



HPC SDK Release Notes

Release 26.11

NVIDIA Corporation

Nov 17, 2026

Contents

1 Component Updates **3**

2 Highlights **5**

3 Platform Support Changes **7**

4 Performance Improvements **9**

5 Release Component Versions **11**

6 Supported Platforms **15**
6.1 Platform Requirements for the HPC SDK 15
6.2 Supported CUDA Toolchain Versions 16

7 Known Limitations and Recommendations **17**

8 Deprecations and Changes **21**

NVIDIA HPC SDK Release Notes

Welcome to version 26.11 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 26.11 release of the HPC SDK includes component updates as well as important functionality and performance improvements.

- HPC SDK 26.11 supports CUDA 13.x and CUDA 12.x. The 26.11 release packages include components from CUDA 13.4U1 and 12.9U1.
- CUDA 13.4, driver R615, changes how GPU memory is managed by default on coherent systems i.e. GH200, GB200, GB300, and Vera Rubin.
  - These systems support two modes. In NUMA mode, Linux treats GPU memory as part of system memory and can move data into it automatically. In CDMM mode, data stays in CPU memory and the GPU accesses it over the NVLink-C2C link. Until R615, NUMA was the default; starting with R615, CDMM is the default. A system runs in one mode or the other, set per node. For more information on how to check and change the default mode for a system, see blog post Understanding Memory Management on Hardware-Coherent Platforms
  - HPC applications work correctly in both modes. Performance will remain unchanged for OpenACC, OpenMP, and Stdpar applications compiled for Separate or Managed Memory Mode. There may be a performance degradation when running Unified Memory HPC applications in CDMM mode. For further details see Coherent Driver-based Memory Management (CDMM)
  - We recommend that HPC systems continue to use NUMA mode.

Chapter 1. Component Updates

Component	Version
CUDA	13.4 Update 1
NCCL	2.32.3, including Fortran interfaces for `ncclSetEncryption` and the new 2.32 types and constants
NVSHMEM	3.8.0.0, including Fortran interfaces for the region and counted-signal APIs
NVPL	26.10
cuTENSOR	2.8.1
CCCL (Thrust, CUB, libcu++)	3.4.3
Nsight Compute	2026.3.1
Nsight Systems	2026.5.1

- The NCCL Fortran module used a `ncclCommProperties` layout that was too small, so `ncclCommQueryProperties` could write past the end of the caller's variable. This is fixed.

Chapter 2. Highlights

- The NVHPC compilers add new features for tuning GPU memory bandwidth. For details, see OpenACC Memory Bandwidth Tuning
- The NVHPC Fortran compiler now supports the `unroll` modifier on the OpenACC `vector` and `seq` loop clauses, as in `vector(unroll:n)` and `seq(unroll:n)`, to set the unroll factor of a loop explicitly.
- The NVIDIA C++ compiler supports virtual function calls in OpenACC compute regions when the function written at the call has an explicit `#pragma acc routine` directive and the object is constructed in device code. Each override that can run on the device needs the same directive. A host constructor stores the host virtual-table address in the object, and copying the object to the device copies that address. A virtual call with no explicit directive, and a call through a function pointer or a pointer to a member, remain accelerator restrictions. The compiler reports them with `-Minfo=accel` and leaves the region on the host.
- The OpenACC `cache` directive includes functionality improvements and can now use the Tensor Memory Accelerator (TMA) on GPUs with compute capability 9.0 and later; enable TMA with `-gpu=cache:tma[:n]`, where `n` sets the minimum section size in bytes, and disable it with `-gpu=cache:notma`.
- The NVHPC Fortran compiler now supports the OpenACC `prefetch` directive, which preloads array sections into the L1 cache; it requires CUDA 13.4 or later.
- The NVHPC compilers now support the `__nv_minctapersm(n)` clause on the OpenACC `parallel`, `kernels`, and `serial` constructs, which sets the minimum number of thread blocks per SM for the generated kernels.
- A new NVFORTRAN option, `-Mmoddeps`, prints the absolute path of every `.mod` file a source file depends on, including indirect USE dependencies, one per line. Compilation stops after semantic analysis and no object file is produced. The option is available on both x86_64 and Arm.
- OpenMP now supports C++ range-based `for` loops over arrays and pointer ranges with `parallel for`, `distribute parallel for`, `loop`, and `collapse`.
- OpenMP now supports host `long double` reductions, `volatile` scalars in reduction clauses, and `!=` as a canonical loop test in either loop direction.
- GNU `asm goto` is now supported in NVC and NVC++.

Chapter 3. Platform Support Changes

- Programs generated by the HPC Compilers for x86_64 processors require a minimum of AVX2 instructions, which includes Haswell and newer CPUs from Intel, as well as Excavator and newer CPUs from AMD. Target processors without AVX2 are no longer supported.
- Including `openacc_lib.h` in Fortran now emits a deprecation warning. The header was removed from the OpenACC specification in version 2.5. Use the `openacc` module instead.

Chapter 4. Performance Improvements

- Vera (Neoverse-V3) math library gather costs are modeled more accurately, and short loops over stack arrays are no longer versioned.
- On Arm, the math library now selects its Armv9 code path from the detected CPU architecture, so NVIDIA Vera and future Armv9 processors use the Armv9 path.
- Faster kernels in the Arm math library:
  - SVE and NEON kernels for single-precision `exp`
  - Advanced SIMD `asin`, `acos`, `sin`, `cos`, `sincos`, `logf`, and `log10f`
  - Scalar Fortran `MOD` of whole-number real operands no longer calls `fmod`.

Chapter 5. Release Component Versions

The NVIDIA HPC SDK 26.11 release contains the following versions of each component:

Table 1: HPC SDK Release Components

	Linux_x86_64		**Linux_aarch64**	
	CUDA 12.9U1	CUDA 13.4U1	CUDA 12.9U1	CUDA 13.4U1
nvc++	26.11		26.11	
nvc	26.11		26.11	
nvfortran	26.11		26.11	
nvcc	12.9.79	13.4.92	12.9.79	13.4.92
NCCL	2.32.3	2.32.3	2.32.3	2.32.3
NVSHMEM	3.8.0.0	3.8.0.0	3.8.0.0	3.8.0.0
cuBLAS	12.9.2.10	13.8.046	12.9.2.10	13.8.046
cuBLASMp	0.10.0	0.10.0(cc80+)	0.10.0	0.10.0(cc80+)
cuFFT	11.4.1.4	12.4.0.43	11.4.1.4	12.4.0.43
cuFFTMp*	11.4.0	12.1.3	11.4.0	12.1.3
cuRAND	10.3.10.19	10.4.4.72	10.3.10.19	10.4.4.72
cuSOLVER	11.7.5.82	12.3.5.3	11.7.5.82	12.3.5.3
cuSOLVERMp*	0.9.1.0	0.9.1.0 cc80+	0.9.1.0	0.9.1.0 cc80+
cuSPARSE	12.5.10.65	12.8.6.72	12.5.10.65	12.8.6.72
cuTENSOR	2.2.0 (cc70) 2.8.1 (>cc70)	2.8.1	2.2.0 (<cc70) 2.8.1 (>=cc70)	2.8.1
Nsight Compute	2025.2.1	2026.2.1	2025.2.1	2026.3.1
Nsight Systems	2025.3 (cc70) 2026.4.1	2026.4.1	2025.3 (cc70) 2026.5.1	2026.5.1
HPC-X	2.20 (12.0-12.2) 2.50 (12.3+)	2.50	2.20 (12.0-12.2) 2.50 (12.3+)	2.50
OpenBLAS	0.3.34		0.3.34	
ScaLAPACK	2.2.0		2.2.0	
Thrust	3.4.2	3.4.2	3.4.2	3.4.3
CUB	3.4.2	3.4.2	3.4.2	3.4.3
libcu++	2.8.2	3.4.2	2.8.2	3.4.3
NVPL	N/A		26.10	

* product in beta

Chapter 6. Supported Platforms

6.1. 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 - 9.8, 10.0 - 10.2 OpenSUSE Leap 15.6 - 16.0 SLES 15SP6, 15SP7, 16.0 Ubuntu 22.04, 24.04, 26.04 Debian 12, 13 Fedora 41 - 44 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, 26.04 SLES 15SP6, 15SP7, 16.0 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 AVX2 instructions, which includes Haswell and newer CPUs from Intel, as well as Excavator 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 2.1. The minimum system requirements for CUDA and NVIDIA Math Library requirements are available in the NVIDIA CUDA Toolkit documentation.

6.2. 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 26.11 includes the following CUDA toolchain versions:

- CUDA 12.9U1
- CUDA 13.4U1

The minimum required CUDA driver versions are listed in the table in Section 2.1.

Chapter 7. 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 26.11 defaults to using HPC-X version 2.50 which is incompatible with the CUDA 12.0 - 12.3 driver (R545). HPC-X 2.20 is available as a fallback for users requiring CUDA 12.0 - 12.3. HPC-X 2.20 can be selected by loading the `nvhpc-hpcx-2.20-cuda12` environment module.
  - Starting with Open MPI v5.0 in HPC-X 2.50, the default runtime environment has transitioned from the Open Run-Time Environment (ORTE) to the PMIx Reference RunTime Environment (PRRTE).
    - Parallel job launch, monitoring, and both processor and memory affinity management are now fully handled by PRRTE.
    - Along with this runtime change, the default process affinity mapping has shifted from map-by-socket to map-by-core under PRRTE.
    - For more information on processor and memory affinity, see Open MPI Affinity Guide
    - For more information on the role of PMIx and PRRTE, see: Open MPI v5.0.x Launching Applications Guide
  - 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.

Chapter 8. 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 standalone 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 removed; 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

All information provided in this document is provided as-is, for your informational purposes only and is subject to change at any time without notice. Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing. To obtain the latest information, please contact your NVIDIA representative. Product or service performance varies by use, configuration and other factors. Your costs and results may vary. No product or component is absolutely secure. TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, NVIDIA DISCLAIMS ALL WARRANTIES AND REPRESENTATIONS OF ANY KIND, WHETHER EXPRESS, IMPLIED OR STATUTORY, RELATING TO OR ARISING UNDER THIS DOCUMENT, INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF TITLE, NONINFRINGEMENT, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, USAGE OF TRADE AND COURSE OF DEALING. NVIDIA products are not intended or authorized for use as critical components in a system or application where the use of or failure of such system or application developed with products, technology, software or services provided by NVIDIA could result in injury, death or catastrophic damage.

Except for your permitted use of the information contained in this document, no license or right is granted by implication, estoppel or otherwise. If this document directly includes or links to third-party websites, products, services or information, please consult other sources to evaluate if and how to use that information since NVIDIA does not support, endorse or assume any responsibility for any third party offerings or its accuracy or usefulness.

TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY (I) INDIRECT, PUNITIVE, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, OR (II) DAMAGES FOR THE (A) COST OF PROCURING SUBSTITUTE GOODS OR (B) LOSS OF PROFITS, REVENUES, USE, DATA OR GOODWILL ARISING OUT OF OR RELATED TO THIS DOCUMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, OR OTHERWISE, AND EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES AND EVEN IF A PARTY'S REMEDIES FAIL THEIR ESSENTIAL PURPOSE. ADDITIONALLY, TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, NVIDIA'S TOTAL CUMULATIVE AGGREGATE LIABILITY FOR ANY AND ALL LIABILITIES, OBLIGATIONS OR CLAIMS ARISING OUT OF OR RELATED TO THIS DOCUMENT WILL NOT EXCEED FIVE U.S. DOLLARS (US$5).

Statements in this document that refer to future plans or expectations are forward-looking statements. These statements are based on currently available information, beliefs, assumptions and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in these statements. For more information on the factors that could cause actual results to differ materially, see our most recent earnings release and SEC filings at NVIDIA Corporation SEC Filings.

© NVIDIA Corporation. All rights reserved. NVIDIA, the NVIDIA logo, and other NVIDIA marks are trademarks of NVIDIA Corporation or its affiliates. Other names and brands may be claimed as the property of others.

Trademarks

NVIDIA, the NVIDIA logo, CUDA, CUDA-X, GPUDirect, HPC SDK, NGC, NVIDIA Volta, NVIDIA DGX, NVIDIA Nsight, NVLink, NVSwitch, and Tesla are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

©2022-2026, NVIDIA Corporation & affiliates. All rights reserved