NVIDIA Ampere GPU Architecture Compatibility Guide for CUDA Applications

Application Note

Table of Contents

Chapter 1. NVIDIA Ampere GPU Architecture Compatibility

1.1. About this Document

This application note, *NVIDIA Ampere GPU Architecture Compatibility Guide for CUDA Applications*, is intended to help developers ensure that their NVIDIA® CUDA® applications will run on the NVIDIA® Ampere 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 the NVIDIA Ampere GPU architecture.

1.2. Application Compatibility on the NVIDIA Ampere GPU 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 7.0 is supported to run on a GPU with compute capability 7.5, however a cubin generated for compute capability 7.5 is *not* supported to run on a GPU with compute capability 7.0, and a cubin generated with compute capability 7.x is *not* supported to run on a GPU with compute capability 8.x.

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 7.x is supported to run on compute capability 7.x or any higher revision (major or minor), including compute capability 8.x. 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 *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 NVIDIA Ampere architecture based 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 NVIDIA Ampere architecture based GPUs. To know more about building compatible applications read Building Applications with the NVIDIA Ampere GPU Architecture Support

1.3. Verifying Ampere Compatibility for Existing Applications

The first step towards making a CUDA application compatible with the NVIDIA Ampere GPU 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.

1.3.1. Applications Built Using CUDA Toolkit 10.2 or Earlier

CUDA applications built using CUDA Toolkit versions 2.1 through 10.2 are compatible with NVIDIA Ampere architecture based 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 compatible with the NVIDIA Ampere GPU architecture 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 compatible with the NVIDIA Ampere GPU architecture.

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

[1] Just-in-time compilation

1.3.2. Applications Built Using CUDA Toolkit 11.0

CUDA applications built using CUDA Toolkit 11.0 are compatible with the NVIDIA Ampere GPU architecture as long as they are built to include kernels in native cubin (compute capability 8.0) or PTX form or both.

1.4. Building Applications with the NVIDIA Ampere GPU 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 NVIDIA Ampere GPU 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. 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.

1.4.1. Building Applications Using CUDA Toolkit 10.x or Earlier

The `nvcc` compiler included with versions 10.x (10.0, 10.1 and 10.2) of the CUDA Toolkit can generate cubins native to the Volta and Turing architectures (compute capability 7.x). When using CUDA Toolkit 10.x, 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=compute_75
  --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_75,code=compute_75
  -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 10.0, one or more of the `-gencode` options will need to be removed according to the architectures supported by the specific toolkit version (for example, CUDA toolkit 9.x supports architectures up to _60 and _61). The final `-gencode` to generate PTX would also need to be update – 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.**

1.4.2. Building Applications Using CUDA Toolkit 11.0

With versions 11.0 of the CUDA Toolkit, `nvcc` can generate cubin native to the NVIDIA Ampere GPU architecture (compute capability 8.0). When using CUDA Toolkit 11.0, 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_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
```

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.**

1.4.3. 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[2], this feature can alter the set of threads participating in the executed code compared to previous architectures. Please see *Compute Capability 7.0* in the *Programming Guide* for details and corrective actions. To aid migration to the NVIDIA Ampere GPU architecture, developers can opt-in to the Pascal scheduling model with the following combination of compiler options.

```
nvcc -gencode=arch=compute_60,code=sm_80 ...
```

[2] 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.

Appendix A. Revision History

Version 1.0

- Initial public release.