limitations

Limitations of PGI's OpenACC library:

▸ The OpenACC Profiling interface is not available for applications linked with static libraries, with "-Bstatic" or "-Bstatic_pgi".

Chapter 6.
CONTACT INFORMATION

You can contact NVIDIA's PGI compilers and tools team at:

> 9030 NE Walker Road, Suite 100
> Hillsboro, OR 97006

Or electronically using any of the following means:

> Fax: +1-503-682-2637
> Sales: sales@pgroup.com
> WWW: https://www.pgroup.com or pgicompilers.com

The PGI User Forum, pgicompilers.com/userforum 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, pgicompilers.com/login 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), pgicompilers.com/faq.

Submit support requests using the PGI Technical Support Request form, pgicompilers.com/support-request.