CUDA Programming

Jing Liu
Dimitar Lukarski
Jarmo Rantakokko

March 11, 2014

1 General instructions for Windows users

If you are using MAC iOS or Linux (Unix) system, please skip this section, and move on to the next one.

1.1 The Windows environment in our lab

The system we used in the lab is Window 7 64bits, with pre-installed CUDA 5.5 and Visual Studio 2010. To achieve our goal of this lab, you don't need to install anything or change Users/ System Path. Openmp 3.0 is also supported by the Microsoft Visual Studio 2010.

1.2 How to create a new project

Create a simple Nvidia CUDA runtime project:

1. Open Microsoft Visual Studio 2010. Click on the **Start** menu on the left bottom corner, go to **All Programs**, under folder **Microsoft Visual Studio 2010** you will find the application called **"Microsoft Visual Studio 2010"**.

2. Create a new project. On the **File** menu, click **New** and then click **Project**, you will see the **New Project dialog** box. Or in the **Start Page**, click on **New Project**. In the **New Project dialog** box, under the **Installed Templates**, click on **NVIDIA**, and then click **CUDA 5.5**. Fill in the **Name** and/or **Solution name** (ex. hello_world) in the bottom of the box. Click **OK**. Figure 1 is a screenshot about creating a **CUDA 5.5 Runtime** Project. Those RED rectangles emphasize where you need to click.

1.3 Compile, Debug and Release

Now you have a **CUDA 5.5 Runtime** Project with an example, who's name is "kernel.cu". The simplest way to get our own code work is to copy our code, and replace the content of "kernel.cu" with our code (simply do copy and paste).

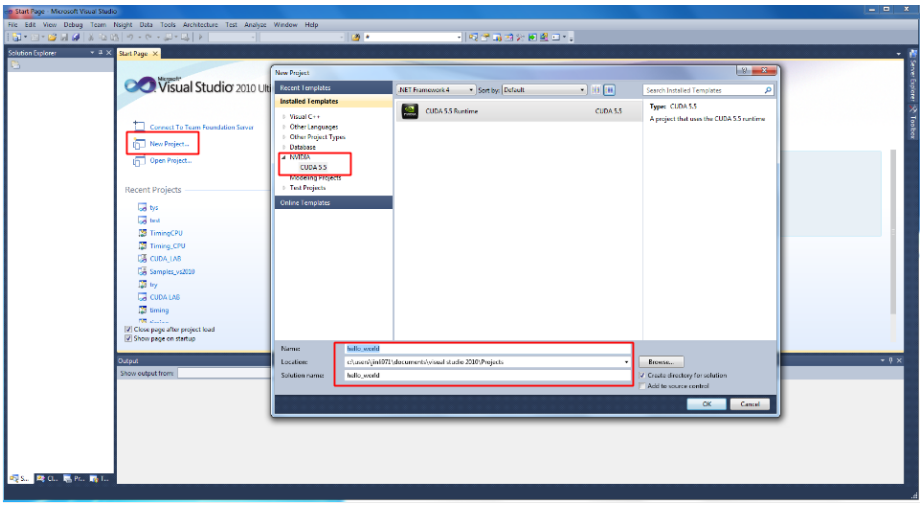

Figure 1: How to create a **CUDA 5.5 Runtime** Project

Compile . Compiling can be done with debug/release or separately. In the **Solution Explorer** box, right click on the project name, you can find **Compile** in the menu.

Debug and release are the 2 different ways to run programs. Change the mode in the tool bar as in the RED box in Figure 1.3

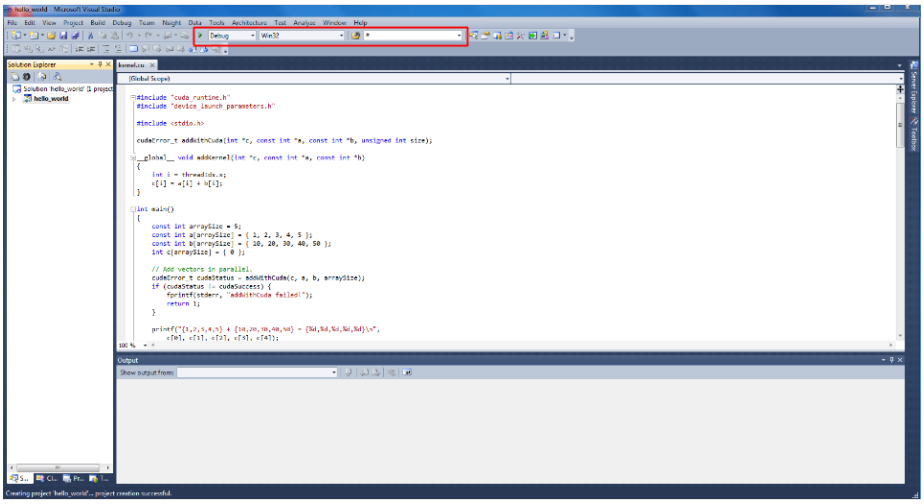

Figure 2: Compile, debug and release.

2 General instructions for Linux/Mac users

Compile and run your code with commands:

```
$ /usr/local/cuda/bin/nvcc hello_world.cu -o hello_world
$ ./hello_world

$ /usr/local/cuda/bin/nvcc timing.cu -o timing
$ ./timing
```

3 Your Tasks

Copy the CUDA files (*.cu) from the course homepage.

For Windows users – create a new VS2010 solution and replace the content of the code with your program, click *Debug/Release* or use shortcut $ctrl + F5$ to run it.

For Linux users – Just compile and run the files.

3.1 Analysis the code

Read and analysis the code (hello_world.cu) - what does the code do? Will the code work if one changes the thread-block configuration and why?

3.2 Timing and 1D blocks

- Read and run the code, understand what the coding is doing and what are the optimal values for the blocksize?
- Implement serial and OpenMP function for computing the $x = x + y$. What is the corresponding performance on the CPU? Compare it for the GPU function (again 500 calls).
- What is the bandwidth of the $PCI - E$ (copying data to and from the GPU)
- If you take into account the $PCI - E$ transfers, how many vector updates a program need to perform in order to be faster when moving the data to the GPU? What is the limiting factor (if any)?

4 2D blocks

Modify the program timing.cu to use two dimension indexing in the vector kernel (i.e. for blockIdx, blockDim and threadIdx use .x and .y). Check the results with your CPU version.

4.1 Question

If we have a CPU with 50 GFlop/s peak performance and a GPU with 500 GFlop/s peak performance - what will be the performance difference on these two devices of our vector function?

5 Report

The laboration is a part of the examination. To pass you need to either attend the lab and work actively on the tasks or write a short informal report summarizing your results and answers together with your source code for the different tasks.