ITCS443 Parallel and Distributed Systems

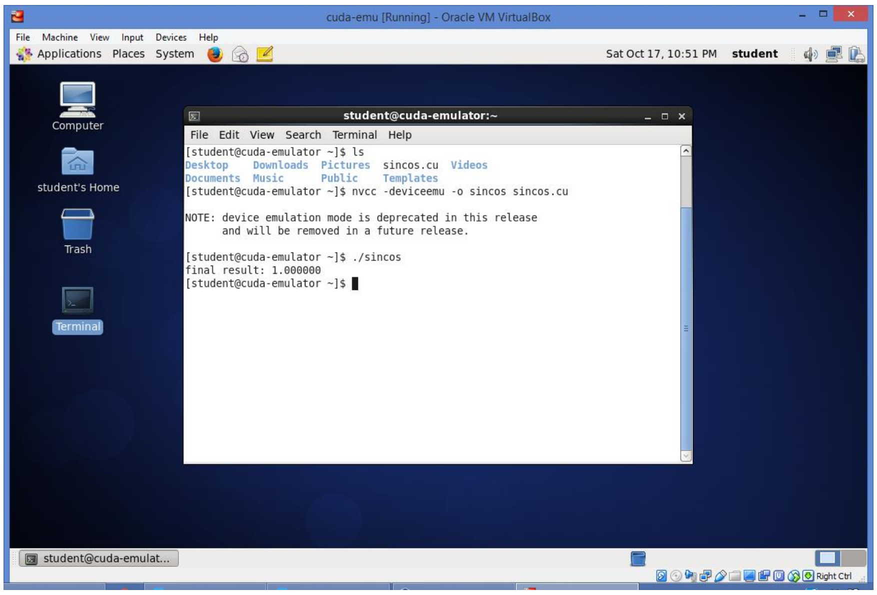
Lab 9 Introduction to CUDA

Prerequisite:

Option A) Use NVCC with -deviceemu (simulation mode) on our server at **10.34.73.11**

Option B) Use the cuda-emu virtual machine (VM). The NVCC with -deviceemu is pre-installed in the VM. Download the VM image at

https://drive.google.com/file/d/1Pa41YB50AEU3iPaViP1v24frtzmHQSYF/view?usp=sharing


cuda-emu [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Applications Places System
Sat Oct 17, 10:51 PM student
Computer
student's Home
Trash
Terminal
student@cuda-emulator:~
File Edit View Search Terminal Help
student@cuda-emulat...
Right Ctrl

```
[student@cuda-emulator ~]$ ls
Desktop    Downloads  Pictures  sincos.cu  Videos
Documents  Music      Public    Templates
[student@cuda-emulator ~]$ nvcc -deviceemu -o sincos sincos.cu

NOTE: device emulation mode is deprecated in this release
      and will be removed in a future release.

[student@cuda-emulator ~]$ ./sincos
final result: 1.000000
[student@cuda-emulator ~]$ 
```

Then, use VirtualBox or VMware Player to import and run the cuda-emu VM. The username and password is *student*.

Option C) For only students who have a real Nvidia card. Install Visual Studio, Nvidia driver, and CUDA toolkit. https://www.youtube.com/watch?v=cuCWbztXk4Y

1) Create and run the vecAdd.cu program.
1.1) Use text editor to create the following `vecAdd.cu` program.

```
#include <stdio.h>

#define N 256

__global__ void vecAdd(int *A, int *B, int *C) {
        int i = threadIdx.x;
        C[i] = A[i] + B[i];
}

int main (int argc, char *argv[] ) {

    int i;
    int size = N *sizeof( int);
    int  a[N], b[N], c[N], *devA, *devB, *devC;

    for (i=0; i < N; i++) {
        a[i] = 1; b[i] = 2;
    }

    cudaMalloc( (void**)&devA, size);
    cudaMalloc( (void**)&devB, size);
    cudaMalloc( (void**)&devC, size);

    cudaMemcpy( devA, a, size, cudaMemcpyHostToDevice);
    cudaMemcpy( devB, b, size, cudaMemcpyHostToDevice);

    vecAdd<<<1, N>>>(devA, devB, devC);

    cudaMemcpy( c, devC, size, cudaMemcpyDeviceToHost);
    cudaFree( devA);
    cudaFree( devB);
    cudaFree( devC);

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

}
```

1.2) Compile the program using the following command:

```
nvcc -o vecAdd vecAdd.cu
```

or if using emulation, remember to use -deviceemu option for every nvcc command.

```
nvcc -deviceemu -o vecAdd vecAdd.cu
```

1.3) Run the program

```
./vecAdd
```

2) Given an array A of 256 integers, write a CUDA program named `vecInc.cu` to increase the value of each element in the array A by one. For example, if A = {1, 3, …, 6, 0}, then after calling a CUDA kernel, array A becomes {2, 4, …, 7, 1}.

3) Modify `vecInc.cu` from Question 2 into `vecInc2.cu` to work with an array A of any size (e.g. A[1000]) but using only 256 threads. (Do not use 1000 threads)

4) Submit the results (in the answer sheet) to mycourses website.