Project 2 : In the Interest of Time

Due 9/30/2015 @ 12pm (noon)

TOPICS:

- NVIDIA Threads Architecture
- Time Library
- File I/O
- Random Numbers
- Parallel Algorithm Analysis

BACKGROUND:
During class, we discussed how to implement a matrix multiplication subroutine. We also discussed several metrics for measuring parallel algorithm performance.

DIRECTIONS:
Design and implement C and CUDA programs that called MatrixMult.c and MatrixMult.cu, respectively.

The MatrixMult programs will use a command line argument that corresponds to the row/column length to multiply two matrices with random single precision numbers (i.e., floats). They will then output 1) the entire execution time in seconds from the beginning to the end of the function/kernel call only, and 2) the product of the two matrices with each element printed out to a file called "product.dat" in a tab-delimited, row/column format. The C program must implement a serial code via a function call, and the CUDA program must implement a parallel code via a kernel call. **It is completely up to you to implement the serial and parallel algorithms, including any parameters associated the implementation except that the parallel algorithm must use tiles.**

You must check for 1) the appropriate number of command line arguments and 2) whether the command line argument corresponds to a positive number (you may round floating point inputs). Appropriate error messages must be issued, followed by a graceful exit.

To analyze the relative performances of the CPU vs. GPU implementations of your matrix multiplication algorithms, you will compute the time it takes to complete a matrix multiplication function/kernel call. For the GPU implementation, you will determine the optimal tile size that you will use for your analysis. You will then construct a graph of the speedup vs. matrix size (i.e., row/column length) for several tile sizes to see how their performances vary with system size and tile size. It is up to you to choose appropriate values for the matrix size to construct your graph, but a matrix size of 2,718 will be used for testing during grading.

Hint: In the execution of your GPU implementation, for accurate timing information, you must call the kernel twice because the GPU initialization takes time that is unrelated to the algorithm.

IMPLEMENTATION NOTES:
Any program that does not compile or does not have correctly constructed Makefiles will not be graded. The C program must compile using g++ and the CUDA program must compile using nvcc.

COMMENTS AND STYLE:
Although there will be no formal policy on commenting and style, the reader should able to easily follow the main purpose of the code. Each set of code that does something significant must be commented. The variable names should be easily recognizable and acronyms should be avoided if possible.

Do not be surprised if help is not forthcoming if your code is poorly commented and/or difficult to follow. <u>You have been warned.</u>

PROJECT SUBMISSION:
The programs should be in appropriate directories named “MatrixMult-cpu” and “MatrixMult-gpu”. Both programs must have corresponding Makefiles. The contents of the directories must be archived in a tarball that is gzipped called Proj2.tar.gz.

Place the gzipped tarball in your Drop Box on Sakai before it is due.

PLEDGED WORK POLICY:
Assignments in Computer Science courses may be specified as "pledged work" assignments by the professor of the course. When an assignment is specified as "pledged work" the only aid that the student may seek is from either the course professor or an assistant that the professor has explicitly specified. On "pledged work" assignments the student may not use the services of a tutor.

For this project, you may discuss only <u>basic C syntax</u> with others. Any other discussions of the project are strictly prohibited except with the professor of the course. Your code and your implementation of the project must be the product of your own work.