GPU Lab 1

Part 1: *GPU Fractals: warm up exercises*

Make sure you can logon to the lab workstation and familiarize yourself with the Visual Studio interface. If you have problems please raise your hand.

i. Compile and execute the GPU Mandelbrot example code in the `HPC11/Mandelbrot` directory.

ii. View the output `mandelbrotGPU.png` using a browser window.

iii. Modify the `main` function to generate the fractal in a different part of the complex plane.

iv. Replace the recurrence relation in the

```
__global__ void mandelbrot2D
```

kernel function with one of your choosing

[HINT: see the wiki here for some interesting alternatives.]

v. Generate a cool looking fractal with your new recurrence relationship. Show your image to one of the instructors.

See next page for **Part 2**

Part 2: *GPU Matrix-matrix multiply*

A core task in high performance computing is the evaluation of matrix-matrix multiplication. We consider a restricted example: given two matrices

$$\begin{aligned} \mathbf{A} &\in \mathbb{R}^{M\times 16}, \\ \mathbf{B} &\in \mathbb{R}^{16\times N}, \end{aligned}$$

then their matrix product is a matrix $\mathbf{AB} \in \mathbb{R}^{M\times N}$ with entries given by

$$(\mathbf{AB})_{ij} = \sum_{k=0}^{k=15} A_{ik}B_{kj} \ ,$$

where $0 \leq i < M$ is the row index and $0 \leq j < N$ is the column index of entries in the product matrix. A simple, i.e. non-optimized, CPU implementation is given in

`MatrixMultiply/MatrixMultiply.c`.

This implementation uses a row-major storage convention for the matrices, i.e. the matrix is stored as one long vector with the column index running fastest.

Tasks:

i. Develop CUDA code that allocates memory on the GPU and copies the entries of the matrices **A** and **B** from the CPU host to the GPU arrays.

ii. Decide how you will convert the `MatrixMultiply` function into a CUDA kernel [think about how to design the thread grid for this specific problem.

iii. Implement the CUDA kernel you designed in **ii.**.

iv. Add code that:

   a. Allocates space for the result matrix **AB** .
   b. Invokes the CUDA kernel .
   c. Copies the result from the GPU **AB** array to the CPU host.

v. Test and debug your CUDA implementation `MatrixMultiply`. You should compare the output of your CUDA matrix multiplication routine with the output of the CPU version we have supplied.

vi. Instrument your code with a timer and estimate the number of gigaflops per second it produces on the GPU and the memory bandwidth it achieves in gigabytes per second. [Hint: for accurate results use stream events for timing].