ELEC 374: GPU Machine Problem #1

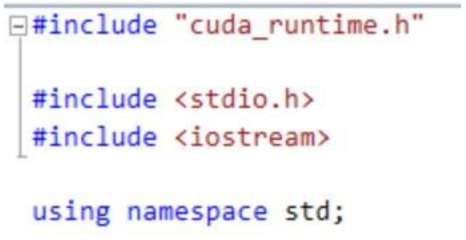
Device Query

Due Date: 2022-03-27

Isabel Frolick – 20155540

To identify the types of CUDA devices on the GPU servers of the environment on which this program was run, a query of the device specifications was ran. This query aided in improving the comprehension of the NVIDIA GPU. To gain a thorough understanding of the device, the query included specs concerning: the clock rate, the number of streaming multiprocessors (SM), number of cores, warp size, amount of global memory, amount of constant memory, amount of shared memory per block, number of registers available per block, maximum number of threads per block, maximum size of each dimension of a block, and the maximum size of each dimension of a grid.

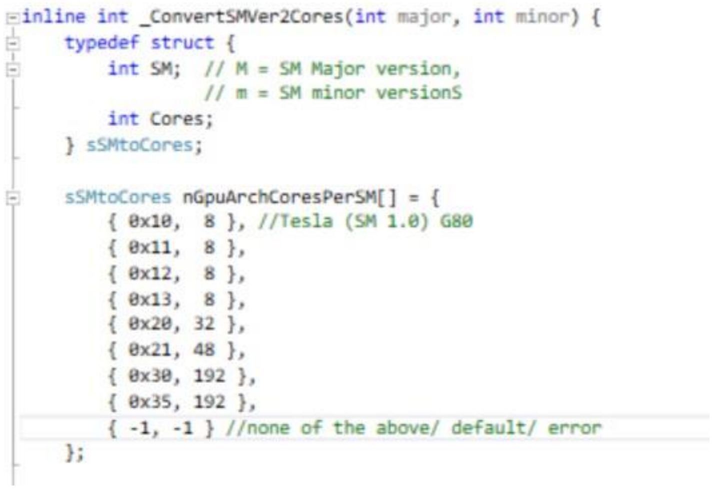
To invoke the necessary functionality in CUDA, one must include the main CUDA library, "cuda_runtime.h" which calls numerous other CUDA libraries. The other libraries included in this problem, <stdio.h> and <iostream> allowed utilization of C++ syntax, as CUDA is running from a Visual Studio 2015 C++ program. The std namespace was invoked as well.



```
#include "cuda_runtime.h"

#include <stdio.h>
#include <iostream>

using namespace std;
```

Figure 1: Machine Problem #1 libraries used

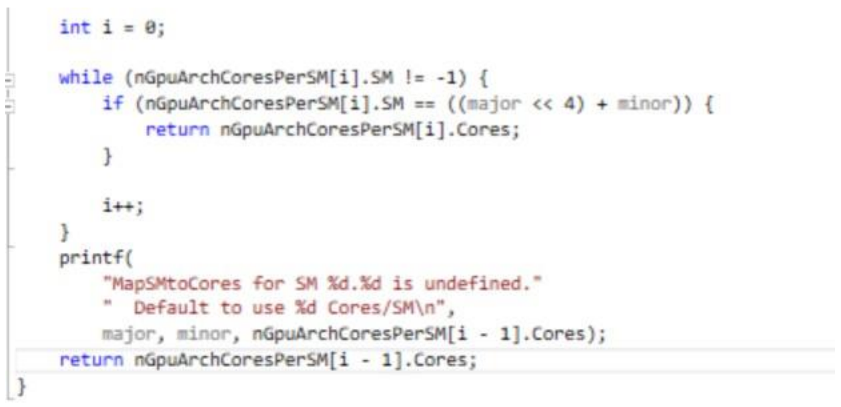
To obtain CUDA device information, a CUDA utility function 'ConvertSMVer2Cores' was invoked. This function utilized hard coded streaming multiprocessor (SM) information to return the number of CUDA cores per multiprocessor for a specified compute capability version. The function 'ConvertSMVer2Cores' used a type defining structure to define the SM as either the major or minor version and the nGpuArchCoresPerSM[] hard coded array to define the device.



```
inline int _ConvertSMVer2Cores(int major, int minor) {
    typedef struct {
        int SM;  // M = SM Major version,
                 // m = SM minor versionS
        int Cores;
    } sSMtoCores;

    sSMtoCores nGpuArchCoresPerSM[] = {
        { 0x10,  8 }, //Tesla (SM 1.0) G80
        { 0x11,  8 },
        { 0x12,  8 },
        { 0x13,  8 },
        { 0x20, 32 },
        { 0x21, 48 },
        { 0x30, 192 },
        { 0x35, 192 },
        { -1, -1 } //none of the above/ default/ error
    };
```

Figure 2: ConvertSMVer2Cores function implementation.

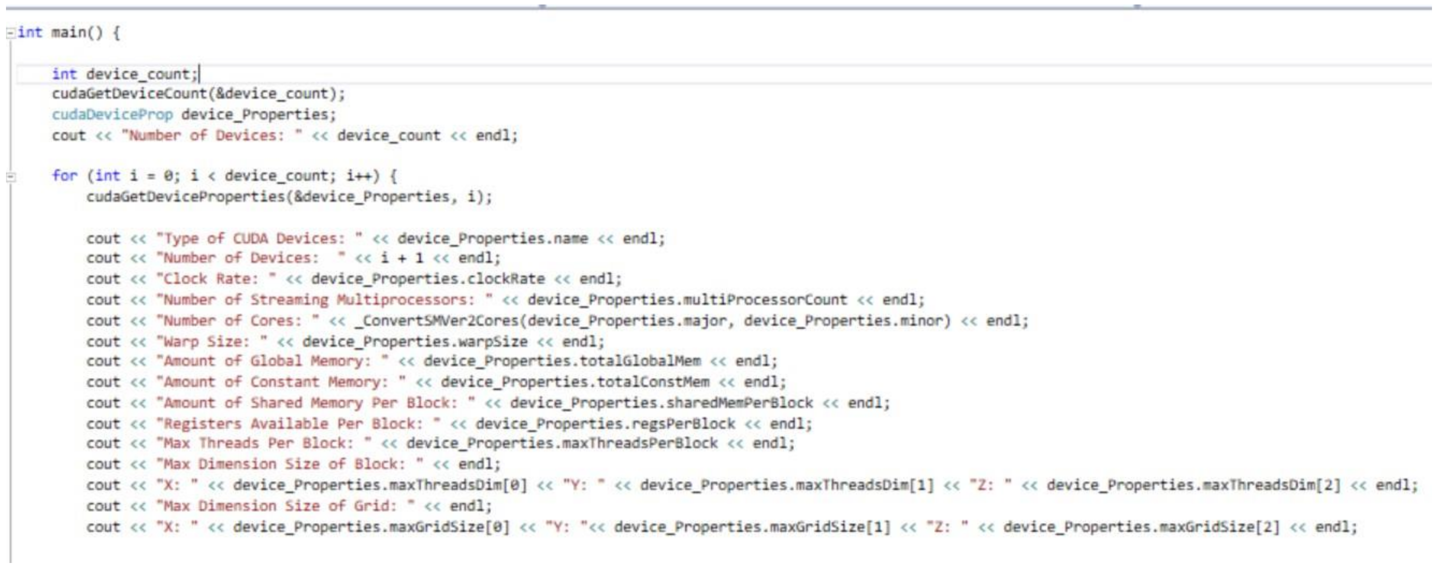
The list of hardcoded versions was iterated until the specifications aligned, at which point the number of cores per multiprocessor was returned. If this never occurs, an error message is returned.



```
    int i = 0;

    while (nGpuArchCoresPerSM[i].SM != -1) {
        if (nGpuArchCoresPerSM[i].SM == ((major << 4) + minor)) {
            return nGpuArchCoresPerSM[i].Cores;
        }

        i++;
    }
    printf(
        "MapSMtoCores for SM %d.%d is undefined."
        "  Default to use %d Cores/SM\n",
        major, minor, nGpuArchCoresPerSM[i - 1].Cores);
    return nGpuArchCoresPerSM[i - 1].Cores;
}
```

Figure 3: Continuation of ConvertSMVer2Cores function implementation.

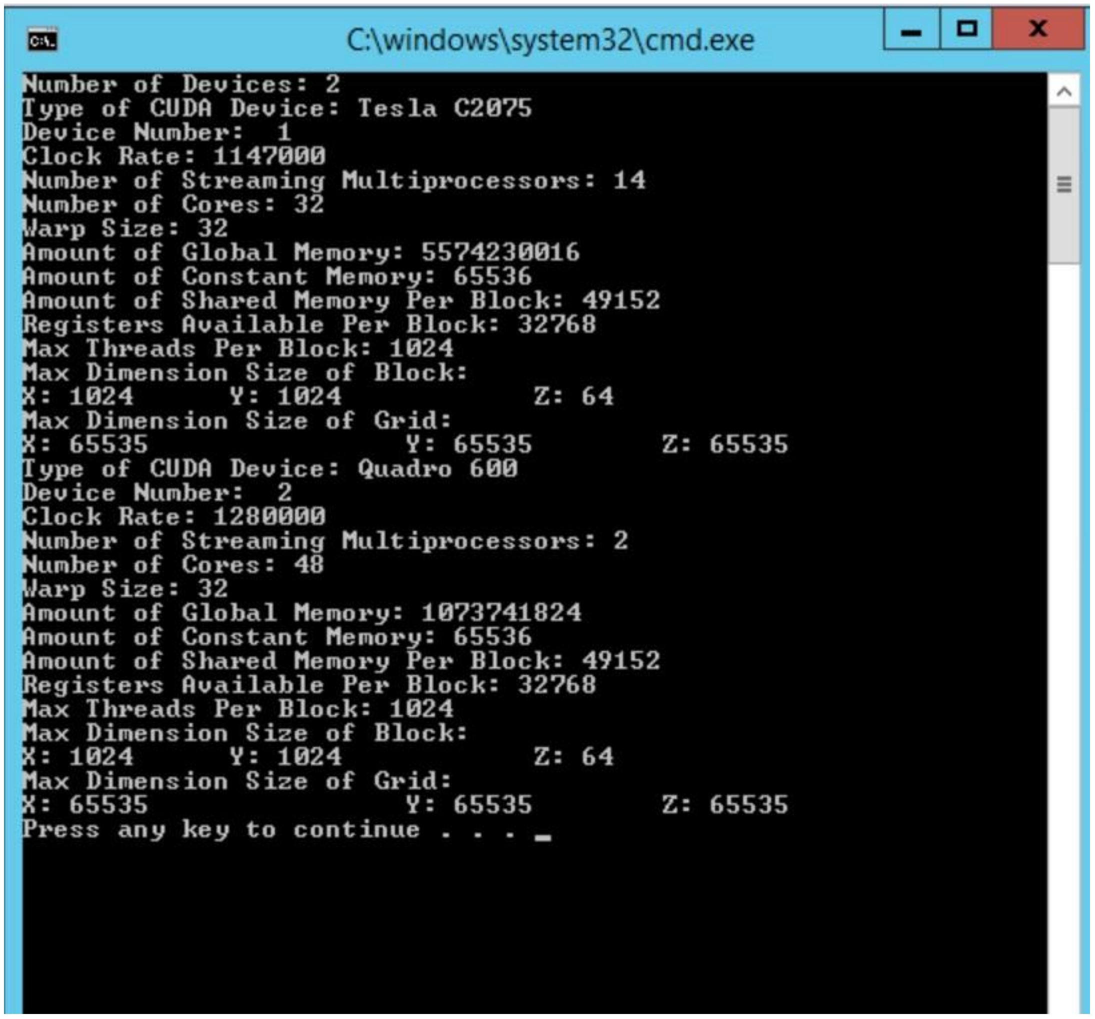
In the main function, the count of CUDA devices in the system was found using the CUDA function ‘cudaGetDeviceCount’. The device count was used to iterate the avaliable CUDA devices and, using the ‘*cudaDeviceProp’* structure, which returns the properties of the respective device, the necessary information pertaining to the device was found.



```
int main() {

    int device_count;
    cudaGetDeviceCount(&device_count);
    cudaDeviceProp device_Properties;
    cout << "Number of Devices: " << device_count << endl;

    for (int i = 0; i < device_count; i++) {
        cudaGetDeviceProperties(&device_Properties, i);

        cout << "Type of CUDA Devices: " << device_Properties.name << endl;
        cout << "Number of Devices:  " << i + 1 << endl;
        cout << "Clock Rate: " << device_Properties.clockRate << endl;
        cout << "Number of Streaming Multiprocessors: " << device_Properties.multiProcessorCount << endl;
        cout << "Number of Cores: " << _ConvertSMVer2Cores(device_Properties.major, device_Properties.minor) << endl;
        cout << "Warp Size: " << device_Properties.warpSize << endl;
        cout << "Amount of Global Memory: " << device_Properties.totalGlobalMem << endl;
        cout << "Amount of Constant Memory: " << device_Properties.totalConstMem << endl;
        cout << "Amount of Shared Memory Per Block: " << device_Properties.sharedMemPerBlock << endl;
        cout << "Registers Available Per Block: " << device_Properties.regsPerBlock << endl;
        cout << "Max Threads Per Block: " << device_Properties.maxThreadsPerBlock << endl;
        cout << "Max Dimension Size of Block: " << endl;
        cout << "X: " << device_Properties.maxThreadsDim[0] << "Y: " << device_Properties.maxThreadsDim[1] << "Z: " << device_Properties.maxThreadsDim[2] << endl;
        cout << "Max Dimension Size of Grid: " << endl;
        cout << "X: " << device_Properties.maxGridSize[0] << "Y: "<< device_Properties.maxGridSize[1] << "Z: " << device_Properties.maxGridSize[2] << endl;
```

Figure 4: Machine Problem #1 main function implementation for query.

The output of the CUDA device query can be seen below, in Figure 5.



```
Number of Devices: 2
Type of CUDA Device: Tesla C2075
Device Number:  1
Clock Rate: 1147000
Number of Streaming Multiprocessors: 14
Number of Cores: 32
Warp Size: 32
Amount of Global Memory: 5574230016
Amount of Constant Memory: 65536
Amount of Shared Memory Per Block: 49152
Registers Available Per Block: 32768
Max Threads Per Block: 1024
Max Dimension Size of Block:
X: 1024      Y: 1024            Z: 64
Max Dimension Size of Grid:
X: 65535                Y: 65535        Z: 65535
Type of CUDA Device: Quadro 600
Device Number:  2
Clock Rate: 1280000
Number of Streaming Multiprocessors: 2
Number of Cores: 48
Warp Size: 32
Amount of Global Memory: 1073741824
Amount of Constant Memory: 65536
Amount of Shared Memory Per Block: 49152
Registers Available Per Block: 32768
Max Threads Per Block: 1024
Max Dimension Size of Block:
X: 1024      Y: 1024            Z: 64
Max Dimension Size of Grid:
X: 65535                Y: 65535        Z: 65535
Press any key to continue . . . _
```

Figure 5: Device Query console output.

The GPU servers utilized for this query have two CUDA devices: a Tesla C2075 and a Quadro 600. The Tesla C2075 device has 14 SMs, compared to the Quadro 600 device with 2 SMs, while both devices have the same warp size (32 bytes), maximum thread size per block (1024 bytes), and the same dimension size for both block and grid (1024 bytes). The Quadro 600 device has a faster clock rate at 1,280MHz and more cores at 48, compared to the Tesla C2075 device which has a clock rate of 1,147MHz and only 32 cores.