PGI® COMPILERS & TOOLS

RELEASE NOTES FOR X86 CPUS AND TESLA GPUS

Version 2020

NVIDIA

TABLE OF CONTENTS

Chapter 1. WHAT'S NEW IN PGI 2020

Welcome to Release 2020 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 20.4

All Compilers

Added support for interoperability with GNU-compiled OpenMP objects. Our optimized heterogeneous OpenMP runtime now includes a GNU OpenMP interface layer which provides compatibility when mixing PGI-compiled and GNU-compiled OpenMP objects. For more information, refer to OpenMP.

Changed the compile and link time option **`-nomp`** such that its use now prevents the addition of the OpenMP runtime library (**`nvomp`**) to the link line.

C/C++

Added **`DWARF`** metadata for inlined function calls.

Fortran

Improvement performance in **`TRANSFER`** intrinsic for large contiguous data transfers.

Added **`DWARF`** metadata for inlined function and subroutine calls.

OpenACC and CUDA Fortran

Changed the default CUDA Toolkit version selected by the compiler from CUDA 10.0 to 10.1. The default CUDA Toolkit version is used when the compiler cannot find a GPU or CUDA driver on the system where it is compiling and no CUDA Toolkit version has been specified on the compilation line. For more information about selecting CUDA Toolkit versions see CUDA Toolkit Versions.

Updated the implementation of the OpenACC API rouintes **`acc_copyin`** and **`acc_create`** to adhere to the OpenACC 2.0 specification to use **`present_or_`** semantics.

1.2. What's New in 20.1

All Compilers

LLVM 9.0 integrated — Upgraded Linux/x86-64 compilers to use LLVM 9.0 as the default code generator. LLVM 8.0 and the legacy PGI code generator are still available via compile and link-time command-line options.

cuTENSOR support — Added support for the new cuTENSOR library, including automatic mapping of Fortran transformational intrinsics operating on device data to cuTENSOR calls. See below for details.

New Heterogeneous OpenMP Runtime — Replaced the OpenMP runtime with a new optimized heterogeneous OpenMP runtime. In 20.1 this new runtime is used by default for multicore CPU targeting, and in future releases it will be used for integrated CPU and GPU OpenMP targeting. The new runtime is fully KMPC-compatible, but is not yet GOMP compatible, so mixing of PGI-compiled and GNU-compiled OpenMP objects is not supported and will result in a runtime error message.

AMD Rome CPUs support — The 20.1 release of the PGI Fortran, C and C++ compilers is fully supported on AMD Rome CPUs. The compilers auto-detect Rome as the default target CPU when installed and used on Rome systems.

Added support for the following versions of Linux/x86-64:

- CentOS 7.7, 8.0, and 8.1
- Fedora 30
- openSUSE Leap 15.1
- RHEL 7.7, 8.0, and 8.1
- SLES 15 SP1
- Ubuntu 19.04

C/C++

Upgraded the PGI C compiler **`pgcc`** significantly, including substantial support for C11.

Fortran

Improved debugging metadata for PURE, RECURSIVE, and ELEMENTAL procedures.

Improved implementation of the OpenMP API to follow the OpenMP specification. Fortran developers should import the OpenMP API by using the omp_lib module rather than by defining its types (i.e, omp_lock_t) or declaring its functions directly in the application code. Failure to do so (or failure to do so correctly) may cause an application built with the NVIDIA OpenMP runtime to function incorrectly.

OpenACC and CUDA Fortran

Added Fortran interfaces to the NVIDIA cuTENSOR library which is bundled in the PGI packages. Added support for a new cutensorEx Fortran module that maps Fortran intrinsic functions **`RESHAPE()`**, **`TRANSPOSE()`**, **`SPREAD()`** and **`MATMUL()`** operating on device data to the appropriate cuTENSOR functionality. This new feature is usable in both CUDA Fortran and OpenACC, and enables use of V100 tensor cores for operations on **`real(2)`** data. See the Fortran CUDA Library Interfaces document for more information.

Added support for the CUDA Toolkit version 10.2. See CUDA Toolkit Versions for instructions on selecting alternate CUDA Toolkit versions using **`CUDA_HOME`**. CUDA 10.0 is now the default toolkit version if there is no GPU or CUDA driver found on the system. If you are relying on this default, you must specify a version of CUDA Toolkit version 10.0 using **`CUDA_HOME`** because the CUDA 10.0 toolchain and libraries are not bundled in this release.

Added support for calling the built-in CUDA math functions sinpi, sinpif, cospi, cospif, sincospi, and sincospif from device code.

Improved support for **`attributes(host,device)`** functions in CUDA Fortran, which enables compiling a program unit for both host and device execution.

Implemented support for **`on_device`** in CUDA Fortran, which enables dynamic checking for current execution on a GPU device.

Changed copy behavior for OpenACC reductions to adhere to the OpenACC specification. The compilers will now follow the defined behavior for **`copy`** and not copy the reduction variable to the device if it is already present. To enable the compiler's previous behavior, use an **`update device/host if_present`** directive.

Reorganized the accelerator runtime libraries. Removed the **`libaccapi`**, **`libaccg`**, **`libaccg2`**, **`libaccn`**, and **`libaccnc`** libraries. Replaced these libraries with three new libraries organized by functionality. The compiler drivers link in the new libraries based on flags used when linking:

- The compiler adds **`libaccdevice`** with **`-acc -ta=tesla`**.
- The compiler adds **`libacccuda`** with **`-acc -ta=tesla -Mcuda`**.

- The compiler adds **libacchost** with **-ta=multicore** or **-ta=host** as long as **-ta=tesla** is not present.

Dropped support for **__CUDA_API_VERSION**. Use **CUDA_VERSION** instead.

Updated the profiling interface to conform to the OpenACC 3.0 specification.

Deprecations and Eliminations

CUDA 9.2 and 10.0 are no longer included as part of the PGI compilers installation package. CUDA 10.0 is still supported via **CUDA_HOME**. Support for CUDA 9.2 has been removed.

Official support for the **nonvvm** sub-option **-ta=tesla** and **-Mcuda** ended with the PGI 19.10 release. Although the compilers will not reject this sub-option, we recommend moving to the default method for generating device code by removing **nonvvm** from compilation.

The **pgf77** driver is no longer provided. Use **pgfortran** to compile F77 Fortran.

The PGI Debugger **pgdbg** has been discontinued. The last release of **pgdbg** was PGI 19.10. 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 Linux, the GNU debugger **gdb** can be used for basic debugging of PGI-generated code.

PGI Visual Fortran (PVF) has been discontinued. The final PVF release was 19.10. Existing PVF licensees can continue to use it indefinitely, but no new licenses will be issued. PGI command-level compilers for Windows will continue to be enhanced and supported.

Dropped support for the macOS platform. The last release of PGI products with support for Apple's macOS was PGI 19.10.

Chapter 2. RELEASE OVERVIEW

This chapter provides an overview of Release 2020 of the PGI Accelerator™ compilers and development tools for 64-bit x86-compatible processor-based workstations, servers, and clusters running versions of the Linux 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 2020 includes the following components:

- PGFORTRAN™ native CUDA Fortran, 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.
- 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 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 two 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 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 2020 that are specific to Linux and Windows.

2.5.1. Linux

- CentOS 6.4 through 8.1
- Fedora 28 through 30
- openSUSE Leap 42.3 through 15.1

- RHEL 6.4 through 8.1
- SLES 12 SP3 through SLES 15 SP1
- Ubuntu 14.04, 16.04, 18.04, 18.10, 19.04

2.5.2. 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 2020 requires the Windows 10 SDK, even on Windows 7 and 8.1.

These Windows operating systems are supported in PGI 2020:

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

2.6. OpenMP

OpenMP 3.1

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

The NVIDIA OpenMP runtime implements nested parallelism such that the inner parallel region or regions are run with one thread.

OpenMP 4.5

The PGI Fortran, C, and C++ compilers compile most OpenMP 4.5 programs for parallel execution across all the cores of a multicore CPU or server. **`target`** regions are implemented with default support for the multicore host as the target, and **`parallel`** and **`distribute`** loops are parallelized across all OpenMP threads. This feature is supported on Linux/x86 platforms with the LLVM code generator only.

Current limitations include:

- The **simd** construct can be used to provide tuning hints; the **simd** construct's **private**, **lastprivate**, **reduction**, and **collapse** clauses are processed and supported.
- The **declare simd** construct is ignored.
- The **ordered** construct's **simd** clause is ignored.
- The **task** construct's **depend** and **priority** clauses are not supported.
- The loop construct's **linear**, **schedule**, and **ordered(n)** clauses are not supported.
- The **declare reduction** directive is not supported.
- The **reduction** clause does not accept pointer or reference types.

Compatibility between NVIDIA and GNU OpenMP Runtimes

NVIDIA's OpenMP runtime implements a subset of OpenMP functionality; this OpenMP runtime functionality is provided in a library called **libnvomp**. The GNU OpenMP runtime also implements a subset of OpenMP functionality OpenMP applications and libraries compiled with the GNU compilers link in a library called **libgomp**.

If GNU and PGI compilers are used to compile different modules of an application that each uses the OpenMP API, both the GNU and the NVIDIA OpenMP runtimes will be required for the application to operate. But because these two runtimes implement intersecting subsets of the OpenMP specification and do not share their state, loading both OpenMP runtimes is not advised. NVIDIA's OpenMP runtime includes an interface layer containing the GNU OpenMP API (GOMP). When a PGI-compiled OpenMP application linked against a GNU-compiled OpenMP library is loaded, the GOMP runtime calls in the library are resolved by the GOMP interfaces contained in the NVIDIA OpenMP runtime library; the GOMP library is not itself loaded.

Most of the features in the GNU OpenMP runtime as of GCC 9.2.0 are supported in our GOMP interface layer. OpenMP 5.0 features are not supported yet and a few other features such as doacross loops are not supported in our interface layer. GNU-compiled apps making use of these features will likely terminate at some point if run with libnvomp loaded instead of libgomp. Certain other features such as nested parallelism may be implemented differently in the two libraries.

2.7. 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 `2020/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 20.4 includes the following versions of the CUDA Toolkit:

- CUDA 10.1
- CUDA 10.2

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 `2020/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 20.4 compilers fall back to the default of CUDA 10.1.

If the only PGI compiler you have installed is PGI 20.4, then:

- If your CUDA Driver is 10.2, the compilers use CUDA Toolkit 10.2.
- If your CUDA Driver is 10.1, the compilers use CUDA Toolkit 10.1.
- If your CUDA Driver is 10.0, the compilers will issue an error that CUDA Toolkit 10.0 was not found; CUDA Toolkit 10.0 is not bundled with PGI 20.4.
- If you do not have a CUDA driver installed on the compilation system, the compilers use the default CUDA Toolkit version 10.1.
- If your CUDA Driver is newer than CUDA 10.2, the compilers will still use the CUDA Toolkit 10.2. 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 10.2 Toolkit you would use:

  ```
  pgcc -ta=tesla:cuda10.2
  ```

  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 10.2 Toolkit as the default, add the following line to one of these files:

  ```
  set DEFCUDAVERSION=10.2;
  ```

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 10.0 with PGI 20.4, there are three options:

- Use the rcfile variable `DEFAULT_CUDA_HOME` to override the base default
  ```
  set DEFAULT_CUDA_HOME = /opt/cuda-10.0;
  ```
- Set the environment variable `CUDA_HOME`
  ```
  export CUDA_HOME=/opt/cuda-10.0
  ```
- Use the compiler compilation line assignment CUDA_HOME=
  ```
  pgfortran CUDA_HOME=/opt/cuda-10.0
  ```

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 `2020/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 20.4, this default is CUDA 10.1.
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 `2020/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).

2.8. 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.

2.9. 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 2020 Release includes these directories:

```
$PGI/linux86-64/20.4/REDIST
$PGI/win64/20.4/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 2020 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 20.4 `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 or PGI 2020 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. 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.
- 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. 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.

4.3. 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.