# Hopper Compatibility Guide

*Release 12.9*

NVIDIA Corporation

Apr 18, 2025

# Contents

## Hopper Compatibility Guide for CUDA Applications

The guide to building CUDA applications for Hopper GPUs

# Chapter 1. About this Document

This application note, Hopper Architecture Compatibility Guide for CUDA Applications, is intended to help developers ensure that their NVIDIA® CUDA® applications will run on the NVIDIA® Hopper architecture based GPUs. This document provides guidance to developers who are familiar with programming in CUDA C++ and want to make sure that their software applications are compatible with Hopper architecture.

# Chapter 2. Application Compatibility on Hopper Architecture

A CUDA application binary (with one or more GPU kernels) can contain the compiled GPU code in two forms, binary cubin objects and forward-compatible PTX assembly for each kernel. Both cubin and PTX are generated for a certain target compute capability. A cubin generated for a certain compute capability is supported to run on any GPU with the same major revision and same or higher minor revision of compute capability. For example, a cubin generated for compute capability 8.0 is supported to run on a GPU with compute capability 8.6, however a cubin generated for compute capability 8.6 is *not* supported to run on a GPU with compute capability 8.0, and a cubin generated with compute capability 8.x is *not* supported to run on a GPU with compute capability 9.0.

Kernel can also be compiled to a PTX form. At the application load time, PTX is compiled to cubin and the cubin is used for kernel execution. Unlike cubin, PTX is forward-compatible. Meaning PTX is supported to run on any GPU with compute capability higher than the compute capability assumed for generation of that PTX. For example, PTX code generated for compute capability 8.x is supported to run on compute capability 8.x or any higher revision (major or minor), including compute capability 9.0. Therefore although it is optional, **it is recommended that all applications should include PTX of the kernels to ensure forward-compatibility.** To read more about cubin and PTX compatibilities see Compilation with NVCC from the *CUDA C++ Programming Guide*.

When a CUDA application launches a kernel on a GPU, the CUDA Runtime determines the compute capability of the GPU in the system and uses this information to find the best matching cubin or PTX version of the kernel. If a cubin compatible with that GPU is present in the binary, the cubin is used as-is for execution. Otherwise, the CUDA Runtime first generates compatible cubin by JIT-compiling[1] the PTX and then the cubin is used for the execution. If neither compatible cubin nor PTX is available, kernel launch results in a failure.

Application binaries that include PTX version of kernels, should work as-is on the Hopper GPUs. In such cases, rebuilding the application is not required. However application binaries which do not include PTX (only include cubins), need to be rebuilt to run on the Hopper GPUs. To know more about building compatible applications read *Building Applications with Hopper Architecture Support*.

Application binaries that include PTX version of kernels with architecture conditional features using `sm_90a` or `compute_90a` in order to take full advantage of Hopper GPU architecture, are not forward or backward compatible.

[1] Just-in-time compilation.

# Chapter 3. Verifying Hopper Compatibility for Existing Applications

The first step towards making a CUDA application compatible with Hopper architecture is to check if the application binary already contains compatible GPU code (at least the PTX). The following sections explain how to accomplish this for an already built CUDA application.

## 3.1. Applications Built Using CUDA Toolkit 11.7 or Earlier

CUDA applications built using CUDA Toolkit versions 2.1 through 11.7 are compatible with Hopper GPUs as long as they are built to include PTX versions of their kernels. This can be tested by forcing the PTX to JIT-compile at application load time with following the steps:

- Download and install the latest driver from https://www.nvidia.com/drivers.
- Set the environment variable `CUDA_FORCE_PTX_JIT=1`.
- Launch the application.

With `CUDA_FORCE_PTX_JIT=1`, GPU binary code embedded in an application binary is ignored. Instead PTX code for each kernel is JIT-compiled to produce GPU binary code. An application fails to execute if it does not include PTX. This means the application is not Hopper architecture compatible and needs to be rebuilt for compatibility. On the other hand, if the application works properly with this environment variable set, then the application is Hopper compatible.

**Note:** Be sure to unset the `CUDA_FORCE_PTX_JIT` environment variable after testing is done.

## 3.2. Applications Built Using CUDA Toolkit 11.8

CUDA applications built using CUDA Toolkit 11.8 are compatible with Hopper architecture as long as they are built to include kernels in native cubin (compute capability 9.0) or PTX form or both.

# Chapter 4. Building Applications with Hopper Architecture Support

Depending on the version of the CUDA Toolkit used for building the application, it can be built to include PTX and/or native cubin for the Hopper architecture. Although it is enough to just include PTX, including native cubin also has the following advantages:

- It saves the end user the time it takes to JIT-compile kernels that are available only as PTX. All kernels which do not have native cubins are JIT-compiled from PTX, including kernels from all the libraries linked to the application, even if those kernels are never launched by the application[2]. Especially when using large libraries, this JIT compilation can take a significant amount of time. The CUDA driver caches the cubins generated as a result of the PTX JIT, so this is mostly a one-time cost for a user, but it is time best avoided whenever possible.
- PTX JIT-compiled kernels often cannot take advantage of architectural features of newer GPUs, meaning that native-compiled cubins may be faster or of greater accuracy.
- PTX code compiled to target architecture conditional features using `sm_90a` or `compute_90a` only runs on devices with compute capability 9.0 and is not backward or forward compatible.

## 4.1. Building Applications Using CUDA Toolkit 11.7 or Earlier

The `nvcc` compiler included with version 11.7 or earlier (11.0-11.7) of the CUDA Toolkit can generate cubins native to the NVIDIA Ampere GPU architectures (compute capability 8.x). When using CUDA Toolkit 11.7 or earlier, to ensure that `nvcc` will generate cubin files for all recent GPU architectures as well as a PTX version for forward compatibility with future GPU architectures, specify the appropriate `-gencode=` parameters on the `nvcc` command line as shown in the examples below.

**Windows**

```
nvcc.exe -ccbin "C:\vs2010\VC\bin"
  -Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
  -gencode=arch=compute_52,code=sm_52
  -gencode=arch=compute_60,code=sm_60
```

(continues on next page)

[2] Starting with CUDA toolkit 11.8, this default behavior can be changed with environment variable CUDA_MODULE_LOADING. See Environment Variables in the *CUDA C++ Programming Guide* for details.

(continued from previous page)

```
  -gencode=arch=compute_61,code=sm_61
  -gencode=arch=compute_70,code=sm_70
  -gencode=arch=compute_75,code=sm_75
  -gencode=arch=compute_80,code=sm_80
  -gencode=arch=compute_80,code=compute_80
  --compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

**Mac/Linux**

```
/usr/local/cuda/bin/nvcc
  -gencode=arch=compute_52,code=sm_52
  -gencode=arch=compute_60,code=sm_60
  -gencode=arch=compute_61,code=sm_61
  -gencode=arch=compute_70,code=sm_70
  -gencode=arch=compute_75,code=sm_75
  -gencode=arch=compute_80,code=sm_80
  -gencode=arch=compute_80,code=compute_80
  -O2 -o mykernel.o -c mykernel.cu
```

Alternatively, the simplified `nvcc` command-line option `-arch=sm_XX` can be used. It is a shorthand equivalent to the following more explicit `-gencode=` command-line options used above. `-arch=sm_XX` expands to the following:

```
-gencode=arch=compute_XX,code=sm_XX
-gencode=arch=compute_XX,code=compute_XX
```

However, while the `-arch=sm_XX` command-line option does result in inclusion of a PTX back-end target binary by default, it can only specify a single target cubin architecture at a time, and it is not possible to use multiple `-arch=` options on the same `nvcc` command line, which is why the examples above use `-gencode=` explicitly.

For CUDA toolkits prior to 11.0, one or more of the `-gencode` options need to be removed according to the architectures supported by the specific toolkit version (for example, CUDA toolkit 10.x supports architectures up to sm_72 and sm_75). The final `-gencode` to generate PTX also needs to be updated. For further information and examples see the documentation for the specific CUDA toolkit version.

**Note:** `compute_XX` refers to a PTX version and `sm_XX` refers to a cubin version. The `arch=` clause of the `-gencode=` command-line option to `nvcc` specifies the front-end compilation target and must always be a PTX version. The `code=` clause specifies the back-end compilation target and can either be cubin or PTX or both. **Only the back-end target version(s) specified by the code= clause will be retained in the resulting binary; at least one should be PTX to provide compatibility with future architectures.**

# 4.2. Building Applications Using CUDA Toolkit 11.8

With versions 11.8 of the CUDA Toolkit, `nvcc` can generate cubin native to the Hopper architecture (compute capability 9.0). When using CUDA Toolkit 11.8, to ensure that `nvcc` will generate cubin files for all recent GPU architectures as well as a PTX version for forward compatibility with future GPU architectures, specify the appropriate `-gencode=` parameters on the `nvcc` command line as shown in the examples below.

**Windows**

```
nvcc.exe -ccbin "C:\vs2010\VC\bin"
  -Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
  -gencode=arch=compute_52,code=sm_52
  -gencode=arch=compute_60,code=sm_60
  -gencode=arch=compute_61,code=sm_61
  -gencode=arch=compute_70,code=sm_70
  -gencode=arch=compute_75,code=sm_75
  -gencode=arch=compute_75,code=sm_75
  -gencode=arch=compute_90,code=sm_90
  -gencode=arch=compute_90,code=compute_90
  --compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

**Mac/Linux**

```
/usr/local/cuda/bin/nvcc
  -gencode=arch=compute_52,code=sm_52
  -gencode=arch=compute_60,code=sm_60
  -gencode=arch=compute_61,code=sm_61
  -gencode=arch=compute_70,code=sm_70
  -gencode=arch=compute_75,code=sm_75
  -gencode=arch=compute_80,code=sm_80
  -gencode=arch=compute_90,code=sm_90
  -gencode=arch=compute_90,code=compute_90
  -O2 -o mykernel.o -c mykernel.cu
```

**Note:** `compute_XX` refers to a PTX version and `sm_XX` refers to a cubin version. The `arch=` clause of the `-gencode=` command-line option to `nvcc` specifies the front-end compilation target and must always be a PTX version. The `code=` clause specifies the back-end compilation target and can either be cubin or PTX or both. **Only the back-end target version(s) specified by the code= clause will be retained in the resulting binary; at least one should be PTX to provide compatibility with future architectures.**

## 4.3. Independent Thread Scheduling Compatibility

NVIDIA GPUs since Volta architecture have Independent Thread Scheduling among threads in a warp. If the developer made assumptions about warp-synchronicity[3], this feature can alter the set of threads participating in the executed code compared to previous architectures. Please see Compute Capability 7.x in the *CUDA C++ Programming Guide* for details and corrective actions. To aid migration to the Hopper architecture, developers can opt-in to the Pascal scheduling model with the following combination of compiler options.

```
nvcc -gencode=arch=compute_60,code=sm_90 ...
```

[3] Warp-synchronous refers to an assumption that threads in the same warp are synchronized at every instruction and can, for example, communicate values without explicit synchronization.

# Chapter 5. Revision History

**Version 1.0**

- Initial public release.

# Chapter 6. Notices

## 6.1. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

## 6.2. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

## 6.3. Trademarks

NVIDIA and the NVIDIA logo are trademarks 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