LABORATORY MANUAL

CE4011/CZ4011 Parallel Computing

No. 3 : CUDA Programming and Optimizations

SESSION 2016/2017
SEMESTER 1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

1. OBJECTIVE

You will do several things in this lab. First, you will become familiar with the GPU computing environment and be able to compile and run CUDA programs. Next, you will use the CUDA API to implement a basic kernel for the all pairs shortest path problem. Lastly, you will optimize your code using coalesced memory accesses and shared memory.

2. LABORATORY

SWLab 1A/1B, previously MM2 Lab (N4-01a-02).

3. EQUIPMENT

PC running Linux operating system and with NVIDIA GPUs, C and CUDA 5.0.

4. INTRODUCTION

The CUDA environment. In this lab, we will use the CUDA C 5.0 programming platform. CUDA 5.0 comes with a number of example programs to help you start practicing CUDA programming. They can be found in the "cuda5_samples" directory in your home directory.

Among the sample programs are NVIDIA SDK utilities to determine the specifications of the GPU hardware.

- *Bandwidth Test:* This is a simple program to measure the memcopy bandwidth of the GPU and the memcpy bandwidth across PCI-e. This test application is capable of measuring device to device copy bandwidth, host to device copy bandwidth for pageable and page-locked memory, and device to host copy bandwidth for pageable and page-locked memory.
- *Device Query:* This sample program enumerates the properties of the CUDA devices present in the system.

In the first part of this lab, you will run those two utilities to get familiar with the GPU hardware in your machine.

Writing a kernel. In the second part of the lab, you will practice using CUDA by writing a simple kernel. The kernel will solve the same all pairs shortest path problem you looked at in the previous labs. As a brief reminder, it works as follows.

We use Floyd's algorithm for solving the all pairs shortest path problem. In Floyd's algorithm, the adjacency matrix is transformed into a matrix containing the lengths of the shortest paths between all pairs of vertices. A sequential version of Floyd's algorithm may be described by the following pseudocode and has a time complexity of $O(n^3)$:

```
read number of vertices n and n × n adjacency matrix a;
for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            a[i, j] = min(a[i, j], a[i, k] + a[k, j]);
```

When the algorithm terminates, the matrix *a* contains the lengths of the shortest paths between all pairs of vertices.

Write Floyd's algorithm as a CUDA program, using one thread to compute each distance value a[i,j] Use one kernel invocation to compute the two innermost loops. Invoke the kernel n times to compute the outermost loop.

Optimizing the kernel. In the last part of the lab, you will optimize the kernel from part 2. Specifically, you should perform coalesced memory accesses and use the shared memory to speed up your kernel.

5. EXPERIMENT

In your experiments, perform the following tasks.

5.1 Check the CUDA environment

From your home directory, go to the "cuda5_samples/1_Utilities" directory.

5.2 Check the GPU hardware

Step 1: Go to the "bandwidthTest" directory and execute the *bandwidthTest* program from the console without any arguments. If you cannot find the executable file *bandwidthTest*, simply run "make" to compile the project and generate the executable. Study the memory performance of the GPU.

Question: What are your major observations on memory bandwidths from the program output?

Step 2: Go to the "deviceQuery" directory and execute the *deviceQuery* program from the console without any arguments. Study the hardware parameters of the GPU.

Question: What is the Compute Capability of the GPU? What are the differences between GPUs with different Compute Capabilities?

5.3 Write the basic kernel

You can start a project from scratch. Here, we use a quicker way of developing a CUDA program which is based on a template project. The template project is located in "~/0_Simple/template".

Using the template project, you need to 1) add/remove code from the existing template files 2) add/remove .h/.cpp/.cu files and edit the Makefile accordingly.

Use "make" to compile your program.

5.4 Benchmark the basic kernel

Use the Nvidia Visual Profiler to measure the execution time of your program. To run the profiler, select menu item "File / New Session" and select the executable to profile and set its arguments. Then select menu item "Run / Collect metrics and events" and select the metrics you want to profile.

Perform sensitivity studies to understand the performance of your kernel. Compare the GPU kernel with the *sequential* CPU version of Floyd's algorithm by measuring the execution time for different numbers of vertices (10, 100, …) and different thread configurations (i.e. the size and number of thread blocks). For each of the following tests, vary one parameter while keeping the others fixed.

- For large numbers of vertices, the data files may be generated randomly and the solution of the CUDA program checked against the solution of the sequential program from your previous lab experiments. Show the results as graphs and provide a full analysis.
- For different size thread blocks and number of thread blocks, show the execution times as graphs and provide a full analysis.

5.5 Write optimized kernels

There are many ways to optimize the basic Floyd kernel to use coalesced memory accesses and shared memory. To find the most effective method, you may need to perform multiple rounds of code profiling, analysis, code rewriting and testing. The basic Floyd kernel is bandwidth constrained, and your code will typically run faster if you can decrease the number of global memory transactions. Pay attention to this metric when you profile your program.

Demonstrate the performance impact of individual optimizations by writing the following three versions of CUDA code:

- **Coalesced accesses only**: the basic Floyd kernel optimized with coalesced memory accesses *only*.
- **SM only**: the basic kernel optimized with shared memory optimizations *only*.
- **Full optimization**: a version with both coalesced accesses and shared memory optimizations.

5.6 Benchmark the optimized kernels

Using the Nvidia Visual Profiler, perform sensitivity studies to understand the performance of the optimized kernels. Compare the following:

- **Basic**: the basic kernel you developed in part 5.3.
- **Coalesced only, SM only, full optimization**: the three optimized kernels from part 5.5.

Perform the comparison by 1) measuring the execution time and 2) profiling relevant metrics (see "8. Metrics Reference" in [2]). Perform the same types of tests as in part 5.4, namely by varying the number of vertices and thread configuration.

6. REPORT

Your report only needs to contain the results from Experiments 5.4 and 5.6. Record the results in your team logbook (you may also use a loose-leaf folder with printed pages). Include the source code listings of your programs, examples of data files and their output, the timing results of the programs shown as graphs, and a full desription and analysis of the work, including a discussion of the performance results. For Experiment 5.6, be sure to also describe the metrics you looked at in your profiling, and explain the types of optimizations you performed and the performance improvements you observed.

The report is due by 5pm Friday, Nov. 4. Submit your report to the Software Projects Lab (N4-B1b-11) in the pigeonhole for your group.

7. REFERENCES

[1] CUDA C Programming Guide 5.0, NVIDIA, 2012. [2] CUDA Visual Profiler. http://docs.nvidia.com/cuda/profiler-users-guide/index.html#visual-profiler-views