ITCS 4145/5145 Fall 2013
Assignment 5 CUDA Programming Assignment

B. Wilkinson, Nov 21, 2013 (minor change to measuring first launch)

The purpose of this assignment is to become familiar with writing, compiling and executing CUDA programs. We will use **cci-grid08**, which has a 2496-core NVIDIA K20 GPU.[1] In Part 1, you are asked to compile an existing CUDA program that adds two vectors using a given make file. In Part 2, you are asked to write a matrix multiplication program in CUDA and test on the GPU with various grid and block sizes. In Part 3, you are asked to write a sorting program using the RankSort algorithm. Part 4 is for extra credit – a CUDA program that implements perform Odd-Even Transposition Sort. In all your programs, a sequential version is to be executed that runs on the CPU alone, for comparison purposes.

For this assignment, you may use your own computer if you have a NVIDIA GPU card and you install NVIDIA CUDA Toolkit, instead of using coit-grid08. However you may not get as great a speed-up, and finally you may wish to test on the K20.

Preliminaries (2%)

Login to **cci-gridgw.uncc.edu** and ssh to **cci-grid08**. You will be able to see your home directory on the cluster. Create a directory called **Assign5** and cd into this directory.

GPU limitations. Display the details of the GPU(s) installed by issuing the command:

```
deviceQuery
```

Keep the output as you may need it. In particular, note the maximum number of threads in a block and maximum sizes of blocks and grid. Also invoke the bandwidth test by issuing the command:

```
bandwidthTest
```

Note the maximum host to device, device to host, and device to device bandwidths.

Part 1 Compiling and executing vector addition CUDA program (33%)

In this part, you will compile and execute a CUDA program to perform vector addition. This program is given below:

```
// VectorAdd.cu
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#define N 10                            // size of vectors
```

[1] Two other GPU server are available, **coit-grid06.uncc.edu** and **cci-grid07**, both of which have 448-core NVIDIA C2050 GPUs.

```
#define B 1                                    // blocks in the grid
#define T 10                                    // threads in a block

__global__ void add (int *a,int *b, int *c) {
        int tid = blockIdx.x *  blockDim.x + threadIdx.x;
        if(tid < N) {
                c[tid] = a[tid]+b[tid];
        }
}

int main(void) {
            int a[N],b[N],c[N];
            int *dev_a, *dev_b, *dev_c;

            cudaMalloc((void**)&dev_a,N * sizeof(int));
            cudaMalloc((void**)&dev_b,N * sizeof(int));
            cudaMalloc((void**)&dev_c,N * sizeof(int));

            for (int i=0;i<N;i++) {
                        a[i] = i;
                        b[i] = i*1;
            }

            cudaMemcpy(dev_a, a , N*sizeof(int),cudaMemcpyHostToDevice);
            cudaMemcpy(dev_b, b , N*sizeof(int),cudaMemcpyHostToDevice);
            cudaMemcpy(dev_c, c , N*sizeof(int),cudaMemcpyHostToDevice);

            add<<<B,T>>>(dev_a,dev_b,dev_c);

            cudaMemcpy(c,dev_c,N*sizeof(int),cudaMemcpyDeviceToHost);

            for (int i=0;i<N;i++) {
                        printf("%d+%d=%d\n",a[i],b[i],c[i]);
            }

            cudaFree(dev_a);
            cudaFree(dev_b);
            cudaFree(dev_c);

            return 0;
}
```

Create a file called **VectorAdd.cu** containing the vector addition program above. Next, create a file called **Makefile** and copy the following into it:

```
NVCC = /usr/local/cuda/bin/nvcc
CUDAPATH = /usr/local/cuda
NVCCFLAGS = -I$(CUDAPATH)/include
```

```
LFLAGS = -L$(CUDAPATH)/lib64 -lcuda -lcudart -lm

VectorAdd: VectorAdd.cu
	$(NVCC) $(NVCCFLAGS) -o VectorAdd VectorAdd.cu $(LFLAGS)
```

Be careful to have a tab here. (Very important)

To compile the program, type **make VectorAdd** (or **make** as there is only one build command). Execute the program by typing the name of the executable (to include the current directory **./**), i.e. **./VectorAdd**. Confirm the results are correct.

What to submit from this part

Your submission document should include the following:

1) A screenshot of compiling and executing **VectorAdd**

Part 2 Matrix multiplication (35%)

Modify the CUDA program in Part 1 to perform matrix multiplication with *N* x *N* matrices. Use a square 2-D grid and square 2D block structure. Incorporate the following features to enable experimentation to be done on speed-up factor:

(a) Different sizes for the matrices – Use dynamically allocated memory and add keyboard input statements to be to specify N.

(b) Add host code to compute the matrix multiplication on the host only.

(c) Add code to verify that both CPU and GPU versions of vector addition produce the same correct results

(d) Different CUDA grid/block structures – Add keyboard statements to input different values for:

- Numbers of threads in a block (T)
- Number of blocks in a grid (B)

Include checks for invalid input. Ensure that GPUs limitations are met from the data given in deviceQuery (Preliminaries).

(e) Timing -- Add statements to time the execution of the code using CUDA events, both for the host-only (CPU) computation and with the device (GPU) computation, and display results. Compute and display the speed-up factor.

Arrange that the code returns to keyboard input after each computation with entered keyboard input rather than re-starting the code and having kernel code re-launch. Include print statements to show all input values.

During code development, it is recommended that the code is recompiled and tested after adding each of (a), (b), (c), (d) and (e).

Modify the make file according to compile the code. Execute your code and experiment with four different combinations of T, B, and N and collect timing results including speed-up factor. What is the effect of the first kernel launch with one combination of T, B, and N, i.e. record the execution time when the first time the kernel is executed and subsequent execution times the same kernel is executed with the same parameters without exiting the main program? Discuss the results.

What to submit from this part

Your submission document should include the following:

1) A properly commented listing of your matrix multiplication program with the features (a) – (e) specified incorporated.
2) Screenshots of executing your matrix multiplication program with four different combinations of T, B, and N, displaying values and the speedup factor in each case.
3) An experiment showing the effect of the first kernel launch with one combination of T, B, and N.
4) Discussion of the results

Part 3 Sorting (30%)

Write a CUDA program that implements Ranksort. Incorporate the following features to enable experimentation to be done on speed-up factor:

(a) Generate numbers to be sorted using a random number generator. Add keyboard input statements to be to specify the numbers of numbers.

(b) Add host code to compute sorting on the host only.

(c) Add code to verify that both CPU and GPU versions of vector addition produce the same correct results

(d) Different CUDA grid/block structures – Add keyboard statements to input different values for:

- Numbers of threads in a block (T)
- Number of blocks in a grid (B)

Include checks for invalid input. Ensure that GPUs limitations are met from the data given in deviceQuery (Preliminaries).

(e) Timing -- Add statements to time the execution of the code using CUDA events, both for the host-only (CPU) computation and with the device (GPU) computation, and display results. Compute and display the speed-up factor.

Execute your code and experiment with four different combinations of T, B, and N and collect timing results including speed-up factor.

What to submit from this part

Your submission document should include the following:

1) A properly commented listing of your sorting program with the features (a) – (e) specified incorporated.
2) Screenshots of executing your sorting program with four different combinations of T, B, and N, displaying values and the speedup factor in each case.
3) Discussion of the results

Part 4 Extra credit (undergraduate and graduate students, +30%).

Write a CUDA program to perform Odd-Even Transposition Sort. Incorporate features listed in Part 3.

What to submit from this part

As in Part 3

Grading

Every task and subtask specified will be allocated a score so make sure you clearly identify each part you did. The quality of conclusions and discussions will factor into the grading.
What to submit

Assignment Submission

Produce a document that show that you successfully followed the instructions and performs all tasks by taking screen shots and include these screen shots in the document. Give sufficient screen shots to demonstrate each task and sub-task has been fully completed. Provide insightful conclusions. Submit by the due date as described on the course home page. Include all code, not as screen shots but complete properly documented code listing.