Lets have a little fun with matrix-by-matrix multiplication and threads. This problem will allow us to exercise some multithreaded codes while examining the effect of varying the number of simultaneous threads that operate on a problem.

Write a serial AND a threaded matrix-by-matrix multiplication solution.

The goal will be to run your code over the following scenarios and time them to see if you observe any differences in your machine's performance:

- [10pts] Serial version of the code (without threads)
- [20pts] Serial "like" version of code using a series of one thread calls at a time
- [30pts] Do 1-thread for each element in matrix C (one for each C_{rc}). Spawn them all at once and let the operating system deal with the thread scheduling.
- [30pts] Do 1-thread (at a time) for each core on your machine
- [10pts] Analysis and summary of the results – timing of runs on your machine – what you observed and why. This is a pdf document.

The thread-based program should be written such that a thread handles each row-by-column multiplication process. Thus in the example below of two 5x5 matrices AxB=C there would be 25 threads generated.

Since this function will only read matrix A and B, those values may be passed by reference AND with C being written exclusively into any given cell, C may also be passed by reference. This is exploiting shared memory access to A, B, and C.

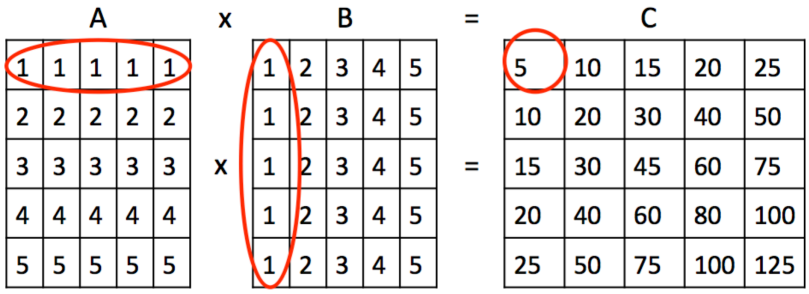
The function call should return a status of success or failure for the call itself. (we will not really use this return value, it is just a good practice.) We observe that each of these row-by-column operations is actually a dot product or vector-by-vector multiplication that produces a single result. Thus our multiplication function may simply do the dot product with the resulting value placed into the proper location in C.

The function to calculate the dot product should look like this…
Status = DotProduct(vectorA_ptr, vectorB_ptr, elementC_ptr, element_count)
vectorA_ptr is a pointer to a vector of long integers
vectorB_ptr is a pointer to a vector of long integers
elementC_ptr long integer, is the result of the dot product
element_count long integer, is the number of elements in the vectors

Matrix by Matrix multiplication equation:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Visually the row-by-column operation looks like this:



You will want to write code to generate the values to go into A and B. We will use the pattern of values given in the above example where; for A, the rows all contain the same value and for B, the columns all contain the same value. Write your functions to follow the patterns… Thus for a 1,000x1,000 matrix A would contain $1x10^6$ entries. You may want to write a function and use threads to fill the matrix. Not sure how large your machine will let you go here. I do expect you to push that limit though… However, regardless of your machine's capability, limit the maximum to the 1kx1k matrix problem – if you can do that large... Otherwise, tell me what your max is as part of your analysis and submission summary document.

Some important notes:

The world is your resource… Do not feel guilty to google what others have done and to work from that code. You will still have to make it your own. I only consider it plagiarism, as opposed to software reuse, if you reuse one of your classmate's codes as your assignment submission.

As some of you have discovered, printing intermediate results to screen during timing will affect that timing – slows it down… i/o is a slow process versus cpu computation. What to do? Write your codes with necessary debug print statements that you may validate your computation. When all is good and you have verified it works. Comment out those print statements, recompile and rerun. How to verify your results during a "speed run"? I would be satisfied with just the output of the last row-column entry (lower-right most cell in matrix) to the screen AFTER all computation and timing is complete.

==========

Submission instructions:

Due: November 22, 2020 at 11:59pm

via Email
to: sscott@tntech.edu
Subject: CSC4760-5760 MatMult

Send email with 3 files as attachments
1. source
2. make file to compile your codes
3. video screen capture of your program running.