ME759
High Performance Computing for Engineering Applications
Tiled Matrix Multiplication

Edit the source files **`matrixmul.cu`** and **`matrixmul_kernel.cu`** to complete the functionality of the matrix multiplication **`P=M*N`** on the device. The two input matrices have dimensions that allow them to be multiplied. Moreover, no dimension of the two input matrices is larger than 4096.

There are several modes of operation for the application. Note that the file interface has been updated to allow the size of the input matrices to be read in.

a) No arguments: The application will create two randomly sized and initialized matrices such that the matrix operation **`M*N`** is valid, and **`P`** is properly sized to hold the result. After the device multiplication is invoked, it will compute the correct solution matrix using the CPU, and compare that solution with the device-computed solution. If it matches (*within a certain tolerance*), it will print out "**`Test PASSED`**" to the screen before exiting.

b) One argument: The application will use the random initialization to create the input matrices, and write the device-computed output to the file specified by the argument.

c) Three arguments: The application will read input matrices from provided files. The first argument should be a file containing three integers. The first, second and third integers will be used as **`M.height`**, **`M.width`**, and **`N.width`**. The second and third function arguments will be expected to be files which have exactly enough entries to fill matrices **`M`** and **`N`** respectively. No output is written to file.

d) Four arguments: The application will read its inputs from the files provided by the first three arguments as described above, and write its output to the file provided in the fourth.

Note that if you wish to use the output of one run of the application as an input, you must delete the first line in the output file, which displays the accuracy of the values within the file. The value is not relevant for this application.

Provide a file called **`comments.pdf`** in which you report on the following:

1) For block sizes of 16x16, Set `CUDA_PROFILE=1` and provide both on the forum (under posting HW5-Tiled Matrix Multiplication Profiling/Timing) and in **`comments.pdf`** the content of your **`cuda_profile_0.log`** file (or whatever the file is where the profiling information was dumped after one successful run). In **`comments.pdf`** provide short comments on the results you see reported in the **`cuda_profile_0.log`** file.
2) Just like before, but use block sizes of 32x32.
3) Explain why the timing differences between the two cases and why one is superior to the other.
4) Run your application to multiply two matrices of dimension 4096x4096. Report the time required to perform the matrix multiplication on the CPU and then on the GPU. When timing the GPU version, make sure you report both the inclusive and exclusive times. Provide your timing results both in **`comments.pdf`** and on the forum.

5) If your input matrices are square, what is the largest dimension that your code can handle on GTX480 after you further modify your code? Provide your timing and profiling results both in **`comments.pdf`** and on the forum.
6) Include in **`comments.pdf`** and also upload on the forum a plot (**`png`** format recommended) that shows how both the CPU and *inclusive* GPU times scale with the dimension of the problem. The scaling analysis should be based on matrices of dimension 32x32, 64x64, 128x128,…,4096x4096. When you provide this plot make sure you build the executables in "production" mode (as opposed to "debug" mode).
7) You might notice that if you have very large matrices, your accuracy test (that verifies the result on the GPU against the result on the CPU) might fail. Why do you believe to be the cause of this odd behavior? That is, working ok for smaller matrices but not when dealing with large matrices.

Grading:
Your submission will be graded using the following scheme.
a) Demo/knowledge: 60%
- Produces correct result output files for provided inputs

b) Functionality: 30%
- Shared memory is used in the kernel to mask global memory access latencies.

c) Report: 10%
- Good comments provided in relation to questions 3) and 7) above.