HPC SDK Release Notes

Release 25.9

NVIDIA Corporation

Sep 23, 2025

Contents

NVIDIA HPC SDK Release Notes

Welcome to version 25.9 of the NVIDIA HPC SDK, a comprehensive suite of compilers and libraries enabling developers to program the entire HPC platform, from the GPU foundation to the CPU and out through the interconnect. The 25.9 release of the HPC SDK includes component updates as well as important functionality and performance improvements.

- HPC SDK 25.9 supports CUDA 12.x and introduces support for CUDA 13.0. The 25.9 release packages include components from CUDA 13.0 and 12.9U1. CUDA 11.x is no longer provided as part of this HPC SDK release; however the compilers will continue to be tested on CUDA 11.x until a future release.
- HPC SDK 25.9 adds support for the RHEL/Rocky 10 Operating System distribution.
- Maxwell, Pascal, and Volta GPUs are no longer supported starting with CUDA 13.0.
- cuSOLVERMp and cuBLASMp have transitioned from using the Communication Abstraction Library (libcal) to using NCCL directly. These libraries should now be able to run on Cray/HPE Slingshot. This is a breaking change and requires changes to initialization in the user application. For cuSOLVERMp, see Migrating from CAL to NCCL , and for cuBLASMp, see Migrating from CAL to NCCL for steps to transition the application from libcal to NCCL.
- The following environment variables can be used to point to components outside the HPC SDK: `NVHPC_CUDA_HOME`, `NVCOMPILER_MATH_LIBS_HOME`, `NVCOMPILER_COMM_LIBS_HOME`, `NVCOMPILER_NCCL_HOME`, `NVCOMPILER_SHMEM_HOME`, `NVCOMPILER_CUPTI_LIBS_HOME`, `NVCOMPILER_NSIGHT_COMPUTE_HOME`, `NVCOMPILER_NSIGHT_SYSTEMS_HOME`, `NVCOMPILER_COMPUTE_SANITIZER_HOME`. As an example, these can be used to point to system CUDA 11.8 components, which can then be used with the nvhpc compiler within the HPC SDK. For more information on these environment varaibles, see the NVIDIA HPC Compilers User's Guide.
- Several new environment variables and API functions have been added to the compiler to enhance use of unified memory, and to add additional thread limit control. For information on the environment variable additions (`NVCOMPILER_ACC_MEMHINTS`, `NVCOMPILER_ACC_MEMPREFETCH`, `NVCOMPILER_ACC_CHECK_UNIFIED`, `NVCOMPILER_CPU_HARD_THREAD_LIMIT`), or API function additions (`accx_set_mem_hints`, `accx_set_mem_prefetch`, `accx_mem_advise`, `accx_mem_prefetch`), see NVIDIA HPC Compilers User's Guide.

Chapter 1. Release Component Versions

The NVIDIA HPC SDK 25.9 release contains the following versions of each component:

Table 1: HPC SDK Release Components

	Linux_x86_64		**Linux_aarch64**	
	CUDA 12.9U1	CUDA 13.0	CUDA 12.9U1	CUDA 13.0
nvc++	25.9		25.9	
nvc	25.9		25.9	
nvfortran	25.9		25.9	
nvcc	12.9.37	13.0.48	12.9.37	13.0.48
NCCL	2.26.5 (12.0-12.1) 2.27.7 (12.2+)	2.27.7	2.26.5 (12.0-12.1) 2.27.7 (12.2+)	2.27.7
NVSHMEM	3.3.24	3.3.24	3.3.24	3.3.24
cuBLAS	12.9.1.4	13.0.0.19	12.9.1.4	13.0.0.19
cuBLASMp	0.5.1	0.5.1 (cc80+)	0.5.1	0.5.1 (cc80+)
cuFFT	11.4.1.4	12.0.0.15	11.4.1.4	12.0.0.15
cuFFTMp*	11.4.0	11.4.0	11.4.0	11.4.0
cuRAND	10.3.10.19	10.4.0.35	10.3.10.19	10.4.0.35
cuSOLVER	11.7.5.82	12.0.3.29	11.7.5.82	12.0.3.29
cuSOLVERMp*	0.7.0	0.7.0 (cc80+)	0.7.0	0.7.0 (cc80+)
cuSPARSE	12.5.10.65	12.6.2.49	12.5.10.65	12.6.2.49
cuTENSOR	2.2.0 (<cc70) 2.3.0 (>=cc70+)	2.3.0	2.2.0 (<cc70) 2.3.0 (>=cc70)	2.3.0
Nsight Compute	2025.2.1	2025.3	2025.2.1	2025.3
Nsight Systems	2025.5.1		2025.5.1	
HPC-X	2.20 (12.0-12.2) 2.24 (12.3+)	2.24	2.20 (12.0-12.2) 2.24 (12.3+)	2.24
OpenBLAS	0.3.23		0.3.23	
Scalapack	2.2.0		2.2.0	
Thrust	2.8.2	3.0.1	2.8.2	3.0.1
CUB	2.8.2	3.0.1	2.8.2	3.0.1
libcu++	2.8.2	3.0.1	2.8.2	3.0.1
NVPL*	N/A		25.5	

* product in beta

Chapter 2. Supported Platforms

2.1. Platform Requirements for the HPC SDK

Table 2: HPC SDK Platform Requirements

Architecture	Linux Distributions	Minimum gcc/glibc Toolchain	Minimum CUDA Driver
x86_64	RHEL/CentOS/Rocky 8.0 - 8.10 RHEL/Rocky 9.2 - 9.6, 10 OpenSUSE Leap 15.4 - 15.6 SLES 15SP4, 15SP5, 15SP6, 15SP7 Ubuntu 22.04, 24.04 Debian 12, 13	Fortran, C, and up to C++17: 7.5 C++20: 10.1 C++23: 12.1	12.x: >=525.60.13 13.x: >=580.65.06
aarch64	RHEL/CentOS/Rocky 8.0 - 8.10 Rocky 9.2 - 9.6, 10 Ubuntu 22.04, 24.04 SLES 15SP6, 15SP7 Amazon Linux 2023	Fortran, C, and up to C++17: 7.5 C++20: 10.1 C++23: 12.1	12.x: >=525.60.13 13.x: >=580.65.06

Programs generated by the HPC Compilers for x86_64 processors require a minimum of AVX instructions, which includes Sandy Bridge and newer CPUs from Intel, as well as Bulldozer and newer CPUs from AMD. The HPC SDK includes support for v8.1+ Server Class Arm CPUs that meet the requirements appendix E specified in the SBSA 7.1 specification.

The HPC Compilers are compatible with gcc and g++ and use the GCC C and C++ libraries; the minimum compatible versions of GCC are listed in the table in Section 3. The minimum system requirements for CUDA and NVIDIA Math Library requirements are available in the NVIDIA CUDA Toolkit documentation.

2.2. Supported CUDA Toolchain Versions

The NVIDIA HPC SDK uses elements of the CUDA toolchain when building programs for execution with NVIDIA GPUs. Every HPC SDK installation package puts the required CUDA components into an installation directory called [install-prefix]/[arch]/[nvhpc-version]/cuda.

An NVIDIA CUDA GPU device driver must be installed on a system with a GPU before you can run a program compiled for the GPU on that system. The NVIDIA HPC SDK does not contain CUDA drivers. You must download and install the appropriate CUDA driver from NVIDIA , including the CUDA Compatibility Platform if that is required.

The `nvaccelinfo` command prints the CUDA Driver version in its output. You can use it to find out which version of the CUDA Driver is installed on your system.

The NVIDIA HPC SDK 25.9 includes the following CUDA toolchain versions:

- CUDA 12.9U1
- CUDA 13.0

The minimum required CUDA driver versions are listed in the table in Section 3.1.

Chapter 3. Known Limitations and Recommendations

The following are recommendations for more effectively using the HPC SDK and its components when unexpected behavior or suboptimal performance is encountered.

- HPC Compilers
  - When using nvfortran with `-g` and mixing Blackwell and non-Blackwell compute capabilities in the same fat binary, `-gpu=nodebug` is implied. When `-g` support on the device is needed, users can specify Blackwell-only compute capability support using the `-gpu` flag and one or more Blackwell sub-options (i.e., `cc100, cc120`).
  - For nvfortran, the `IOSTAT` argument of defined input/output procedures is expected to be of default kind `INTEGER`. `IOSTAT` declared to be other than the default kind may experience undefined behavior at runtime.
  - When a pointer is assigned to an array dummy argument with the target attribute, nvfortran may associate the pointer with a copy of the array argument instead of the actual argument.
  - Passing an internal procedure as an actual argument to a Fortran subprogram is supported by nvfortran provided that the dummy argument is declared as an interface block or as a procedure dummy argument. nvfortran does not support internal procedures as actual arguments to dummy arguments declared external.
  - nvfortran only supports the Fortran 2003 standard maximum of 7 dimensions for arrays (Fortran 2008 raised the standard maximum dimensions to 15). This limit is defined in the standard CFI_MAX_RANK macro in the ISO_Fortran_binding.h C header file.
  - Section "15.5.2.4 Ordinary dummy variables", constraint C1540 and Note 5 in the Fortran 2018 Standard allow Fortran compilers to avoid copy-in/copy-out argument passing provided that the actual and corresponding dummy arguments have the ASYNCHRONOUS/VOLATILE attribute, and the dummy arguments do not have the VALUE attribute. This feature is fully supported in nvfortran with BIND(C) interfaces (i.e., Fortran calling C). Copy-in/copy-out avoidance with asynchronous/volatile attributes may not be available in other cases with nvfortran.
  - Fortran derived type objects with zero-size derived type allocatable components that are used in sourced allocation or allocatable assignment may result in a runtime segmentation violation.
  - When using `-stdpar` to accelerate C++ parallel algorithms, the algorithm calls cannot include virtual function calls or function calls through a function pointer, cannot use C++ exceptions, and must use random access iterators (raw pointers as iterators work best). When unified memory is not enabled, the algorithm calls can only dereference pointers that point to the heap. See the C++ parallel algorithms documentation for more details.

    - There is a known bug in glibc versions 2.34 to 2.38 (inclusive) that can negatively impact performance of `malloc()` when called from inside OpenMP regions and combined with `OMP_PROC_BIND`. While a fix has been backported into those versions of glibc, it is not available for many Linux distributions. The OpenMP runtime will automatically set the value of the `MALLOC_ARENA_MAX` environment variable to 8 times the value of `OMP_NUM_THREADS` if `MALLOC_ARENA_MAX` is not already set. `MALLOC_ARENA_MAX` may be set to `0` to disable the automatic workaround and use the default glibc behavior.

- Communication libraries (HPC-X MPI, OpenSHMEM, UCX, …)
    - HPC SDK 25.9 defaults to using HPC-X version 2.24 which is incompatible with CUDA 12.0 - 12.2 driver (R525). HPC-X 2.20 is available as a fallback for users requiring CUDA 12.0 - 12.2. HPC-X 2.20 can be selected by loading the `nvhpc-hpcx-2.20-cuda12` environment module.
    - HPC-X MPI initialization time on systems with CUDA may be higher than on systems without CUDA installed. If your application does not use GPU communication, you may be able to reduce the initialization overhead by setting the MPI environment variables `OMPI_MCA_coll_ucc_enable=0` and `UCX_MODULES=^cuda`. Please be aware that disabling UCC may degrade performance in other areas of HPC-X MPI, so we recommend testing overall performance changes with these settings.
    - Both NVSHMEM and NCCL rely on GPUDirect RDMA for direct GPU-to-GPU communication within a node. To achieve the best performance on bare metal Linux platforms, the GPUDirect Storage Best Practice Guide recommends that system settings like PCIe Access Control Services (ACS) and Input-Output Memory Management Units (IOMMUs) be disabled or set to passthrough mode. The NCCL documentation also suggests that ACS and IOMMUs be disabled, citing that they could cause a significant performance reduction or even a hang.
    - Any program data specified in `acc declare create` (and related clauses such as `copyin`, `device_resident`) can cause an application crash if used in an HPC-X MPI transport.
    - The MPI wrappers in `comm_libs/mpi/bin` automatically detect the CUDA driver and select the matching MPI library from `comm_libs/X.Y`. Applications that require a full MPI directory hierarchy (e.g., bin, include, lib) or are launched via srun should bypass the MPI wrappers by loading the `nvhpc-hpcx-cuda11` or the `nvhpc-hpcx-cuda12` environment module, depending on the installed CUDA driver version.
    - To use HPC-X, please use the provided environment module files or take care to source the hpcx-init.sh script: `$ . ${NVHPCSDK_HOME}/comm_libs/X.Y/hpcx/latest/hpcx-init.sh` Then, run the hpcx_load function defined by this script: `hpcx_load`. These actions will set important environment variables that are needed when running HPC-X.
    - The following warning from HPC-X may be encountered due to oversubscription, failure to detect proper topology, etc., while running an MPI job – "WARNING: Open MPI tried to bind a process but failed. This is a warning only; your job will continue, though performance may be degraded". This may be suppressed as follows: `export OMPI_MCA_hwloc_base_binding_policy=""`
    - Starting with version 2.17.1, HPC-X does not have performance-optimal support for stream-ordered CUDA-allocated memory. In practical terms it means that IPC methods such as the MPI calls `MPI_Send` and `MPI_Recv` can have significantly degraded throughput when passed data allocated with the `cudaMallocAsync` function or its variants. This limitation will be removed in a future release.

- Math Libraries
    - cuSOLVERMp and cuBLASMp do not support Turing (cc75) with CUDA 13.0. Support for Turing will be added in a future release.

- cuBLASMp redistribution functionality (`cublasMpGemr2D` and `cublasMpTrmr2D`) may fail when the user provided host workspace is allocated as CUDA unified memory. This issue will be fixed in the next cuBLASMp release. To work around this, provide a host workspace that is not allocated using CUDA unified memory.
- cuSolverMp has two dependencies on UCC and UCX libraries in the HPC-X directory. To execute a program linked against cuSolverMP using CUDA 11.8, please use the "nvhpc-hpcx-cuda11" environment module for the HPC-X library, or set the environment variable LD_LIBRARY_PATH as follows: `LD_LIBRARY_PATH=${NVHPCSDK_HOME}/comm_libs/11.8/hpcx/latest/ucc/lib:${NVHPCSDK_HOME}/comm_libs/11.8/hpcx/latest/ucx/lib:$LD_LIBRARY_PATH`
- Known issues related to NVPL are described in the NVPL documentation.

Chapter 4. Deprecations and Changes

- Starting with HPC SDK 25.9, the preprocessor macro `__HLE__` is unavailable by default. HLE refers to the x86_64 processor feature "Hardware Lock Elision". On Intel (x86_64) processors, the NVC and NVC++ compiler drivers had an inconsistency with the definition of the predefined (system) preprocessor macro. If the user specified an x86_64 target processor as a compiler option (for example: `-tp skylake`), the predefined preprocessor system object macro `__HLE__` was not defined (correct behavior). But when compiling on an Intel x86_64 processor without specifying the `-tp <SOME-Intel-PROC>`, the compiler queried the host processor to see if the HLE hardware feature was present, and if it was, the preprocessor system macro `__HLE__` was defined (incorrect behavior). The compiler option `-mhle` can be used to override the default behavior and force the system preprocessor macro `__HLE__` to be defined.
- Architecture support for Maxwell, Pascal, and Volta is considered feature complete. Offline compilation and library support for these architectures have been removed in CUDA Toolkit 13.0 major version release. The use of CUDA Toolkits through the 12.x series to build applications for these architectures will continue to be supported, but newer toolkits will be unable to target these architectures.
- The nvvp and nvprof utilities are deprecated and have been removed from HPC SDK 25.9. Users of nvvp and nvprof are recommended to use NSight Systems and Nsight Compute.
- The Open MPI 4 library has been removed in HPC SDK 25.9. Using HPC-X MPI is recommended.
- The `CUDA_HOME` environment variable is ignored by the HPC Compilers. It is replaced by `NVHPC_CUDA_HOME`.
- Support for using stdpar with C++14 and below has been deprecated; C++17 or higher is required when using stdpar.
- `CUDA_VISIBLE_DEVICES` is not supported at compile time. This environment variable remains effective at application runtime. To affect code generation when compiling on systems with multiple GPU architectures, use the `-gpu=ccXY` option.

Notices

Notice and Disclaimers

Trademarks

Copyright