Assignment 7

CUDA Programming Assignment

B. Wilkinson, April 17a, 2016

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. 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 program to implement Ranksort. Part 4 asks you to add shared memory. In all your programs, a sequential version is to be executed that runs on the CPU alone, for comparison purposes. *The sample programs listed in this assignment are not in the course VM or posted. You have to create them yourself. Do not copy and paste from pdf unless you are sure you are removing any unwanted characters.*

One other GPU server is currently available, **cci-grid07**, which has a 448-core NVIDIA C2050 GPU. Use that server if **cci-grid08** is not functioning. You may also use your own computer instead of **coit-grid08** if you have a NVIDIA GPU card and you install the NVIDIA CUDA Toolkit. However you may not get as great a speed-up. Whatever system you use, clearly state it in your report, and the type of host processor and GPU.

You can use PuTTy/WinSCP to connect to the cluster rather than the VM on a Windows platform and this may be easier in this assignment as all the programming is done on the cluster. See the course link "Additional Information" > "Accessing remote servers from a Windows platform (PuTTY, WinSCP, nano)" for more information on that.

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 **Assign7** 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 (28%)

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
#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*2;
         }
         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. 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) Screen shot(s) of executing **deviceQuery** and **bandwidthTest**
2) Screenshot of compiling and executing **VectorAdd**

Part 2 Matrix multiplication (30%)

Modify the CUDA program in Part 1 to perform matrix multiplication with two *N* x *N* integer matrices, a[N][N] and b[N][N]. Initialize the matrices with randomly generated numbers on the host using the code:

```
int row, col;
srand(2);
for(row=0; row < N; row++)        // load arrays with some numbers
   for(col=0; col < N; col++) {
      a[row * N + col] = rand() % 10;
      b[row * N + col] = rand() % 10;
   }
```

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 matrix multiplication produce the same *and* correct results.
(d) Different CUDA 2-D 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)

Assume a square 2-D grid and square 2-D block structures. 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? That is, 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. Generate the numbers to sort on the host in an array A[N] using the random number generator:

```
srand(3);                     //initialize random number generator
for (i=0; i < N; i++)         //load array with numbers
     A[i] = (int)rand();
```

where there are N numbers. Use a 1-dimensional block and grid structure for the kernel. Incorporate the following features to enable experimentation to be done on speed-up factor:

(a) Add keyboard input statements to specify the numbers of numbers (N).
(b) Add code to sort the numbers using Ranksort on the host only (and make sure the results are correct).
(c) Add code to verify that both CPU and GPU versions of Ranksort produce the same results.
(d) 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) Add statements to time the execution of the code using CUDA events, both for the host-only (CPU) computation and for 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 Part 3

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 Using Shared Memory (10%)

Modify the sorting code in Part 3 to also sort the numbers with Ranksort on the GPU using shared memory to hold the numbers. Include all the features in the code listed in Part 3. The code should give the execution time with shared memory and without shared memory.

What to submit from Part 4

Your submission document should include the following:

1) A properly commented listing of your sorting program.
2) Screenshots of executing your sorting program with four different combinations of T, B, and N, displaying values and the speedup factors in each case, with shared memory and without shared memory.
3) Discussion of the results.

Assignment Report Preparation and Grading

Produce a report that shows that you successfully followed the instructions and performs all tasks by taking screenshots and include these screenshots in the document.

Every task and subtask specified will be allocated a score so make sure you clearly identify each part/task you did. Make sure you include everything that is specified in the “What to include ...” section at the end of each task/part. **Include all code, not as screen shots of the listings but complete properly documented code listing.** The programs are often too long to see in a single screenshot and

also not easy to read if small. Using multiple screenshots is not option for code. Copy/paste the code into the Word document or whatever you are using to create the report.
You will lose 10 points if you submit code as screenshots.

Assignment Submission

Convert your report into a ***single pdf*** document and submit the pdf file on Moodle by the due date as described on the course home page. It is extremely important you submit only a pdf file. *It is possible to loose points if you submit in any other format (i.e. Word, OpenOffice, zip, ...).* ***You will lose 10 points if you submit in any other format (e.g. Word, OpenOffice, ...). zip with multiple files is especially irritating.***