ITCS 4145/5145 Spring 2012
Assignment 5 CUDA Programming Assignment

April 6, 2012

Preliminaries

For this assignment, you will gain experience in using two platforms:

- Windows XP computers in Woodward lab room 335, which have mid-level NVIDIA cards installed,

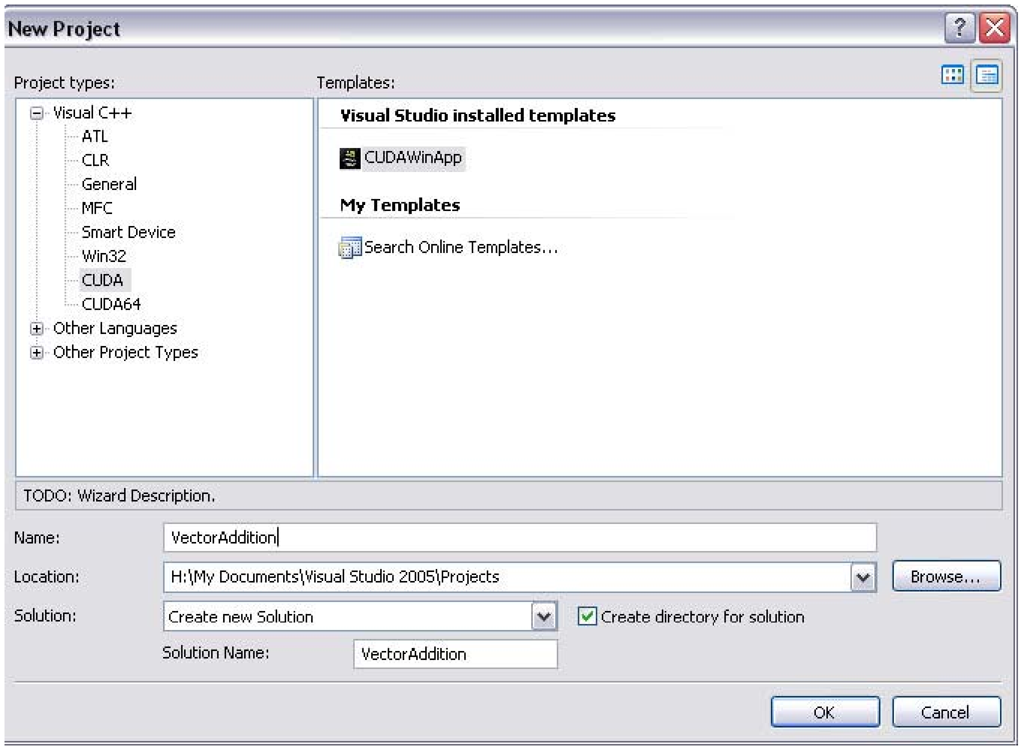
and

- coit-grid07.uncc.edu, which has a full 448-core NVIDIA Tesla GPU (Fermi)

Part 1 Windows XP computer

Task 1 *Compiling and executing CUDA program using Visual Studio*

Go to lab #335 and log onto a Windows machine. To compile and execute CUDA programs on a Windows XP system, we will use Microsoft Visual Studio 2005[1]. Go to Start, Program Files and then select Microsoft Visual Studio 2005. Click on File -> New -> Project. In project types, select CUDA. Name the program as VectorAddition.



TODO: Wizard Description.

Name: VectorAddition
Location: H:\My Documents\Visual Studio 2005\Projects
Solution: Create new Solution
✓ Create directory for solution
Solution Name: VectorAddition

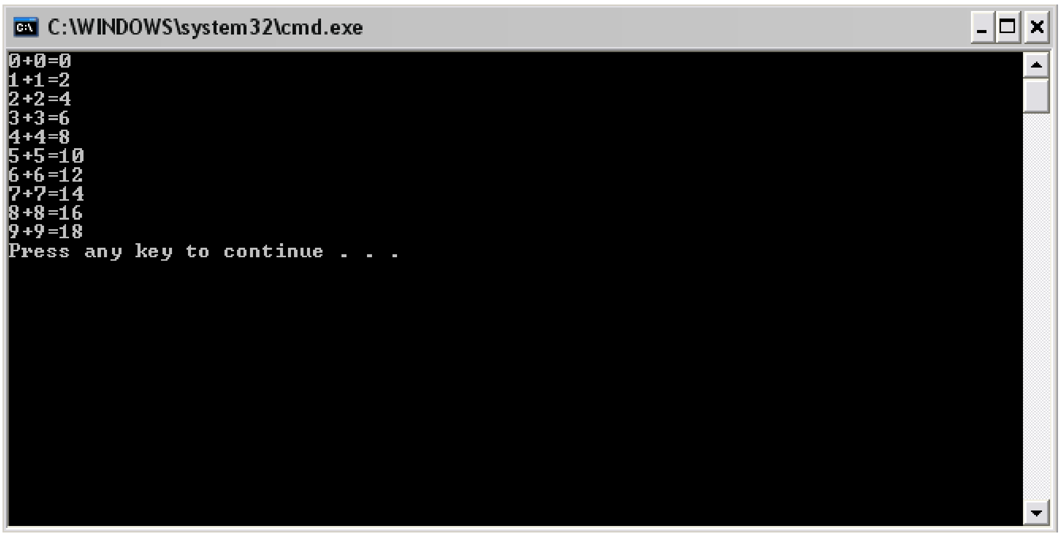
[1] Note, 2005, not 2008 or 2010, as these versions in Woodward 338 have issues with CUDA.

Click OK, then click Next. Select Console Application from the Application Type and Precompiled header and precompiled header from Additional options.

Click on Finish. Open Sample.cu and replace it with the code below:

```
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#define N 10         // size of array
#define T 10     // threads per block
#define B 1       // blocks per grid

__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;
}
```

Rename Sample.cu as VectorAddition.cu. To build the solution click on Build. Then press Ctrl+F5 to run the solution:



```
0+0=0
1+1=2
2+2=4
3+3=6
4+4=8
5+5=10
6+6=12
7+7=14
8+8=16
9+9=18
Press any key to continue . . .
```

Explain code and results.

Task 2 *Experiments with VectorAddition code*

(a) Timing -- Add statements to time the execution of the code and re-run VectorAddition. Describe results.

(b) Different CUDA grid/block structures -- Experiment with a larger vector (N = 4096) with large numbers of threads in a block (up to 512) and more than one block in a CUDA grid (one dimensional structures, at least six different structures). Describe results.

Part 2 Using coit-grid07.uncc.edu

Task 1 *Compiling and executing CUDA program on coit-grid07*

In this task, we will repeat running the vector addition code but on coit-grid07.uncc.edu. Log onto coit-grid07.uncc.edu. Your username and password is the same as for the other coit-grid systems. You will be able to see your home directory for the cluster.

Create a directory called VectorAddition directory in your home directory and cd into it. Create a file called VectorAddition.cu containing the previous program.

Next, create a file called Makefile and copy the following into it:

```
PROJECT_NAME = VectorAddition
# NVCC is path to nvcc. Here it is assumed that /usr/local/cuda is on one's PATH.

NVCC = /usr/local/cuda/bin/nvcc

CUDAPATH = /usr/local/cuda
BUILD_DIR = build
NVCCFLAGS = -I$(CUDAPATH)/include
LFLAGS = -L$(CUDAPATH)/lib64 -lcuda -lcudart -lm

# if we run just `make`, will build and then clean.
all: build

# in order to build, we make the build dir, compile kernels, compile host code, and link.

build: build_dir
	$(NVCC) $(NVCCFLAGS) $(LFLAGS) -o $(BUILD_DIR)/$(PROJECT_NAME) *.cu

build_dir:
	mkdir -p $(BUILD_DIR)

# run `make run` to run project file. provided for simplicity.
run:
	./$(BUILD_DIR)/$(PROJECT_NAME)
```

Be careful to have tabs where needed.

Compile the vector addition program by typing “make”.

Execute the compiled program by typing “make run”.

Task 2 *Experiments with VectorAddition code*

(i) Timing -- Add statements to time the execution of the code and re-run VectorAddition. Describe results and compare with using the Windows lab machine.

(ii) Different CUDA grid/block structures – As with task 1, experiment with a larger vector (N = 4096) with large numbers of threads in a block (up to 512) and more than one block in a CUDA grid (one dimensional structures, at least six different structures). Describe results and compare with using the Windows lab machine.

Part 3 Writing your own CUDA programs

This part is to be done on coit-grid07 system.

Task 1 Monte Carlo π Program

In this section, you will write your own simple program, compile it, and submit it. Write a CUDA program to compute π by a Monte Carlo method. The method is described in Appendix A at the end of this document. You will need to include the statement #include <math.h> at the top of your program for math routines and compile with –lm option (which is included in the make file).

Make sure that each thread uses a different random sequence.

Graduate students are to explore the use of random number generators provided in the NVIDIA toolkit (SDK). Undergraduate students can simply use the C rand() function although a poor random number generator.

Experiment with program parameters including different numbers of threads in a block (N) and more than one block in a CUDA grid (one dimensional structures, at least six different structures). Describe results.

Task 2 For Graduate Students Only (10%, extra credit for undergraduates)

Find another problem that can be solved by a Monte Carlo approach and implement in CUDA. Describe your results and include timing and different problem sizes.

Assignment Submission

Produce a document that provides the following details:

- A full explanation of your code
- Code listing
- Sample output
- Insightful conclusions.

Submit to Moodle by the due date (see home page). Combine everything into one PDF file.
All students must work individually.

Appendix A Computing π by a Monte Carlo Method

The basis of Monte Carlo methods is the use of random selections in calculations. $\pi/4$ (and hence π) can be computed by a Monte Carlo method as follows: A circle is formed within a square as shown in Figure 1. The circle has unit radius so that the square has sides 2×2. The ratio of the area of the circle to the square is given by

$$\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$

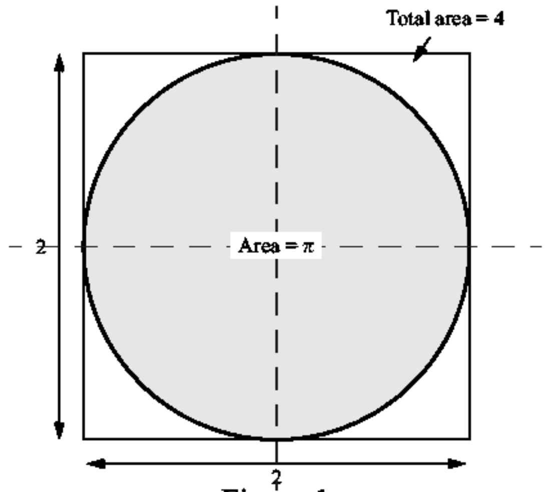


Figure 1

Points within the square are chosen randomly and a score is kept of how many points happen to lie within the circle. The fraction of points within the circle will be $\pi/4$, given a sufficient number of randomly selected samples.

Only one quadrant of the construction need be used. One quadrant of Figure 1 is shown in Figure 2, A random pair of numbers, (x_r, y_r) is generated, each between 0 and 1, and then counted if $y_r^2 + x_r^2 \leq 1$; that is, $y_r \leq \sqrt{1 - x_r^2}$.

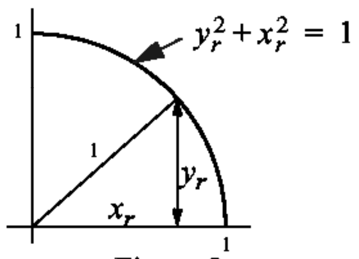


Figure 2