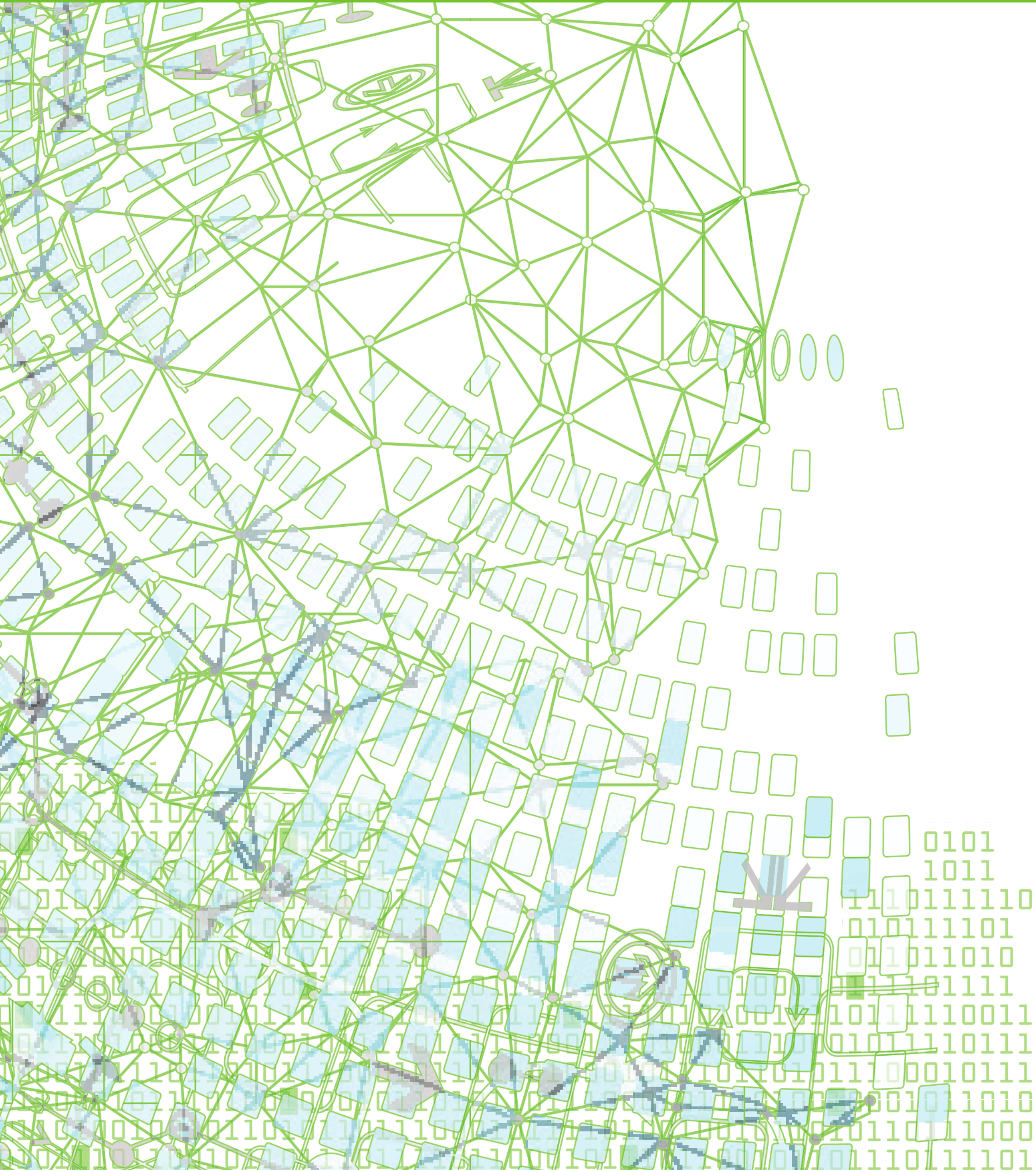
PGI® COMPILERS & TOOLS

INSTALLATION AND RELEASE NOTES FOR OPENPOWER CPUS

Version 2017



NVIDIA.

TABLE OF CONTENTS

Chapter 1. Release Overview....................1

1.1. About This Release.................... 1

1.2. Release Components.................... 1

1.3. Supported Platforms.................... 2

1.4. Supported Operating Systems.................... 2

1.5. Supported CUDA Software....................2

1.5.1. Targeting a CUDA Toolkit Version....................2

1.6. Precompiled Open-Source Packages.................... 3

1.7. Getting Started....................4

Chapter 2. New and Modified Features.................... 5

2.1. What's New in Release 2017....................5

2.2. Command-line Environment.................... 11

2.3. Fortran Language....................12

2.4. C Language.................... 12

2.5. C++ Language.................... 12

2.6. OpenACC.................... 13

2.7. OpenMP....................16

Chapter 3. Installation and Configuration.................... 18

3.1. License Management....................18

3.2. Environment Initialization.................... 19

3.3. Network Installations.................... 19

Chapter 4. Troubleshooting Tips and Known Limitations.................... 21

4.1. Release Specific Limitations....................21

4.2. Profiler-related Issues.................... 21

Chapter 5. Contact Information.................... 23

LIST OF TABLES

Table 1 Typical -fast Options 4

Table 2 Pool Allocator Environment Variables 14

Chapter 1. RELEASE OVERVIEW

Welcome to Release 2017 of the PGI Accelerator™ C11, C++14 and Fortran 2003 compilers hosted on and targeting OpenPOWER+Tesla processor-based servers and clusters running versions of the Linux operating system.

1.1. About This Release

These PGI compilers generate host CPU code for 64-bit little-endian OpenPOWER CPUs, and GPU device code for NVIDIA Kepler and Pascal GPUs.

These compilers include all GPU OpenACC features available in the PGI C/C++/Fortran compilers for x86-64.

Documentation includes the `pgcc`, `pgc++` and `pgfortran` man pages and the -help option. In addition, you can find both HTML and PDF versions of these installation and release notes, the *PGI Compiler User's Guide* and *PGI Compiler Reference Manual* for OpenPOWER on the PGI website, https://www.pgicompilers.com/resources/docs.php.

1.2. Release Components

Release 2017 includes the following components:

- PGFORTRAN™ native OpenMP and OpenACC Fortran 2003 compiler.
- PGCC® native OpenMP and OpenACC ANSI C11 and K&R C compiler.
- PGC++® native OpenMP and OpenACC ANSI C++14 compiler.
- PGI Profiler® OpenACC, CUDA, OpenMP, and multi-thread profiler.
- Open MPI version 2.1.2 including support for NVIDIA GPUDirect. GPUDirect requires CUDA 7.5 or later. 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 and the PGI compilers.
- BLAS and LAPACK library based on the customized OpenBLAS project source.
- Documentation in man page format and online PDFs.

1.3. Supported Platforms

These OpenPOWER hardware/software platforms have been used in testing:

- CPUs: POWER8, POWER8E, POWER8NVL, POWER9
- Linux distributions:
  - Ubuntu 14.04, 16.04
  - RHEL 7.3, 7.4 (POWER9)
- GCC versions: 4.4.6 (POWER9), 4.8.4, 4.8.5, 5.3.1
- CUDA Toolkit versions:
  - 7.5 driver versions 352.39, 352.79
  - 8.0 driver version 361.93.02
  - 9.0 driver version 384.59, 384.65 (POWER9)

1.4. Supported Operating Systems

The PGI 17.10 compilers were built on an OpenPOWER system running Ubuntu 14.04 OS with a GCC 4.8.2 toolchain. They have been tested on that platform, Ubuntu 16.04 with GCC 5.3.1, and RHEL 7.3 with GCC 4.8.5.

1.5. Supported CUDA Software

Select components of the CUDA Toolkit 7.5 and 8 are included under the PGI installation tree in `/opt/pgi`.

1.5.1. Targeting a CUDA Toolkit Version

To use a version of the CUDA Toolkit, you must first download and install the appropriate CUDA Driver from NVIDIA at http://www.nvidia.com/cuda. 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 the first line of output. Use it if you are unsure which version of the CUDA toolkit is installed on your system.
- The CUDA 7.5 Toolkit is set as the default in PGI 17.10.
- You can compile with the CUDA 8 Toolkit either by adding the option -ta=tesla:cuda8.0 to the command line or by adding **`set DEFCUDAVERSION=8.0`** to the `siterc` file. Compiling with the CUDA 8 Toolkit means you must use a CUDA 8 or later driver.
- You can compile with the CUDA 9 Toolkit either by adding the option -ta=tesla:cuda9.0 to the command line or by adding **`set DEFCUDAVERSION=9.0`** to the `siterc` file. Compiling with the CUDA 9 Toolkit means you must use a CUDA 9 or later driver.

1.6. Precompiled Open-Source Packages

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

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

- OpenBLAS 0.2.19 – customized BLAS and LAPACK libraries based on the OpenBLAS project source.
- Open MPI 2.1.2 – 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 2.1.2.

The following list of open-source software packages have been precompiled for execution on OpenPOWER targets using the PGI compilers and are available to download from the PGI website at https://www.pgicompilers.com/support/downloads.php.

- MPICH 3.2 – open-source MPI implementation.
- NetCDF 4.4.1.1 – 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.
  - HDF5 1.10.0-patch1 – data model, library, and file format for storing and managing data.
  - CURL 7.46.0 – tool and a library (usable from many languages) for client-side URL transfers.
  - SZIP 2.1 – extended-Rice lossless compression algorithm.
  - ZLIB 1.2.8 – file compression library.
- Parallel NetCDF 1.7.0 for MPICH
- Parallel NetCDF 1.7.0 for Open MPI

In addition, these software packages have also been ported to PGI on OpenPOWER 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 https://www.pgicompilers.com/resources/tips.htm.

- FFTW 2.1.5 – version 2 of the Fast Fourier Transform library, includes MPI bindings built with Open MPI 2.1.2.
- FFTW 3.3.4 – version 3 of the Fast Fourier Transform library, includes MPI bindings built with Open MPI 2.1.2.

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

1.7. Getting Started

By default, the PGI 2017 compilers generate code that is optimized for the type of processor on which compilation is performed, the compilation host. If you are unfamiliar with the PGI compilers and tools, a good option to use by default is the aggregate option -fast.

Aggregate options incorporate a generally optimal set of flags that enable use of SIMD instructions .

The content of the -fast option is host-dependent.

The following table shows the typical -fast options.

Table 1 Typical -fast Options

Use this option...	To do this...
-O2	Specifies a code optimization level of 2 and -Mvect=SIMD.
-Munroll=c:1	Unrolls loops, executing multiple instances of the original loop during each iteration.
-Mlre	Indicates loop-carried redundancy elimination.
-Mautoinline	Enables automatic function inlining in C & C++.

For best performance on processors that support SIMD instructions, use the PGFORTRAN compiler, even for FORTRAN 77 code, and the -fast option.

You may also be able to obtain further performance improvements by experimenting with the individual -Mpgflag options that are described in the *PGI Compiler Reference Manual*, such as -Mvect, -Munroll, -Minline, -Mconcur, and so on. However, increased speeds using these options are typically application and system dependent. It is important to time your application carefully when using these options to ensure no performance degradations occur.

Chapter 2. NEW AND MODIFIED FEATURES

This chapter provides information about the new or modified features of Release 2017 of the PGI compilers and tools.

Many user-requested fixes and updates are implemented in each PGI release. Refer to the Technical Problem Reports, https://www.pgicompilers.com/support/release_tprs.htm webpage for a complete and up-to-date table of TPRs fixed in recent PGI releases. This table contains a summary description of each problem as well as the version in which it was fixed.

2.1. What's New in Release 2017

17.10 Updates and Additions

- Integrated full support for the official release of the CUDA Toolkit 9.0; resolved PGI 17.9 limitation involving changes to warp shuffle instructions in CUDA 9.0. CUDA 7.5 remains the default version used by the compilers. To use either CUDA 8.0 or CUDA 9.0 instead, add either the `cuda8.0` or `cuda9.0` sub-option to the `-ta=tesla` or `-Mcuda` compile- and link-time options.
- Implemented initial support for CUDA 9.0 cooperative groups in CUDA Fortran. Refer to the CUDA Fortran Programming Guide for details.
- Implemented support for the OpenMP 4.5 **`taskloop`** construct and all its clauses except **`firstprivate`** and **`lastprivate`**. OpenMP 4.5 programs currently support parallel execution across all the cores of a multicore CPU or server.
- Added initial support for the CUDA 9.1 toolkit. Requirements for using CUDA 9.1 with PGI 17.10:
  - Install a version of the CUDA 9.1 toolkit.
  - Specify the location of the CUDA 9.1 toolkit using the compiler option CUDAROOT; for example, if CUDA 9.1 is installed in `/opt/cuda-9.1`, add **`CUDAROOT=/opt/cuda-9.1`** when compiling, linking, or invoking the profiler.

- Implemented `-Minstrument` for C, C++ and Fortran. This option instructs the compiler to generate instrumentation calls for entry and exit to functions. Just after function entry and just before function exit, the following profiling functions are called with the address of the current function and its call site:

  ```
  void __cyg_profile_func_enter (void *this_fn, void *call_site);
  void __cyg_profile_func_exit (void *this_fn, void *call_site);
  ```

  In these calls, the first argument is the address of the start of the current function. This option is equivalent to the GCC option `-finstrument-functions`.
- Implemented `-Mstack-init=snan`. The compiler will initialize stack memory to 64-bit signaling nans.
- Implemented support for GCC 7.2 for OpenPOWER.
- Updated the version of the PGI-built Open MPI libraries to 2.1.2.

17.9 Updates and Additions

- All PGI Compilers

  An initial validation of PGI 17.9 for Linux/POWER9 systems has been performed on a pre-production POWER9 system running RHEL 7.4 and GCC 4.4.6. By default, the PGI 17.9 compilers generate code that will run on either a POWER8 or a POWER9 CPU. In a future release, we expect to add support for POWER9-specific code generation.
- PGI Accelerator Compilers

  Integrated support for the CUDA Toolkit 9.0 Release Candidate (RC). CUDA 7.5 remains the default version used by the compilers. To use either CUDA 8.0 or CUDA 9.0 instead, add either the `cuda8.0` or `cuda9.0` sub-option to the `-ta=tesla` or `-Mcuda` compile- and link-time options.

  There are known problems using PGI 17.9 compilers with CUDA 9.0 installations. In particular, CUDA 9.0 introduces new synchronizing warp-shuffle instructions. The compilers in this PGI release still use older and now unreliable warp shuffle instructions to share values across threads in a warp for reduction operations and to copy values from vector lane zero to all vector lanes, in routines with the **`acc routine vector`** directive. This problem can occur even on Kepler and Maxwell GPUs. This issue will be corrected in an upcoming PGI release.

17.7 Updates and Additions

- PGI Accelerator Compilers
  - Added support for Tesla V100 GPUs in PGI OpenACC and CUDA Fortran. Based on the new NVIDIA Volta GV100 GPU, Tesla V100 offers more memory bandwidth, more streaming multiprocessors, next generation NVLink and new microarchitectural features that add up to better performance and

programmability. For OpenACC and CUDA Fortran programmers, Tesla V100 offers improved hardware support and performance for CUDA Unified Memory features. Use the sub-option `cc70` with the `-ta=tesla` or `-Mcuda` compile- and link-time options to generate code for the Tesla V100 GPUs; the CUDA 9 toolkit must be used when targeting Volta GPUs.

- Added initial support for the CUDA 9 toolkit. Requirements for using CUDA 9 with the PGI 17.7 compilers and profiler:
  - Install the release candidate (RC) or early access (EA) version of the CUDA 9 toolkit.
  - Specify the location of the CUDA 9 toolkit using the compiler option CUDAROOT; for example, if CUDA 9 is installed in `/opt/cuda-9.0`, add **`CUDAROOT=/opt/cuda-9.0`** when compiling, linking, or invoking the profiler.
  - Use the sub-option `cuda9.0` with the `-ta=tesla` or `-Mcuda` compile- and link-time options.
- Added support for CUDA Unified Memory. PGI 17.7 compilers leverage Pascal and Volta GPU hardware features, NVLink, and CUDA Unified Memory to simplify OpenACC and CUDA Fortran programming on GPU-accelerated OpenPOWER processor-based servers. When CUDA Fortran or OpenACC allocatable data is placed in CUDA Unified Memory, no explicit data movement or data directives are needed. This simplifies GPU acceleration of applications that make extensive use of allocatable data, and allows you to focus on parallelization and scalability of your algorithms. Use the `-ta=tesla:managed` compiler option to enable this feature. In previous PGI releases, compiler support for CUDA Unified Memory was provided as an optionally-installed, evaluation feature.
- Added a pool allocator to minimize the number of calls to cudaMallocManaged() for CUDA Unified Memory support. The pool allocator is enabled by default when `-ta=tesla:managed` or `-ta=tesla:pinned` is used.
- Added Beta support for automatic deep copy of Fortran derived types. This feature allows you to port applications with modern deeply nested data structures to Tesla GPUs using OpenACC. The PGI 17.7 compilers allow you to list aggregate Fortran data objects in OpenACC **`copy`**, **`copyin`**, **`copyout`** and **`update`** directives to move them between host and device memory including traversal and management of pointer-based objects within the aggregate data object. When enabled, full deep copy ensures that, when moving Fortran variables of derived type from host to device or device to host, the entire data structure, including pointers and allocatable arrays, is copied back and forth between host and device, or device and host, memory. To enable deep copy, use the `deepcopy` sub-option to `-ta=tesla`. Two things to note: polymorphic data

  types are not supported, and the presence of overlapping pointers may cause runtime errors.
  - The cuSOLVER library is based on the cuBLAS and cuSPARSE libraries, and supports matrix factorization and triangular solve routines for dense matrices, a sparse least-squares solver and an eigenvalue solver, and provides a refactorization library useful for solving sequences of matrices with a shared sparsity pattern. You can now call optimized cuSolverDN routines from CUDA Fortran and OpenACC Fortran using the PGI-supplied interface module and the PGI-compiled version of the cuSOLVER libraries bundled with PGI 17.7 for Linux/x86-64 and OpenPOWER. This same cuSolver library is callable from PGI OpenACC C/C++ as they are built using PGI compilers and are compatible with and use the PGI OpenMP runtime.
- All PGI Compilers
  - Enhanced support for OpenMP 4.5 syntax and features. The PGI Fortran, C, and C++ compilers will now 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.
  - Improved inlining with the `-Minline` option. You may see different functions inlined with this release than you observed in prior releases. In some cases, compilation may take noticeably longer to complete because of an increase in the number of functions inlined. Added the `smallsize:number` sub-option to `-Minline` which you can use to always inline functions of size smaller than number regardless of other size limits.
  - Added a new feature for program exploration called Program Analysis Summary Output (PASO). PASO enables the exporting of information gathered from the internal representation (IR) of various compilation stages in an organized, human readable format. This data forms a detailed summary of the program's internal structure, spanning not just one but all the files that make up a project. To enable PASO, compile and link with the `-Msummary` option. To find out more about how PASO works and how you might leverage it, refer to Program Analysis Using Summary Output, https://www.pgicompilers.com/blogs/posts/paso.htm. PASO is available on all platforms supported by PGI except macOS.
- C++ Compiler
  - Improved the raw performance on the LCALS loops benchmarks by an average of 20%, as well as eliminated the lambda abstraction penalty associated with these loops.
  - Added support for lambdas with capture in OpenACC GPU-accelerated compute regions.
  - Added incremental support for C++17 features consistent with EDG 4.13. This support includes:

        - new rules for auto deduction from braced-init-list
        - allow typename in template template parameter list
        - attributes for namespaces and enumerators
        - u8 character literals
        - nested namespace definitions
        - extended static_assert
        - generalized range-based for
        - remove deprecated register keyword
        - remove deprecated operator++(bool)
        - __has_include
        - [[fallthrough]]
        - [[nodiscard]]
        - [[maybe_unused]]
        - hexadecimal floating literals
        - using attribute namespaces without repetition
        - standard and non-standard attributes
    - Added support for interoperability with GNU 6.3.
    - Added support for the PGI C++ compilers as a CUDA 9.0 NVCC host compiler.
- Fortran Compiler
    - Added generation of debug information (DWARF) for Fortran. Commonly used by tools like debuggers, debug information allows one to access details about a routine's call stack, local and global variables, and set breakpoints.
- Profiler

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

    - Enhanced unified memory profiling:
        - Added association between unified memory events/CPU page faults and the source location of memory allocation.
        - Added new unified memory profiling events for page thrashing, throttling and remote map.
        - Added support for switching between segment and non-segment timelines.
        - Added filtering of events based on the virtual address, migration reason or page fault access type.
    - Added an OpenACC details table summarizing each recorded event.
    - Added support for the profiling of cooperative kernel launches.
    - Added NVLink events to the timeline.

    - Added memory throughput to the NVLink topology diagram.
    - Increased the options available for multi-hop remote profiling.
    - Added support for OpenACC profiling on all multicore systems.
- Documentation

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

17.5 Updates and Additions

- PGI Compilers
    - Bug fixes only.

17.4 Updates and Additions

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

17.3 Updates and Additions

Added the PGI C/C++ compiler plug-in for Eclipse versions Mars and Luna.

17.1 Updates and Additions

- Improved inlining with the `-Minline` option. You may see different functions inlined with this release than you observed in prior releases. In some cases, compilation may take longer to complete because of an increase in the number of functions inlined. Some of the `-Minline` sub-options, which you can use to adjust which functions get inlined, have also changed from previous releases:

  - Added `totalsize:n` to limit inlining to the total size of n where n is the size of the combined program units on a per file basis.
  - Changed `size:n` to `maxsize:n` which allows inlining only of functions smaller than approximately n lines. The compilers silently convert the previous `size:n` to `maxsize:n`.
  - Dropped `levels:n` which limited inlining to n levels of functions. The compilers silently ignore `levels:n`.
- Added the `-cpp` option as an alias for the `-Mpreprocess` option.
- Improved exception handling support and dropped the following related compiler options: `--[no_]exceptions`, `--need_exception_spec`, `--sjlj_eh`, `--[no]zc_eh`.
- Added the `-M[no]variadic_macros` to allow or disallow variadic macros. Variadic macros are allowed by default.
- Changed the default version of LLVM to 3.9.
- Improved support for `-ta=multicore`.
- Dropped support for CUDA 7.0.
- Changed the default version of the CUDA Toolkit used by the compilers from CUDA 7.0 to 7.5. The CUDA Toolkit 8.0 can be used instead of 7.5 by adding the sub-option `cuda8.0` to the `-ta=tesla` or `-Mcuda` compile- and link-time options.
- Improved support for the OpenACC **`cache`** directive.
- Added support for additional OpenACC 2.5 features:

  - Changed the behavior of the **`exit data`** directive to decrement the dynamic reference count.
  - Added the new optional **`finalize`** clause to set the dynamic reference count to zero.
  - Added the **`if_present`** clause to the **`update`** directive which changes the behavior when data is not present from a runtime error to a no-op.
  - Added new **`init`**, **`shutdown`**, and **`set`** directives.
  - Added new API routines to get and set the default async queue value.
  - Added support for the new definition of **`routine bind`** clause.
  - Updated the value of **`_OPENACC`** to 201510.

  With the exception of nested parallelism, declare link, and adding restrictions to **`cache`** clause variable refs to variables within cached region, OpenACC 2.5 feature support is complete in PGI 2017.

2.2. Command-line Environment

The PGI compilers for OpenPOWER are command-line compatible with the corresponding PGI products on Linux x86-64, meaning target-independent compiler options should behave consistently across the two platforms. The intent and expectation is that makefiles and build scripts used to drive PGI compilers on Linux x86-64 should work with little or no modification on OpenPOWER. The -help compiler option lists all compiler options by default, or can be used to check the functionality of a specific option. For example:

```
% pgcc -help -fast
Reading rcfile /opt/pgi/linuxpower/17.10/bin/.pgccrc
-fast               Common optimizations; includes -O2 -Munroll=c:1 -Mlre -
Mautoinline
                    == -Mvect=simd -Mflushz
-M[no]vect[=[no]simd|[no]assoc|[no]fuse]
                    Control automatic vector pipelining
    [no]simd        Generate [don't generate] SIMD instructions
    [no]assoc       Allow [disallow] reassociation
    [no]fuse        Enable [disable] loop fusion
-M[no]flushz        Set SSE to flush-to-zero mode
%
```

2.3. Fortran Language

The PGFORTRAN compiler supports Fortran 2003 features as defined in the document ISO/IEC 1539-1 : 2004, *Information technology – Programming Languages – Fortran*, Geneva, 2004 (Fortran 2003).

2.4. C Language

The PGCC compiler supports C99 and many of the important C11 language features as defined in the document ISO/IEC 9899:2013, *Information Technology – Programming Languages – C*, Geneva, 2011 (C11). In particular, thread-local storage, type generics, the **`_Noreturn`** specifier and **`__Alignof`**/**`__Alignas`** are all supported. Certain C11 features including anonymous **`structs`**/**`unions`**, **`_static_assert`**, atomics and unicode are not yet supported. PGCC supports compiler options -c89, -c99 and -c11 to enable/restrict support for/to C89, C99 and C11 respectively. Support for C99 is default. The -c11 compiler option must be used to enable support and processing of the C11 features.

2.5. C++ Language

The PGC++ compiler supports C++14 as defined in the document ISO/IEC 14882:2014, *Information Technology – Programming Languages – C++*, Geneva. The --c++14 compiler option must be used to enable support and processing of C++14 features, and the --c++11 compiler option must be used to enable support and processing of C++11 features. PGC++ is substantially compatible with GNU `g++` through GCC 6.3. In particular it uses GNU name-mangling conventions, uses GCC header files and libraries directly from a standard GCC installation, and is designed to be link-compatible with `g++`.

2.6. OpenACC

CUDA Unified Memory

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

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

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

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

This feature has the following limitations:

- Use of managed memory applies only to dynamically-allocated data. Static data (C static and extern variables, Fortran module, common block and save variables) and function local data is still handled by the OpenACC runtime. Dynamically allocated Fortran local variables and Fortran allocatable arrays are implicitly managed but Fortran array pointers are not.
- Given an allocatable aggregate with a member that points to local, global or static data, compiling with `-ta=tesla:managed` and attempting to access memory through that pointer from the compute kernel will cause a failure at runtime.
- C++ virtual functions are not supported.

- The `-ta=tesla:managed` compiler option must be used to compile the files in which variables are allocated, even if there is no OpenACC code in the file.

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

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

This feature is not supported on NVIDIA Fermi GPUs.

CUDA Unified Memory Pool Allocator

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

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

Table 2 Pool Allocator Environment Variables

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

Environment Variable	Use
PGI_ACC_POOL_ALLOC_MINSIZE	Set the minimum size for allocation blocks. The default size is 16B but other sizes can be used. The size must be greater than or equal to 16B.
PGI_ACC_POOL_THRESHOLD	Set the percentage of total device memory that the pool allocator can occupy. The default is set to 50% but other percentages can be used.

Multicore Support

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

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

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

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

Default Compute Capability

The default compute capability list for OpenACC and CUDA Fortran compilation for NVIDIA Tesla targets is cc30, cc35, and cc50. If CUDA 8.0 is specified using the `cuda8.0` sub-option, cc60 is added to the default compute capability list. Similarly, if CUDA 9.0 is specified using the `cuda9.0` sub-option, cc70 is added to the default compute capability list. The generation of device code can be time consuming, so you may notice an increase in compile time. You can override the default by specifying one or more compute capabilities using either command-line options or an `rcfile`.

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

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

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

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

OpenACC 2.5

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

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

Support for Profiler/Trace Tool Interface

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

2.7. OpenMP

OpenMP 3.1

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

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.

Current limitations include:

- The **`simd`** construct is ignored, with the exception that the **`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 **`taskloop`** construct's **`firstprivate`** and **`lastprivate`** clauses are not supported.

- 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.

Chapter 3. INSTALLATION AND CONFIGURATION

Follow these steps to install PGI 17.10 compilers on an OpenPOWER system. The default installation directory is `/opt/pgi`, but it can be any directory:

```
% tar zxpf pgilinux-2017-1710-ppc64le.tar.gz
% ./install
<answer installation questions, assent to licenses>
...
```

Typically for this release, you will want to choose the following during the installation:

1. Choose a "Single-system install", not a "Network install".
2. Install the PGI software in the default `/opt/pgi` directory.
3. Install the CUDA toolkit.

   This installs CUDA components in the PGI directory tree, and will not affect a standard CUDA installation on the same system in any way.
4. Install the OpenACC Unified Memory Evaluation package.
5. Create links in the 2017 directory.

   This is the directory where CUDA is installed, along with example programs; links are created to the subdirectories of `/opt/pgi/linuxpower/17.10`.
6. Install Open MPI.

3.1. License Management

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

If you purchased a perpetual license and have obtained your new license key, either replace the contents of license.pgi with your new license key, or set the environment variable `LM_LICENSE_FILE` to the full path of the desired license file.

If you have not yet obtained your new license key, please consult your PGI order confirmation email for instructions for obtaining and installing your permanent license key. Contact PGI Sales at sales@pgroup.com if you need assistance.

Usage Logging: This release provides per-user records of most recent use in the `.pgiusage` subdirectory inside the main installation directory. Set the environment variable `PGI_LOG_DIRECTORY` to specify a different directory for usage logging.

3.2. Environment Initialization

Assuming the software is installed in `/opt/pgi`, use these commands in `csh` to initialize your environment for use of the PGI compilers:

```
% setenv PGI /opt/pgi
% setenv MANPATH "$MANPATH":$PGI/linuxpower/2017/man
% set path=($PGI/linuxpower/2017/bin $path)
% which pgc++
/opt/pgi/linuxpower/2017/bin/pgc++
%
```

In `bash`, `sh` or `ksh`, use these commands:

```
% export PGI=/opt/pgi
% export MANPATH=$MANPATH:$PGI/linuxpower/2017/man
% export PATH=$PGI/linuxpower/2017/bin:$PATH
% which pgc++
/opt/pgi/linuxpower/2017/bin/pgc++
%
```

The PGI directory structure is designed to accommodate co-installation of multiple PGI versions. When 17.10 is installed, it will be installed by default in the directory `/opt/pgi/linuxpower/17.10` and links can optionally be created to its sub-directories to make `17.10` default without affecting a previous (e.g., `16.10`) install. Non-default versions of PGI compilers that are installed can be used by specifying the -V<ver> option on the compiler command line.

3.3. Network Installations

PGI compilers for OpenPOWER may be installed locally on each machine on a network or they may be installed once on a shared file system available to every machine. With the shared file system method, after the initial installation you can run a simple script on each machine to add that system to the family of machines using the common compiler installation. Using this approach, you can create a common installation that works on multiple linuxpower systems even though each system may have different versions of *gcc/libc*.

Follow these steps to create a shared file system installation on OpenPOWER systems:

1. Create a commonly-mounted directory accessible to every system using the same directory path (for example, `/opt/pgi`).
2. Define a locally-mounted directory with a pathname that is identical on all systems. That is, each system has a local directory path with the same pathname (for example `/local/pgi/17.10/share_objects`). Runtime libraries which are *libc*-version dependent will be stored here. This will ensure that executable files built on one system will work on other systems on the same network.
3. Run the install script for the first installation:

```
% tar zxpf pgilinux-2017-1710-ppc64le.tar.gz
% ./install
<answer installation questions, assent to licenses>
...
```

At the "Please choose install option:" prompt, choose "Network install".

4. Once the initial PGI installation is complete, configure the environment as described in the preceding section.
5. On each subsequent system, follow these steps:

   a. Set up the environment as described in the preceding section.
   b. Run the `add_network_host` script which is now in your $PATH:

      ```
      $ add_network_host
      ```

      and the compilers should now work.

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 at https://www.pgicompilers.com/support/faq.htm.

4.1. Release Specific Limitations

The following PGI features are limited or are not implemented in the 17.10 release for OpenPOWER+Tesla:

- -Mipa is not enabled (no PGI inter-procedural analysis/optimization); the command-line option is accepted and silently ignored.
- -Mpfi/-Mpfo are not enabled (no profile-feedback optimization); the command-line options are accepted and silently ignored.

4.2. Profiler-related Issues

Some specific issues related to the PGI Profiler:

- Debugging information is not always available on OpenPower which can cause the profiler to crash when attempting to unwind the call-stack. We recommend you use the --cpu-profiling-unwind-stack off option to disable call-stack tracing if you encounter any problem profiling on OpenPower.
- 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 omit the following option: -gpu and add the options -x PAMI_DISABLE_CUDA_HOOK=1 and -disbale_gpu_hooks.

Chapter 5. CONTACT INFORMATION

You can contact PGI at:

20400 NW Amberwood Drive Suite 100
Beaverton, OR 97006

Or electronically using any of the following means:

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

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

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

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