VOLTA COMPATIBILITY GUIDE FOR CUDA APPLICATIONS

DA-08649-001_v11.0 | July 2020

Application Note

TABLE OF CONTENTS

Chapter 1. Volta Compatibility 1
1.1. About this Document 1
1.2. Application Compatibility on Volta 1
1.3. Verifying Volta Compatibility for Existing Applications 2
1.3.1. Applications Using CUDA Toolkit 8.0 or Earlier 2
1.3.2. Applications Using CUDA Toolkit 9.0 2
1.4. Building Applications with Volta Support 2
1.4.1. Applications Using CUDA Toolkit 8.0 or Earlier 3
1.4.2. Applications Using CUDA Toolkit 9.0 4
1.4.3. Independent Thread Scheduling Compatibility 5
Appendix A. Revision History 6

Chapter 1. VOLTA COMPATIBILITY

1.1. About this Document

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

1.2. Application Compatibility on Volta

The NVIDIA CUDA C++ compiler, **`nvcc`**, can be used to generate both architecture-specific *cubin* files and forward-compatible *PTX* versions of each kernel. Each cubin file targets a specific compute-capability version and is forward-compatible *only with GPU architectures of the same major version number*. For example, cubin files that target compute capability 3.0 are supported on all compute-capability 3.x (Kepler) devices but are *not* supported on compute-capability 5.x (Maxwell) or 6.x (Pascal) devices. For this reason, to ensure forward compatibility with GPU architectures introduced after the application has been released, it is recommended that all applications include PTX versions of their kernels.

CUDA Runtime applications containing both cubin and PTX code for a given architecture will automatically use the cubin by default, keeping the PTX path strictly for forward-compatibility purposes.

Applications that already include PTX versions of their kernels should work as-is on Volta-based GPUs. Applications that only support specific GPU architectures via cubin files, however, will need to be updated to provide Volta-compatible PTX or cubins.

1.3. Verifying Volta Compatibility for Existing Applications

The first step is to check that Volta-compatible device code (at least PTX) is compiled into the application. The following sections show how to accomplish this for applications built with different CUDA Toolkit versions.

1.3.1. Applications Using CUDA Toolkit 8.0 or Earlier

CUDA applications built using CUDA Toolkit versions 2.1 through 8.0 are compatible with Volta as long as they are built to include PTX versions of their kernels. To test that PTX JIT is working for your application, you can do the following:

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

When starting a CUDA application for the first time with the above environment flag, the CUDA driver will JIT-compile the PTX for each CUDA kernel that is used into native cubin code.

If you set the environment variable above and then launch your program and it works properly, then you have successfully verified Volta compatibility.

Be sure to unset the CUDA_FORCE_PTX_JIT environment variable when you are done testing.

1.3.2. Applications Using CUDA Toolkit 9.0

CUDA applications built using CUDA Toolkit 9.0 are compatible with Volta as long as they are built to include kernels in either Volta-native cubin format (see Building Applications with Volta Support) or PTX format (see Applications Using CUDA Toolkit 8.0 or Earlier) or both.

1.4. Building Applications with Volta Support

When a CUDA application launches a kernel, the CUDA Runtime determines the compute capability of each GPU in the system and uses this information to automatically find the best matching cubin or PTX version of the kernel that is available. If a cubin file supporting the architecture of the target GPU is available, it is used; otherwise, the CUDA Runtime will load the PTX and JIT-compile that PTX to the GPU's native cubin format before launching it. If neither is available, then the kernel launch will fail.

The method used to build your application with either native cubin or at least PTX support for Volta depend on the version of the CUDA Toolkit used.

The main advantages of providing native cubins are as follows:

- It saves the end user the time it takes to JIT-compile kernels that are available only as PTX. All kernels compiled into the application must have native binaries at load time or else they will be built just-in-time from PTX, including kernels from all 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 will cache the cubins generated as a result of the PTX JIT, so this is mostly a one-time cost for a given 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 code may be faster or of greater accuracy.

1.4.1. Applications Using CUDA Toolkit 8.0 or Earlier

The compilers included in CUDA Toolkit 8.0 or earlier generate cubin files native to earlier NVIDIA architectures such as Maxwell and Pascal, but they *cannot* generate cubin files native to the Volta architecture. To allow support for Volta and future architectures when using version 8.0 or earlier of the CUDA Toolkit, the compiler must generate a PTX version of each kernel.

Below are compiler settings that could be used to build **`mykernel.cu`** to run on Maxwell or Pascal devices natively and on Volta devices via PTX JIT.

Note that **`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 must be PTX to provide Volta compatibility**.

Windows

```
nvcc.exe -ccbin "C:\vs2010\VC\bin"
  -Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
  -gencode=arch=compute_50,code=sm_50
  -gencode=arch=compute_52,code=sm_52
  -gencode=arch=compute_60,code=sm_60
  -gencode=arch=compute_61,code=sm_61
  -gencode=arch=compute_61,code=compute_61
  --compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

Mac/Linux

```
/usr/local/cuda/bin/nvcc
  -gencode=arch=compute_50,code=sm_50
  -gencode=arch=compute_52,code=sm_52
  -gencode=arch=compute_60,code=sm_60
  -gencode=arch=compute_61,code=sm_61
  -gencode=arch=compute_61,code=compute_61
  -O2 -o mykernel.o -c mykernel.cu
```

Alternatively, you may be familiar with the simplified **nvcc** command-line option **-arch=sm_XX**, which 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 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.

1.4.2. Applications Using CUDA Toolkit 9.0

With version 9.0 of the CUDA Toolkit, **nvcc** can generate cubin files native to the Volta architecture (compute capability 7.0). When using CUDA Toolkit 9.0, 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_50,code=sm_50
  -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_70,code=compute_70
  --compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

Mac/Linux

```
/usr/local/cuda/bin/nvcc
  -gencode=arch=compute_50,code=sm_50
  -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_70,code=compute_70
  -O2 -o mykernel.o -c mykernel.cu
```

Note that **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.**

Also, note that CUDA 9.0 removes support for compute capability 2.x (Fermi) devices. Any compute_2x and sm_2x flags need to be removed from your compiler commands.

1.4.3. Independent Thread Scheduling Compatibility

The Volta architecture introduces Independent Thread Scheduling among threads in a warp. If the developer made assumptions about warp-synchronicity, [1] 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 *CUDA C++ Programming Guide* for details and corrective actions. To aid migration Volta developers can opt-in to the Pascal scheduling model with the following combination of compiler options.

```
nvcc -arch=compute_60 -code=sm_70 ...
```

[1] *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.

Version 1.1

- Use CUDA C++ instead of CUDA C/C++
- Updated references to the CUDA C++ Programming Guide and CUDA C++ Best Practices Guide.

Notice

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.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

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

© 2010-2020 NVIDIA Corporation. All rights reserved.