HPC SDK Release Notes

Release 26.3

NVIDIA Corporation

Mar 25, 2026

Contents

NVIDIA HPC SDK Release Notes

Welcome to version 26.3 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 26.3 release of the HPC SDK includes component updates as well as important functionality and performance improvements.

- HPC SDK 26.3 supports CUDA 13.x and CUDA 12.x. The 26.3 release packages include components from CUDA 13.1U1 and 12.9U1.
- Components updated in this release include: CUDA Toolkit components, cuBLASMp, cuTensor, NCCL, Nsight Compute, Nsight Systems, and HPC-X.
- The nvc++ compiler now includes the `-std=c++26` command-line flag to activate C++26 mode. In this specific release, the primary language updates are limited to the addition of the `#embed` preprocessor directive and the implementation of erroneous behavior for uninitialized local variables.
- The nvc++ compiler now features an implementation of C++26 `std::execution`, frequently referred to as Senders and Receivers or stdexec. This framework introduces a standardized approach for writing structured asynchronous code within standard C++. You can access these features by compiling in C++20 mode or later and including the `<execution>` header, with no additional configuration required.
  - Please note that because the C++26 specification is not yet finalized, the API is not fully guaranteed. While this implementation aligns with the current draft standard and any future adjustments are expected to be minor, some source code changes may be necessary in subsequent releases.
  - In addition to the draft standard features, this release includes non-standard components from the stdexec project. Notable additions include `exec::static_thread_pool`, found in `<exec/static_thread_pool.hpp>` for CPU-based parallel execution, and `nvexec::stream_context`, found in `<nvexec/stream_context.cuh>` for GPU-based parallel execution. To execute any `std::execution` or stdexec code on a GPU, you must include the `-stdpar=gpu` flag during compilation.
- The nvc and nvc++ compilers now support the `-ftrivial-auto-var-init` command-line option, which allows you to define how the system handles uninitialized variables using the uninitialized, zero, or pattern arguments. Detailed information regarding these settings can be found in the command-line option reference guide.
  - When operating in C++26 mode, the compiler defaults to the pattern setting. This specific configuration implements the C++26 standard for erroneous values, which helps identify or mitigate issues with variables that have not been explicitly initialized. For all versions of C and all C++ standards prior to C++26, the default behavior remains set to uninitialized.
- The C++ `mdspan` and stdBLAS (`linalg`) libraries, previously shipped as experimental features, are now fully productized. APIs have moved from the `std::experimental` namespace to the std namespace, aligning with the C++23 and C++26 standards (with small exceptions; please see below). The `<mdspan>` header provides multi-dimensional array views and the `<linalg>` header provides C++ standard linear algebra operations. Accelerated back-ends using cuBLAS (GPU) and OpenBLAS/NVPL (multicore CPU), enabled via the `-stdpar` flag, continue to be supported. Users migrating from the experimental versions should update their codes to use the `std` namespace (e.g., `std::mdspan`, `std::linalg::matrix_product`) and the new standard headers. Please refer to the C++ Parallel Algorithms section of the User's Guide for details. Examples can be found in the `$NVHPC_HOME/examples/stdpar/stdblas` directory.
- The C++26 `submdspan` feature implements slicing (viewing a subset of elements) of `mdspan`, the multidimensional array view class template added in C++23. C++26 will not be finalized until after

this product release date, so any reference to C++26 in this release is speculative. That being said, our implementation of `submdspan` currently differs from the expected C++26 Standard in two ways.

  - The proposal P3663 is not implemented. P3663 was merged into the C++26 Draft in November, 2025. This change affects users who write custom layout mappings and want them to work with submdspan.
  - The proposal P3982 is not implemented. This proposal, which is still in review and subject to change, would rename the "strided_slice" slice type to "extent_slice," change the meaning of its "extent" member, and add a new "range_slice" slice type. These changes may not be accepted into the C++26 standard.

- The NVIDIA Performance Libraries (NVPL) now serve as the default CPU BLAS, LAPACK, and ScaLAPACK implementations for Arm processors. When you use standard flags such as `-lblas`, the compiler will automatically link against the NVPL headers and libraries. You can find additional technical details and configuration options in the NVPL documentation.
  - The OpenBLAS libraries remain available within HPC SDK for Arm processors and can be activated using the new `-Mmathlib=openblas` compiler flag. Alternatively, you may continue to access them by manually defining the header and library search paths along with their respective names. This option is supported on both Arm and x86_64 architectures to ensure cross-platform compatibility.
- The new `FORT_NO_EMPTY_ALLOCATION` environment variable allows the Fortran runtime to trap non-positive array allocations. While the default setting of 0 preserves the standard behavior of allowing zero or negative sizes, setting the variable to a non-zero integer causes the `ALLOCATE` statement to trigger an error instead. This provides a stricter debugging mechanism for identifying unexpected allocation sizes during execution.

Chapter 1. Release Component Versions

The NVIDIA HPC SDK 26.3 release contains the following versions of each component:

Table 1: HPC SDK Release Components

	Linux_x86_64		**Linux_aarch64**	
	CUDA 12.9U1	CUDA 13.1U1	CUDA 12.9U1	CUDA 13.1U1
nvc++	26.3		26.3	
nvc	26.3		26.3	
nvfortran	26.3		26.3	
nvcc	12.9.79	13.1.80	12.9.79	13.1.80
NCCL	2.29.3	2.29.3	2.29.3	2.29.3
NVSHMEM	3.5.19	3.5.19	3.5.19	3.5.19
cuBLAS	12.9.1.4	13.2.1.1	12.9.1.4	13.2.1.1
cuBLASMp	0.8.0	0.8.0 (cc80+)	0.8.0	0.8.0 (cc80+)
cuFFT	11.4.1.4	12.1.0.78	11.4.1.4	12.1.0.78
cuFFTMp*	11.4.0	12.1.3	11.4.0	12.1.3
cuRAND	10.3.10.19	10.4.1.381	10.3.10.19	10.4.1.81
cuSOLVER	11.7.5.82	12.0.9.81	11.7.5.82	12.0.9.81
cuSOLVERMp*	0.7.2	0.7.2 (cc80+)	0.7.2	0.7.2 (cc80+)
cuSPARSE	12.5.10.65	12.7.3.1	12.5.10.65	12.7.3.1
cuTENSOR	2.2.0 (<cc70) 2.6.0 (>=cc70+)	2.6.0	2.2.0 (<cc70) 2.6.0 (>=cc70)	2.6.0
Nsight Compute	2025.2.1	2025.4.1	2025.2.1	2025.4.1
Nsight Systems	2025.3 (cc70) 2026.1.1	2026.1.1	2025.3 (cc70) 2026.1.1	2026.1.1
HPC-X	2.20 (12.0-12.2) 2.25.1 (12.3+)	2.25.1	2.20 (12.0-12.2) 2.25.1 (12.3+)	2.25.1
OpenBLAS	0.3.23		0.3.23	
ScaLAPACK	2.2.0		2.2.0	
Thrust	2.8.2	3.1.4	2.8.2	3.1.4
CUB	2.8.2	3.1.4	2.8.2	3.1.4
libcu++	2.8.2	3.1.4	2.8.2	3.1.4
NVPL*	N/A		25.11	

* 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, 10.0 - 10.1 OpenSUSE Leap 15.4 - 15.6 SLES 15SP4, 15SP5, 15SP6, 15SP7 Ubuntu 22.04, 24.04 Debian 12, 13 Fedora 40, 41, 42 CentOS Stream 9, 10	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, 10.0 - 10.1 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 in 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 26.3 includes the following CUDA toolchain versions:

- CUDA 12.9U1
- CUDA 13.1U1

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`).
  - When using Open MPI with nvfortran, mixing `use mpi` and `use mpi_f08` in the same application is not supported. A potential workaround is to add compiler option `-Msecond_underscore` to your build and perform a clean rebuild of the application and libraries.
  - 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 26.3 defaults to using HPC-X version 2.25.1 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.
  - Starting with version 2.24, HPC-X requires the installed GDRCopy driver to be at least of version 2.5. This is only required when the application needs to use the GDRCopy transport.
  - 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-cuda13` 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
  - Known issues related to NVPL are described in the NVPL documentation.
  - NVPL does not include some routines that were in the prior math libraries, including LAPACK testing routines (e.g., clarge), LAPACK internal helper routines (e.g., dladiv1), and OpenBLAS specific extensions (e.g., ismin, zaxpyc, csbmv). Also, NVPL provides BLACS as a separate library, whereas the prior ScaLAPACK library included BLACS in libscalapack.so.

Chapter 4. Deprecations and Changes

- CUDA 11.x is no longer supported as of the 26.1 release.
- The `-Mnvpl` option is deprecated. Use `-Mmathlib` instead.
- The following libraries are deprecated:
  - `libnvcpumath-avx.so`
  - `libnvcpumath-avx2.so`
  - `libnvcpumath-avx512.so`
  - `libnvcpumath-armv8.so`

  Currently, these libraries are replaced with symbolic links to the library libnvcpumath.so to preserve backward compatibility. In future releases, the symbolic links will be removed.
- Starting with HPC SDK 26.1, the following HPC SDK URLs have changed to align with DevZone strategy to improve consistency and cues to viewers:
  - The page formerly at https://developer.nvidia.com/hpc-sdk-downloads is now at https://developer.nvidia.com/hpc-sdk/downloads
  - Same for https://developer.nvidia.com/hpc-sdk-releases
  - The 26.1 download page never existed but the pattern matches past practice. The new format going forward is https://developer.nvidia.com/hpc-sdk/releases/26.3
- 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

Copyright