Chapter 3
Introduction to CUDA C

Chapter Objectives

- You will write your first lines of code in CUDA C.
- You will learn the difference between code written for the host and code written for a device.
- You will learn how to run device code from the host.
- You will learn about the ways device memory can be used on CUDA-capable devices.
- Your will lean hot to query your system information on its CUDA-capable devices.

3.2.1 Hello, World

```
#include "../common/book.h"
int main( void ) {

    printf( "Hello, World!¥n" );

    return 0;

}
```

A Simple Kernel Call

```
#include <iostream>

__global__ void kernel( void ) {
}

int main( void ) {
    kernel<<<1,1>>>();
    printf( "Hello, World!\n" );
    return 0;
}
```

3.2.2 A Kernel Call

- Source: simple_kernel.cu
- An empty function named kernel() qualified with __global__
  - The __global__ qualifier indicates to the compiler that a function should be complied to run on a device but not on the host.
- A call to the empty function, embellished with <<1,1>>
  - The numbers in <<1,1>> are not arguments to the device code but are parameters that will influence how the runtime system launch our device code.

```
#include <iostream>
#include "book.h"

__global__ void add( int a, int b, int *c ) {
    *c = a + b;
}

int main( void ) {
    int c;
    int *dev_c;
    HANDLE_ERROR( cudaMalloc( (void**)&dev_c, sizeof(int) ) );

    add<<<1,1>>>( 2, 7, dev_c );

    HANDLE_ERROR( cudaMemcpy( &c,
                              dev_c,
                              sizeof(int),
                              cudaMemcpyDeviceToHost ) );
    printf( "2 + 7 = %d\n", c );
    cudaFree( dev_c );

    return 0;
}
```

3.2.3 Passing Parameters

- Source: simple_kernel_param.cu
- Introduction of two concepts
  - We can pass parameters to a kernel as we would with any C function.
  - We need to allocate memory to do anything useful on a device, such as return values to the host.
    - cudaMalloc() allocates memory on a device
    - The first argument is a pointer to the pointer you want to hold the address of the newly allocated memory
    - The second parameter is the size of the allocation.
    - HANDLE_ERROR() is a utility macro, detects that the call has returned an error, prints the associated error message, and exits the application.

3.2.3 Passing Parameters (2)

- The restrictions on the usage of device pointers
  - You can pass pointers allocated with cudaMalloc() to functions that execute on the device.
  - You can use pointers allocated with cudaMalloc() to read or write memory from code that executes on the device.
  - You can pass pointers allocated with cudaMalloc() to functions that execute on the host.
  - You cannot use pointers allocated with cudaMalloc() to read or write memory from code that executes on the host.

3.2.3 Passing Parameters (3)

- cudaMemcpy()
  - accessing device memory by calling cudaMemcpy() from host code.
  - The parameter, cudaMemcpyDeviceToHost indicates that the source pointer is a device pointer and the destination pointer is a host pointer.
  - cudaMemcpyHostToDevice indicates the opposite situation.
  - cudaMemcpyDeviceToDevice is also available.
  - cudaMemcpyHostToHost? -> use standard C's memcpy()

3.3 Querying Devices

- A way of knowing how much memory and that types of capabilities the device had, and so on.
- cudaGetDeviceCount()
  - How many devices in the system were built on the CUDA architecture

1. int count;
2. HANDLE_ERROR(cudaGetDeviceCount(&count));

3.3 Querying Devices (2)

- The CUDA runtime returns us properties in a structure of type cudaDeviceProp.
  - See Table 3.1 CUDA Device Properties in the textbook.
- How to query each device and report the properties of each?
  - See the textbook.
- Source: enum_gpu.cu

3.4 using Device Properties

- Suppose that we are writing an application that depends on having double-precision floating-point support.
  - CUDA 1.3 or higher supports double-precision application.
  - We need to find at least one device of compute capability 1.3 or higher.
  - First we fill a cudaDeviceProp structure with the required conditions, and then call cudaChooseDevice() to know ID of a device which satisfies the required conditions.
  - Source: set_gpu.cu