



# Pascal Compatibility Guide for CUDA Applications

Application Note

# Table of Contents

<b>Chapter 1. Pascal Compatibility</b> .....	<b>1</b>
1.1. About this Document.....	1
1.2. Application Compatibility on Pascal.....	1
1.3. Verifying Pascal Compatibility for Existing Applications.....	2
1.3.1. Applications Using CUDA Toolkit 7.5 or Earlier.....	2
1.3.2. Applications Using CUDA Toolkit 8.0.....	2
1.4. Building Applications with Pascal Support.....	2
1.4.1. Applications Using CUDA Toolkit 7.5 or Earlier.....	3
1.4.2. Applications Using CUDA Toolkit 8.0.....	4
<b>Appendix A. Revision History</b> .....	<b>6</b>

---

# Chapter 1. Pascal Compatibility

## 1.1. About this Document

This application note, *Pascal Compatibility Guide for CUDA Applications*, is intended to help developers ensure that their NVIDIA® CUDA® applications will run on GPUs based on the NVIDIA® Pascal 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 Pascal.

## 1.2. Application Compatibility on Pascal

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.



**Note:** 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 Pascal-based GPUs. Applications that only support specific GPU architectures via cubin files, however, will need to be updated to provide Pascal-compatible PTX or cubins.

## 1.3. Verifying Pascal Compatibility for Existing Applications

The first step is to check that Pascal-compatible device code (at least PTX) is compiled in to 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 7.5 or Earlier

CUDA applications built using CUDA Toolkit versions 2.1 through 7.5 are compatible with Pascal 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 Pascal compatibility.



**Note:** Be sure to unset the `CUDA_FORCE_PTX_JIT` environment variable when you are done testing.

### 1.3.2. Applications Using CUDA Toolkit 8.0

CUDA applications built using CUDA Toolkit 8.0 are compatible with Pascal as long as they are built to include kernels in either Pascal-native cubin format (see [Building Applications with Pascal Support](#)) or PTX format (see [Applications Using CUDA Toolkit 7.5 or Earlier](#)) or both.

## 1.4. Building Applications with Pascal 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 Pascal 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 7.5 or Earlier

The compilers included in CUDA Toolkit 7.5 or earlier generate cubin files native to earlier NVIDIA architectures such as Kepler and Maxwell, but they *cannot* generate cubin files native to the Pascal architecture. To allow support for Pascal and future architectures when using version 7.5 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 Kepler or Maxwell devices natively and on Pascal 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 Pascal compatibility.**

#### Windows

```
nvcc.exe -ccbin "C:\vs2010\VC\bin"
-Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
-gencode=arch=compute_30,code=sm_30
-gencode=arch=compute_35,code=sm_35
-gencode=arch=compute_50,code=sm_50
-gencode=arch=compute_52,code=sm_52
-gencode=arch=compute_52,code=compute_52
--compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

#### Mac/Linux

```
/usr/local/cuda/bin/nvcc
-gencode=arch=compute_30,code=sm_30
-gencode=arch=compute_35,code=sm_35
```

```
-gencode=arch=compute_50,code=sm_50
-gencode=arch=compute_52,code=sm_52
-gencode=arch=compute_52,code=compute_52
-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 8.0

With version 8.0 of the CUDA Toolkit, `nvcc` can generate cubin files native to the Pascal architectures (compute capability 6.0 and 6.1). When using CUDA Toolkit 8.0, 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_30,code=sm_30
-gencode=arch=compute_35,code=sm_35
-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_30,code=sm_30
-gencode=arch=compute_35,code=sm_35
-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
```

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.**

---

# Appendix A. Revision History

## Version 1.0

- ▶ Initial public release.

## Version 1.1

- ▶ Use CUDA C++ instead of CUDA C/C++



## 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.

## VESA DisplayPort

DisplayPort and DisplayPort Compliance Logo, DisplayPort Compliance Logo for Dual-mode Sources, and DisplayPort Compliance Logo for Active Cables are trademarks owned by the Video Electronics Standards Association in the United States and other countries.

## HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

## OpenCL

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

## 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

© -2020 NVIDIA Corporation. All rights reserved.