KEPLER COMPATIBILITY GUIDE FOR CUDA APPLICATIONS

DA-06287-001_v1.0 | April 2012

Application Note

DOCUMENT CHANGE HISTORY

DA-06287-001_v1.0

Version	Date	Authors	Description of Change
1.0	April 6, 2012	CW	Initial public release.

TABLE OF CONTENTS

Chapter 1. KEPLER COMPATIBILITY

1.1 ABOUT THIS DOCUMENT

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

1.2 APPLICATION COMPATIBILITY ON KEPLER

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 CUDA architectures of the same major version number*. For example, **cubin** files that target compute capability 2.0 are supported on all compute-capability 2.x (Fermi) devices but are *not* supported on compute-capability 3.0 (Kepler) devices. For this reason, to ensure forward compatibility with CUDA architectures introduced after the application has been released, it is recommended that all applications support launching PTX versions of their kernels.[1]

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

[1] 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.

1.3 VERIFYING KEPLER COMPATIBILITY FOR EXISTING APPLICATIONS

The first step is to check that Kepler-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 4.1 or Earlier

CUDA applications built using CUDA Toolkit versions 2.1 through 4.1 are compatible with Kepler 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`
- Create an empty temporary directory on your system.
- Set the environment variable `CUDA_CACHE_PATH` to be the path to this empty directory.
- 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. The generated **cubin** for the target GPU architecture is cached on disk by the CUDA driver.

If you set the environment variables above and then launch your program and it works properly, and if the directory you specified with the CUDA_CACHE_PATH environment variable is now populated with cache files, then you have successfully verified Kepler compatibility. Note that it is not necessary to inspect the contents of the cache files themselves; just check that the previously empty cache directory is now non-empty.

Be sure to unset these two environment variables when you are done testing if you do not normally use them. The temporary cache directory you created is safe to delete.

1.3.2 Applications Using CUDA Toolkit 4.2

CUDA applications built using CUDA Toolkit 4.2 are compatible with Kepler as long as they are built to include kernels in either Kepler-native **cubin** format (see Section 1.4) or PTX format (see Section 1.3.1 above) or both.

1.4 BUILDING APPLICATIONS WITH KEPLER SUPPORT

The methods used to build your application with support for Kepler depend on the version of the CUDA Toolkit used and on the choice of the CUDA Runtime API or CUDA Driver API.

Note: The CUDA Runtime API is characterized by the use of functions named with the `cuda*()` prefix and by launching kernels using the triple-angle-bracket <<<>>> notation. The CUDA driver API functions use the `cu*()` prefix, including for kernel launch.

1.4.1 CUDA Runtime API Applications

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 main advantages of providing native **cubin**s are as follows:

- It saves the end user the time it takes to PTX JIT a kernel that has been compiled as PTX. (However, since the CUDA driver will cache the **cubin** generated as a result of the PTX JIT, this is mostly a one-time cost for a given user.)
- 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.

1.4.1.1 Applications Using CUDA Toolkit 4.1 or Earlier

The compilers included in CUDA Toolkit 4.1 or earlier generate **cubin** files native to earlier NVIDIA architectures such as Fermi, but they cannot generate **cubin** files native to the Kepler architecture. To allow support for Kepler and future architectures when using version 4.1 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 Fermi and earlier devices natively and on Kepler devices via PTX JIT. In these examples, the lines shown in blue provide compatibility with earlier architectures, and the lines shown in red provide a PTX path for compatibility with Kepler and later architectures.

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 Kepler compatibility.**

Windows:

```
nvcc.exe -ccbin "C:\vs2008\VC\bin"
  -Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
  -gencode=arch=compute_10,code=sm_10
  -gencode=arch=compute_20,code=sm_20
  -gencode=arch=compute_20,code=compute_20
  --compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

Mac/Linux:

```
/usr/local/cuda/bin/nvcc
  -gencode=arch=compute_10,code=sm_10
  -gencode=arch=compute_20,code=sm_20
  -gencode=arch=compute_20,code=compute_20
  -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.

1.4.1.2 Applications Using CUDA Toolkit 4.2

Beginning with version 4.2 of the CUDA Toolkit, `nvcc` can generate **cubin** files native to the Kepler architecture (compute capability 3.0). When using CUDA Toolkit 4.2, to ensure that `nvcc` will generate **cubin** files for all released 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.

In these examples, the lines shown in blue provide compatibility with earlier architectures, the lines shown in green provide native **cubin**s for Kepler, and the lines in red provide a PTX path for compatibility with future architectures.

Windows:

```
nvcc.exe -ccbin "C:\vs2008\VC\bin"
  -Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
  -gencode=arch=compute_10,code=sm_10
  -gencode=arch=compute_20,code=sm_20
  -gencode=arch=compute_30,code=sm_30
  -gencode=arch=compute_30,code=compute_30
  --compile -o "Release\mykernel.cu.obj" "mykernel.cu"
```

Mac/Linux:

```
/usr/local/cuda/bin/nvcc
  -gencode=arch=compute_10,code=sm_10
  -gencode=arch=compute_20,code=sm_20
  -gencode=arch=compute_30,code=sm_30
  -gencode=arch=compute_30,code=compute_30
  -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.**

1.4.2 CUDA Driver API Applications

Applications that use the CUDA Driver API load their own kernels explicitly. Therefore, the kernel-loading portions of such applications must include a path capable of loading PTX when native **cubin**s for the target GPU(s) are not available.

- Compile CUDA kernel files to PTX, even if also compiling native **cubin** files for existing architectures. If multiple compilation target types/versions are to be used, `nvcc` must be called separately for each generated output file of either type, and the type and version must be specified explicitly at compile time. (An advanced technique is to use a "fat binary" (fatbin) file, which contains both **cubin** and PTX formats. This technique is outside the scope of this document.)

  A common pattern in many applications is to include **cubin**s for all supported existing architectures plus PTX of the highest-available version for forward compatibility to future architectures.

  The example below demonstrates compilation of `compute_20` PTX, which will work on devices of compute capability 2.x and 3.0, but *not* on devices of compute capability 1.x. Presumably an application using this example as-is would also include **cubin**s for compute capability 1.x and/or 2.x as well.

  Windows:

  ```
  nvcc.exe -ccbin "C:\vs2008\VC\bin"
    -Xcompiler "/EHsc /W3 /nologo /O2 /Zi /MT"
    -ptx -arch=compute_20
    -o "mykernel.compute_20.ptx" "mykernel.cu"
  ```

  Mac/Linux:

  ```
  /usr/local/cuda/bin/nvcc
    -ptx -arch=compute_20
    -O2 -o mykernel.compute_20.ptx "mykernel.cu"
  ```

- At runtime, your application will need to explicitly check the compute capability of the current GPU with the CUDA Driver API function in order to select the best-available **cubin** or PTX to load. The `deviceQueryDrv` code sample from the NVIDIA GPU Computing SDK includes a detailed example of the use of this function.

  ```
  cuDeviceComputeCapability(&major, &minor, dev)
  ```

- Refer to the "PTX Just-in-Time Compilation" (`ptxjit`) code sample GPU Computing SDK, available at the URL below, which demonstrates how to use the CUDA Driver API to launch PTX kernels.

  http://developer.nvidia.com/cuda-cc-sdk-code-samples

A more complex example can be found in the `matrixMulDrv` code sample from the GPU Computing SDK, which follows a pattern similar to the following:

```
CUmodule cuModule;
CUfunction cuFunction = 0;
string ptx_source;

// Helper function load PTX source to a string
findModulePath ("matrixMul_kernel.ptx",
                module_path, argv, ptx_source));

// We specify PTX JIT compilation with parameters
const unsigned int jitNumOptions = 3;
CUjit_option *jitOptions = new CUjit_option[jitNumOptions];
void **jitOptVals = new void*[jitNumOptions];

// set up size of compilation log buffer
jitOptions[0] = CU_JIT_INFO_LOG_BUFFER_SIZE_BYTES;
int jitLogBufferSize = 1024;
jitOptVals[0] = (void *)(size_t)jitLogBufferSize;

// set up pointer to the compilation log buffer
jitOptions[1] = CU_JIT_INFO_LOG_BUFFER;
char *jitLogBuffer = new char[jitLogBufferSize];
jitOptVals[1] = jitLogBuffer;

// set up maximum # of registers to be used
jitOptions[2] = CU_JIT_MAX_REGISTERS;
int jitRegCount = 32;
jitOptVals[2] = (void *)(size_t)jitRegCount;

// Loading a module will force a PTX to be JIT
status = cuModuleLoadDataEx(&cuModule, ptx_source.c_str(),
                            jitNumOptions, jitOptions,
                            (void **)jitOptVals);

printf("> PTX JIT log:\n%s\n", jitLogBuffer);
```

APPENDIX A. Revision History

A.1 VERSION 1.0

- Initial public release.

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

© 2012 NVIDIA Corporation. All rights reserved.