PGI® COMPILERS & TOOLS

RELEASE NOTES FOR X86 CPUS AND TESLA GPUS

Version 2019

NVIDIA

TABLE OF CONTENTS

Chapter 1. What's New in PGI 2019 1
1.1. What's New in 19.9 1
1.2. What's New in 19.7 1
1.3. What's New in 19.5 3
1.4. What's New in 19.4 3
1.5. What's New in 19.3 3
1.6. What's New in 19.1 4
1.7. LLVM Code Generator 6
Chapter 2. Release Overview 9
2.1. Licensing 9
2.1.1. Licensing Terminology 9
2.1.2. Bundled License Key 10
2.1.3. Node-locked and Network Floating Licenses 10
2.2. Release Components 10
2.3. Terms and Definitions 11
2.4. Supported Platforms 11
2.5. Supported Operating System Updates 11
2.5.1. Linux 11
2.5.2. Apple macOS 12
2.5.3. Microsoft Windows 12
2.6. CUDA Toolkit Versions 12
2.7. Compute Capability 15
2.8. Precompiled Open-Source Packages 15
Chapter 3. Distribution and Deployment 17
3.1. Application Deployment and Redistributables 17
3.1.1. PGI Redistributables 17
3.1.2. Linux Redistributables 17
Chapter 4. Troubleshooting Tips and Known Limitations 18
4.1. Platform-specific Issues 18
4.1.1. Linux 18
4.1.2. Apple macOS 18
4.1.3. Microsoft Windows 18
4.2. Issues Related to Debugging 19
4.3. Profiler-related Issues 19
4.4. OpenACC Issues 20
Chapter 5. Contact Information 21

Chapter 1. WHAT'S NEW IN PGI 2019

Welcome to Release 2019 of the PGI compilers and tools!

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.

1.1. What's New in 19.9

OpenACC

The CUDA 10.1 version bundled with PGI has been upgraded from 10.1 Update 1 to 10.1 Update 2. More information can be found in the CUDA 10.1 release notes.

Removed support for putting worker loops inside vector loops, or gang loops inside worker or vector loops, unless the loops are tightly nested. This change aligns with the restrictions in the OpenACC specification.

1.2. 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.

Added support for capture of the ***this** pointer by value to the OpenACC C++ compiler.

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.

1.3. 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.

1.4. 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.

1.5. 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.

1.6. What's New in 19.1

All Compilers

The LLVM backend has been upgraded from version 6.0 to 7.0 and is now the default code generator for Linux x86-64 platforms. This is a significant change that is described in more detail in the LLVM Code Generator section.

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 the following versions of Linux/x86-64:

- CentOS 7.6
- Fedora 29
- RHEL 7.6
- Ubuntu 18.10

Added support for Windows Server 2019 and macOS Mojave.

C/C++

Updated support for SIMD intrinsic functions in the **pgc++** compiler for Linux/x86-64 platforms. Support is comprehensive for SSE, SSE2, SSE3, SSSE3, SSE4.1 and SSE4.2 intrinsics. Many of the AVX, AVX2 and AVX-512 intrinsics are now implemented as well. We expect to add comprehensive support for the AVX intrinsics in a future release.

The **pgc++** compiler now implements full support for the C+17 language standard.

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.

1.7. LLVM Code Generator

The PGI 2019 compilers for Linux/x86-64 platforms now use an LLVM-based code generator and OpenMP runtime library by default. In general, the new compilers deliver better performance than those in PGI 2018 and prior releases, which used a PGI-proprietary code generator and OpenMP runtime. However, the PGI 2019 compilers using LLVM are not link-compatible with PGI 2018 and prior releases, and they have certain limitations as outlined below. The PGI-proprietary code generator and OpenMP implementation can still be used optionally with the PGI 2019 compilers via a compile- and link-time option or with appropriate path settings.

After installing PGI 2019, the following subdirectories will exist in your base installation directory:

- `linux86-64/`
- `linux86-64-llvm/`
- `linux86-64-nollvm/`

`linux86-64-llvm/` contains the default LLVM-based compilers. The 19.9/ and 2019/ subdirectories in `linux86-64/` are symbolic links to their equivalents in `linux86-64-llvm/`. If either of these directories have precedence in your path settings, then invoking any of the PGI compilers will result in use of the LLVM backend code generator by default.

`linux86-64-nollvm/` contains the version of the compilers that use the PGI-proprietary code generator and OpenMP runtime. You can control which code generator is used, and in particular can override the default and revert to using the PGI-proprietary code generator if needed. However, in general you cannot mix object files or libraries compiled with the two different code generators in the same executable.

If you are using default PGI and environment path settings in which the location of the LLVM-based compilers takes precedence, then invoking any of the PGI compiler drivers with the -Mnollvm option will choose the version of the PGI 2019 compilers that use the PGI-proprietary code generator in linux86-64-nollvm/19.9/bin. You must use this option on all files and libraries in your program, and as both a compile- and link-time option.

The installation is designed so that if needed you can set up your environment and path settings to point directly to the linux86-64-nollvm/19.9/bin directory, in which case all invocations of the PGI compiler drivers will use the PGI-proprietary code generator. In that case, using the **-Mllvm** option will toggle to use of the LLVM-based compilers and again it must be used for both compilation and linking.

It is recommended that you initialize your environment and path settings to point to linux86-64/19.9/bin and use the **-M[no]llvm** switch to toggle the desired backend. The PGI-proprietary code generator will be deprecated in a future release of the PGI compilers.

Environment Modules

If your environment is set up to use PGI's environment module files, the following commands load the PGI compilers with the default code generator for a given release:

```
module load pgi/<release_number>
```

The default code generator in PGI 19.9 and all future releases is the LLVM-based code generator, so

```
module load pgi/19.9
```

will choose the LLVM-based code generator. The default code generator in PGI 2018 and all prior releases was the PGI-proprietary code generator, and LLVM was available optionally in PGI 2018. So,

```
module load pgi/18.10
```

will choose the PGI-proprietary code generator.

You can be explicit about the version and backend by executing a module load of `pgi-llvm` or `pgi-nollvm` first, followed by a module load of the desired version. For example:

```
module load pgi-llvm
module load pgi/18.10
```

will override the 18.10 default and initialize your environment to use the LLVM-based code generator and OpenMP runtime and PGI 18.10 release. Similarly:

```
module load pgi-nollvm
module load pgi/19.9
```

will override the 19.9 default and initialize your environment to use the PGI-proprietary code generator and OpenMP runtime and the PGI 19.9 release. You could then toggle to using the default LLVM-based code generator with these commands:

```
module purge
module load pgi/19.9
```

or an appropriate sequence of module unload and module load commands.

Limitations

There are a few known limitations with the LLVM-based code generator, including:

- Only available for Linux/x86-64 platforms

- The PGI Unified Binary feature that allows targeting multiple types of CPU in one binary executable is not supported; if you specify multiple targets to the **-tp** compiler option you will get an error message and compilation will fail
- Interprocedural analysis and optimizations are not performed; the **-Mipa** compiler option is processed by the compiler drivers without error, but is effectively ignored
- Some source-code directives, e.g. **DEC$**, may have no effect

Chapter 2. RELEASE OVERVIEW

This chapter provides an overview of Release 2019 of the PGI Accelerator™ compilers and development tools for 64-bit x86-compatible processor-based workstations, servers, and clusters running versions of the Linux, Apple macOS and Microsoft Windows operating systems.

2.1. Licensing

All PGI products for a given platform include exactly the same PGI compilers and tools software. The difference is in which features are enabled by the license keys.

PGI release 2019 version 19.1 and newer contains updated (v11.16.2) FlexNet Publisher license management software.

The updated version of FlexNet Publisher includes various fixes and improved support across all operating systems.

Important Users with PGI 2018 (18.x) or older need to update their license daemons to support 19.1 or newer. The new license daemons are backward-compatible with older PGI releases. For more information, see the FlexNet Update FAQ.

2.1.1. Licensing Terminology

The PGI compilers and tools are license-managed. Before discussing licensing, it is useful to have common terminology.

- **License** – the right to use PGI compilers and tools as defined by the End-user License Agreement (EULA), this is a legal agreement between NVIDIA and PGI end-users. PGI Professional (for-fee, perpetual) licenses are identified by a Product Identification Number (PIN - see below). You can find a copy of the EULA on the PGI website , pgicompilers.com/LICENSE, and in the `$PGI/<platform>/<rel_number>/doc` directory of every PGI software installation.
- **License keys** – ASCII text strings that enable use of the PGI software and are intended to enforce the terms of the License. For PGI Professional, License keys are generated by each PGI end-user on the PGI website using a unique hostid and are

typically stored in a file called `license.dat` that is accessible to the systems for which the PGI software is licensed.

- **PIN** – Product Identification Number, a unique 6-digit number associated with a PGI Professional license. This PIN is included in your order confirmation. The PIN can also be found in your license key file after **`VENDOR_STRING=`**.
- **PIN tie code** – A unique 16-digit number associated with each license (PIN) that allows others to "tie" that license to their PGI user account, pgicompilers.com/ account for administrative purposes. PGI Professional licensees can use their PIN tie code to share license administration capabilies with others in their orgaization.

2.1.2. Bundled License Key

Installation may place a temporary license key file named `license.dat` in the PGI installation directory if no such file already exists.

If you use a separate license server, for example `LM_LICENSE_FILE=`*port@server.domain.com*, that supports this version, it is recommended that you remove or rename the license key file in the installation directory.

2.1.3. Node-locked and Network Floating Licenses

- Node-locked single-user licenses allow one user at a time to compile solely on the system on which both the PGI compilers and tools, and PGI license server are installed.
- Network floating licenses allow one or more users to use the PGI compilers and tools concurrently on any compatible client systems networked to a license server, that is, the system on which the PGI network floating license key(s) are installed. There can be multiple installations of the PGI compilers and tools on client systems connected to the license server; and client systems can use the license concurrently up to the maximum number of seats licensed for the license server.

2.2. Release Components

Release 2019 includes the following components:

- PGFORTRAN™ native OpenMP and OpenACC Fortran 2003 compiler.
- PGCC® native OpenMP and OpenACC ISO C11 and K&R C compiler.
- PGC++® native OpenMP and OpenACC ISO C++17 compiler.
- PGI Profiler® OpenACC, CUDA, OpenMP, and multi-thread graphical profiler.
- PGI Debugger® MPI, OpenMP, and multi-thread graphical debugger.
- Open MPI version 3.1.3 for 64-bit Linux including support for NVIDIA GPUDirect. Note that 64-bit linux86-64 MPI messages are limited to < 2 GB size each. As NVIDIA GPUDirect depends on InfiniBand support, Open MPI is also configured to use InfiniBand hardware if it is available on the system. InfiniBand support requires OFED 3.18 or later.

- ScaLAPACK 2.0.2 linear algebra math library for distributed-memory systems for use with Open MPI, MPICH or MVAPICH, and the PGI compilers on 64-bit Linux and macOS for Intel 64 or AMD64 CPU-based installations.
- Microsoft HPC Pack 2012 MS-MPI Redistributable Pack (version 4.1) for 64-bit development environments (Windows only).
- BLAS and LAPACK library based on the customized OpenBLAS project source.
- A UNIX-like shell environment for 64-bit Windows platforms.
- FlexNet license utilities.
- Documentation in man page format and online, pgicompilers.com/docs, in both HTML and PDF formats.

2.3. 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.

2.4. Supported Platforms

There are three platforms supported by the PGI compilers and tools for x86-64 processor-based systems.

- **64-bit Linux** – supported on 64-bit Linux operating systems running on a 64-bit x86 compatible processor.
- **64-bit macOS** – supported on 64-bit Apple macOS operating systems running on a 64-bit Intel processor-based Macintosh computer.
- **64-bit Windows** – supported on 64-bit Microsoft Windows operating systems running on a 64-bit x86-compatible processor.

2.5. Supported Operating System Updates

This section describes updates and changes to PGI 2019 that are specific to Linux, macOS, and Windows.

2.5.1. Linux

- CentOS 6.4 through 7.6

- RHEL 6.4 through 7.6
- Fedora 14 through 29
- Ubuntu 14.04, 16.04, 17.10, 18.04, 18.10
- openSUSE Leap 42.2 through openSUSE Leap 15.0
- SLES 12 SP2 through SLES 15

2.5.2. Apple macOS

PGI 2019 for macOS supports most of the features of the version for Linux environments. Except where noted in these release notes or the user manuals, the PGI compilers and tools on macOS function identically to their Linux counterparts.

- The compilers, debugger, and profiler are supported on macOS versions 10.10.5 (Yosemite) through 10.14 (Mojave).

2.5.3. Microsoft Windows

PGI products for Windows support most of the features of the PGI products for Linux environments. PGI products require that Visual Studio 2017, including the Windows 10 Software Development Kit (SDK), be installed prior to installing the compilers.

PGI 18.7 is the last release for Windows that includes bundled Microsoft toolchain components. Subsequent releases require users to have the Microsoft toolchain components pre-installed on their systems.

PGI 2019 requires the Windows 10 SDK, even on Windows 7 and 8.1.

These Windows operating systems are supported in PGI 2019:

- Windows Server 2008 R2
- Windows 7
- Windows 8.1
- Windows 10
- Windows Server 2012
- Windows Server 2016
- Windows Server 2019

2.6. 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 19.9 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 19.9 compilers fall back to the default of CUDA 9.2.

If the only PGI compiler you have installed is PGI 19.9, 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 19.9
- 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 19.9, 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 19.9, 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).

2.7. 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.

2.8. Precompiled Open-Source Packages

Many open-source software packages have been ported for use with PGI compilers on Linux x86-64.

The following PGI-compiled open-source software packages are included in the PGI Linux x86-64 download package:

- OpenBLAS 0.3.3 – customized BLAS and LAPACK libraries based on the OpenBLAS project source.
- Open MPI 3.1.3 – open-source MPI implementation.
- ScaLAPACK 2.0.2 – a library of high-performance linear algebra routines for parallel distributed memory machines. ScaLAPACK uses Open MPI 3.1.3.

The following list of open-source software packages have been precompiled for execution on Linux x86-64 targets using the PGI compilers and are available to download from the PGI website at pgicompilers.com/downloads.

- ESMF 7.1.0r for Open MPI 3.1.3 – The Earth System Modeling Framework for building climate, numerical weather prediction, data assimilation, and other Earth science software applications.
- NetCDF 4.6.2 for C++11 – A set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-

oriented scientific data, written in C. Included in this package are the following components:

- NetCDF-C++ 4.3.0 – C++ interfaces to NetCDF libraries.
- NetCDF-Fortran 4.4.4 – Fortran interfaces to NetCDF libraries.
- Parallel NetCDF 1.11.0 – for Open MPI 3.1.3.
- HDF5 1.10.4 – data model, library, and file format for storing and managing data.
- SZIP 2.1.1 – extended-Rice lossless compression algorithm.
- ZLIB 1.2.11 – file compression library.

- NetCDF 4.6.2 for C++98 – includes all the components listed in NetCDF for C++11 above.

In addition, these software packages have also been ported to PGI on Linux x86-64 but due to licensing restrictions, they are not available in binary format directly from PGI. You can find instructions for building them in the Porting & Tuning Guides section of the PGI website at pgicompilers.com/tips.

- FFTW 2.1.5 – version 2 of the Fast Fourier Transform library, includes MPI bindings built with Open MPI 3.1.3.
- FFTW 3.3.8 – version 3 of the Fast Fourier Transform library, includes MPI bindings built with Open MPI 3.1.3.

For additional information about building these and other packages, please see the Porting & Tuning Guides section of the PGI website at pgicompilers.com/tips.

Chapter 3. 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.

3.1. 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.

3.1.1. PGI Redistributables

The PGI 2019 Release includes these directories:

```
$PGI/linux86-64/19.9/REDIST
$PGI/win64/19.9/REDIST
```

These directories contain all of the PGI Linux runtime library shared object files or Windows dynamically linked libraries that can be re-distributed by PGI 2019 licensees under the terms of the PGI End-User License Agreement (EULA). For reference, a text-form copy of the PGI EULA is included in the 19.9 `doc` directory.

3.1.2. Linux Redistributables

The Linux `REDIST` directories contain the PGI runtime library shared objects for all supported targets. This enables users of the PGI compilers to create packages of executables and PGI runtime libraries that will execute successfully on almost any PGI-supported target system, subject to these requirements:

- End-users of the executable have properly initialized their environment.
- Users have set **`LD_LIBRARY_PATH`** to use the relevant version of the PGI shared objects.

Chapter 4. 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.

4.1. Platform-specific Issues

4.1.1. Linux

The following are known issues on Linux:

- Programs that incorporate object files compiled using -mcmodel=medium cannot be statically linked. This is a limitation of the linux86-64 environment, not a limitation of the PGI compilers and tools.
- 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.

4.1.2. Apple macOS

The following are known issues on Apple macOS:

- The PGI 2019 compilers do not support static linking of binaries. For compatibility with future Apple updates, the compilers only support dynamic linking of binaries.

4.1.3. Microsoft Windows

The following are known issues on Windows:

- For the Cygwin `emacs` editor to function properly, you must set the environment variable **`CYGWIN`** to the value "tty" before invoking the shell in which `emacs` will run. However, this setting is incompatible with the PGBDG command line interface (-text), so you are not able to use **`pgdbg -text`** in shells using this setting.
- On Windows, the version of `vi` included in Cygwin can have problems when the **`SHELL`** variable is defined to something it does not expect. In this case, the following messages appear when `vi` is invoked:

  E79: Cannot expand wildcards Hit ENTER or type command to continue

  To work around this problem, set **`SHELL`** to refer to a shell in the Cygwin `bin` directory, e.g., `/bin/bash`.
- On Windows, runtime libraries built for debugging (e.g., `msvcrtd` and `libcmtd`) are not included with PGI products. 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.

4.2. Issues Related to Debugging

The following are known issues in the PGI debugger:

- Debugging of PGI Unified Binaries, that is, programs built with more than one -tp option, is not fully supported. The names of some subprograms are modified in compilation and the debugger does not translate these names back to the names used in the application source code.
- When debugging on the Windows platform, the Windows operating system times out **`stepi`**/**`nexti`** operations when single stepping over blocked system calls.

4.3. Profiler-related Issues

Some specific issues related to the PGI Profiler:

- The Profiler relies on being able to directly call 'dlsym'. If this system call is intercepted by the program being profiled or by some other library the profiler may hang at startup. We have encountered this specific problem with some implementations of MPI. We recommend you disable any features that may be intercepting the 'dlsym' system call or disable CPU profiling with the --cpu-profiling off option.
  - To disable 'dlsym' interception when using IBM's spectrum MPI set the environment variable: PAMI_DISABLE_CUDA_HOOK=1, omit the following option: -gpu and add the options: -x PAMI_DISABLE_CUDA_HOOK and -disable_gpu_hooks.

4.4. 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 5. 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.