Assignment #5

CS 398 GPGPU Programming, Fall 2020

Due Date:	As shown on the moodle
Topics covered:	Advanced Host/Device Interface, Streams, Events, and Concurrency
Deliverables:	The project files (release version only) including source code and the report `A5Report.pdf`. These files should be put in a folder and subsequently zipped according to the stipulations set out in the course syllabus.
Objectives:	Learn how to use pinned memory and multiple streams to write efficient kernel functions.

Programming Statement

This is the CUDA C programming assignment. Students are expected to finish the programming for a given computation problem (same as Problem 2 in Assignment 2). You will implement matrix multiplication, using pinned memory and task parallelism (multiple streams) to achieve better performance.

Problem Statement

The problem is the same as described in Assignment 2. However, your program should support the following options:

```
mm.exe [argument 1] [argument 2] [argument 3] blockSize tileSize no_of_streams
```

where the possible values for $no\_of\_streams$ are 1 and 3. $no\_of\_streams = 1$ means the program you developed in Assignment 2. Otherwise, when no_of_stream is 3, you are multistreaming the operations into GPU device. You must use shared memory for tile matrix multiplication. `blockSize` is the block size of x and y dimension of 2D thread block. `tileSize` is the length or width of the tiles that are described below.

You should consider at least the following test cases

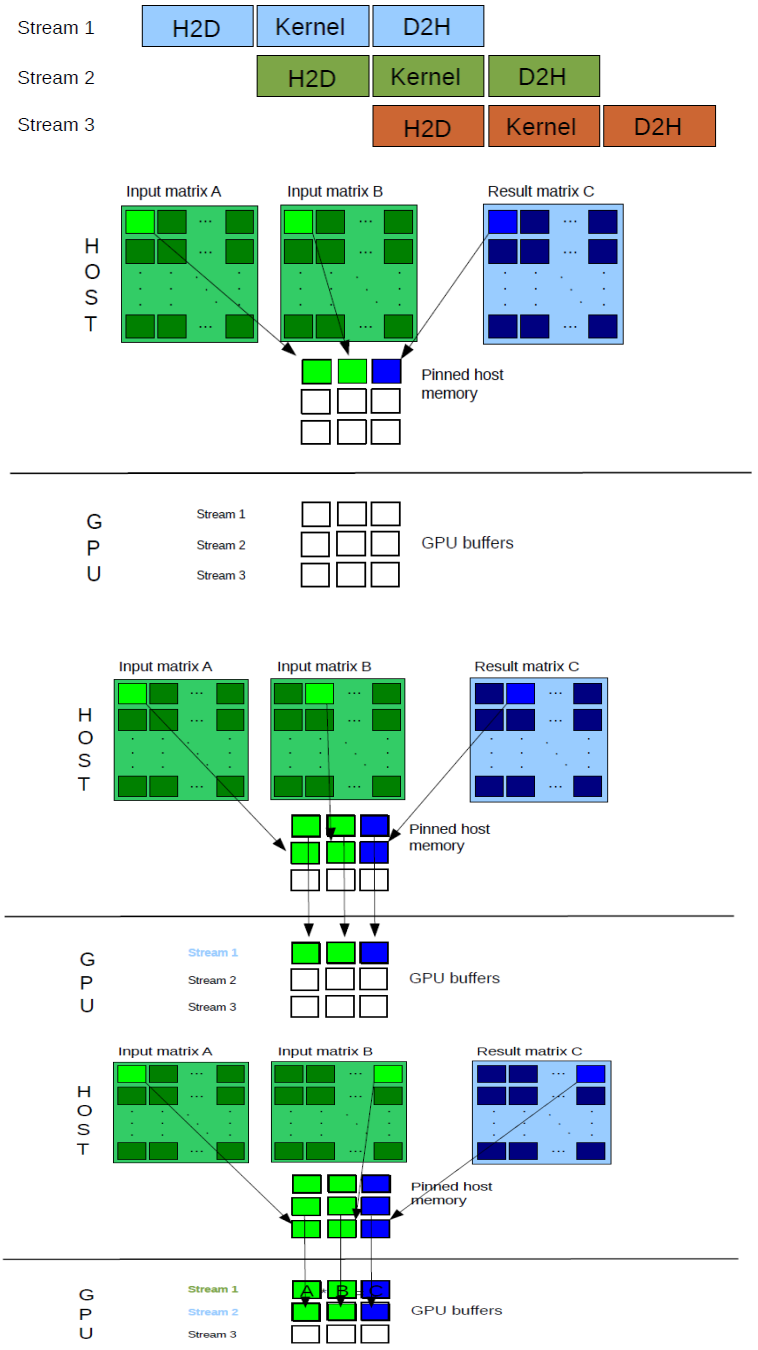
```
mm.exe 100 200 300 #(you have tested in A2 part 2)
mm.exe input0.raw input1.raw output.raw #(you have tested in A2 part 2)
mm.exe input0.raw input1.raw output.raw 16 32 3 #(this is a new mode)
```

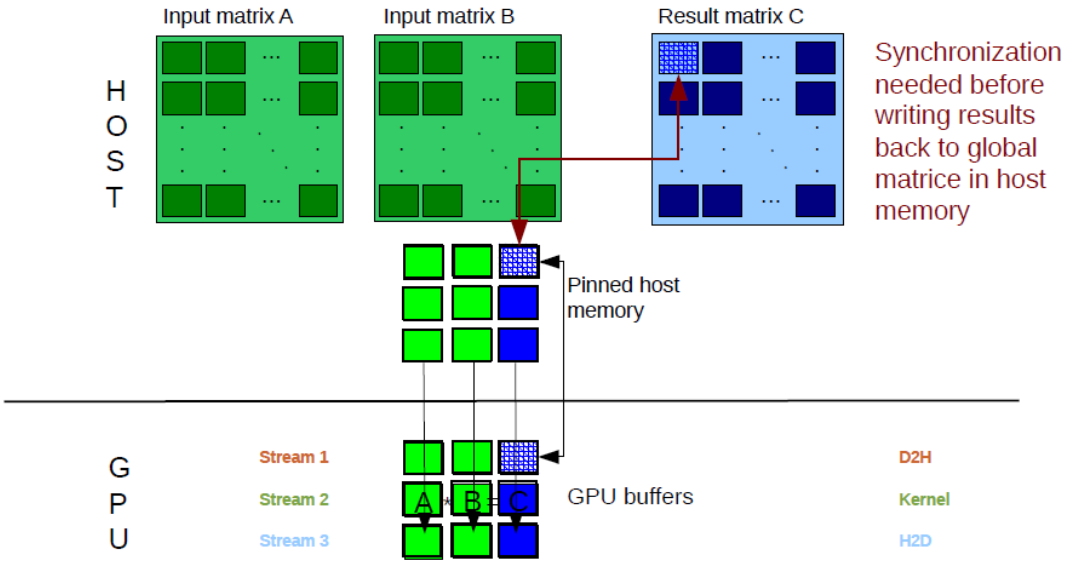
You are required to turn in your homework that is compilable under the Windows environment using Visual Studio.

Streaming

There will be 3 streams for task parallelism. You may slice the matrix into a number of tiles, each tile consisting of `m` rows and `n` columns, where `m` and `n` is the dimension of tiles in rows and columns. You will establish 3 stream operations where the first stream operation is used for copying the data from the host pinned memory to the device, the second stream

operation is used to perform the calculation of tile matrix multiplication for partial results and the third stream operation is used to send back the partial results on the device to the host pinned memory. The synchronization is required to write the results back to the the host memory.





In the example, matrices `A`, `B` and `C` are divided into tiles. Each time one tile from `A` (*blockSize* rows and *tileSize* columns), one tile from `B` (*tileSize* rows and *blockSize* columns) and one tile from `C` ((*blockSize* rows and (*blockSize* columns) work on one stream. The tiles are copied from host page-unlocked memory to host pinned memory and then copied to device memory on a stream. Once it is done, a kernel is launched for the stream. The kernel performs tile matrix multiplication to obtain partial results for matrix `C`. The results will then be transferred back to host pinned memory and are written to the result matrix in host page-unlocked memory.

```
//thread 1
loop over column tiles for A
    loop over row tiles per A column tile
        loop over column tile per B row tile
            select a stream
            this stream wait for host pinned memory to be available
            copy matrix tiles from A, B, C to host pinned memory
            signal to thread 2

//thread 2
loop over column tiles for A
    loop over row tiles per A column tile
        loop over column tile per B row tile
            select a stream
            this stream wait for host pinned memory to be available
            copy tiles from host pinned memory to device memory
            generate event H2D finished for this stream

            this stream wait for H2D finished
            launch kernel to compute tiled matrix multiplication
            generate event Calc kernel finished for this stream

            this stream wait for Calc kernel finished
            copy result tile C from device memory to host pinned memory
            generate event D2H finished for this stream
            signal to thread 3

//thread 3
loop over column tiles for A
    loop over row tiles per A column tile
        loop over column tile per B row tile
            select a stream
            wait for host pinned memory to be available
            synchronize on this stream
            copy result tile from host pinned memory to the result matrix C
            signal to thread 1
```

The example of using semaphore is shown here.

//using Semaphore

//Sem12, Sem23, Sem31

Thread 1

```
loop over column tiles for A
  loop over row tiles per A column tile
        loop over column tile per B row tile
        Select streamId
        if (initOnce[streamId])
          initOnce[streamId] = 0;
        else
          Sem31[streamId].wait()
        Copy(ATile, pmA) //pmA pinned memory
        Copy(BTile, pmB)
        Copy(CTile, pmC)
        Sem12[streamId].signal()
```

Thread 2

```
loop over column tiles for A
  loop over row tiles per A column tile
        loop over column tile per B row tile
        Select streamId
        Sem12[streamId].wait()
        H2D(d_A,pmA);
        H2D(d_B,pmB);
        H2D(d_C,pmC);
        Mark H2D event for this stream
        Wait for H2D event for this stream
        Calc kernel launch
        Mark Calc event for this stream
        Wait for Calc event for this stream
        H2D(pmA, d_A);
        H2D(pmB,d_B);
        H2D(pmC,d_C);
        Mark D2H event for this stream
        Sem23[streamId].signal()
```

Thread 3

```
loop over column tiles for A
  loop over row tiles per A column tile
        loop over column tile per B row tile
        Select streamId
        Sem23[streamId].wait()
        Copy(pmA, A)
        Copy(pmB, B)
        Copy(pmC, C)
        Sem31[streamId].signal()
```

Here is an example. Let $A \times B = C$, where

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

and $C_{ij} = \sum_{k=1}^{2} A_{ik} B_{kj}$. There are two A column tiles as follows:

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$$

and

$$\begin{bmatrix} A_{21} \\ A_{22} \end{bmatrix}$$

Each A column tile has two block matrices, e.g. first A column tile has has two block matrices, A_{11} and A_{21}. Each of block matrices multiplies with a block matrix in a B row tile, e.g. $A_{11} \times B_{11}$ and $A_{11} \times B_{12}$ for first A column tile and first B row tile. Once A column tile changes, B row tile also changes. For outermost loop, there are two iterations to compute partial result of C, which is as follows:

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{21}B_{12} \end{bmatrix}$$

and

$$\begin{bmatrix} A_{21} \\ A_{22} \end{bmatrix} \begin{bmatrix} B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{bmatrix}$$

What to do

What you have to do:

1. Write a CUDA version using shared memory, pinned memory, and concurrent streams to support tiled matrix multiplication.

2. Report the overall time taken (i.e. neither user nor system times). You may need to repeat the experiments 5 times and take the average to amortize any skew that may be caused by system load. Your measurement for latency should include data transfer from Host to GPU device as well as initialization. `You should start to measure once GPU or CPU version program begins e.g. before any memory allocation and data transfer between CPU and GPU`. Report the latency value.

Streams	Matrix Size	blockSize	tileSize	Latency
3	1,000, 1,000, 1,000	32	256	0.51s
3	2,000, 2,000, 2,000	32	256	2.68s
3	3,000, 3,000, 3,000	32	256	8.64s

Table 1: Streaming Latency

Grading Guideline

Your submission will be graded on the following parameters.

1. If your code fails to compile, zero mark is awarded immediately.
2. If you did not use shared memory, host pinned memory, streams/concurrency to implement, zero mark is awarded immediately.
3. Comments/Code Readability - 10 points. Comments are important for CUDA C code to enhance readability. Otherwise, C code cannot be easily maintained.
4. Correctness - 60 points. You must ensure that the code works for all the test cases. Zero mark will be given as the final score if you fail the test cases. Zero mark will be given if the given streaming method is not followed.
5. Latency - 10 points. Close to at least 0.51s (as shown in Table 1) for the case of 1000, 1000, 1000 on the computers in the lab is required for full credit here.
6. Report - 20 points. You should also describe the performance effect you have observed using different number of streams (e.g. 1 and 3) when using pinned memory, with different tile size and block size. The measurement should be based on NVIDIA GeForce GTX 1060 6GB (installed in the Lab).