Project 2: Parallel Matrix Multiplication using Multi-threading

1. Assignment

In this project you are asked to implement a parallel algorithm using multi-threading: You are given two matrices A and B each of size NXN, and you are asked to implement matrix multiplication of A and B to compute C in parallel. A serial (single-threaded) version of the algorithm can easily be implemented as follows (C = A x B):

```
for (int i=0; i<N; i++){
    for (int j=0; j<N; j++){
        C[i][j] = 0;
        for (int k=0; k<N; k++)
            C[i][j] += A[i][k]*B[k][j];
    }
}
```

Assume that you have 2 CPUs, and you want to solve this problem in parallel. The easiest parallel algorithm would be one that creates two threads T1 and T2 and have T1 compute the result for rows 0 to N/2-1 and T2 compute the result for rows N/2 to N-1. Assuming kernel threads, T1 and T2 would most likely be scheduled concurrently, one running in one CPU, and the other running in the other CPU. Thus we can expect the problem being solved twice as fast; that is, the algorithm runs 2 times faster than the single-threaded version (we get a speedup of 2).

2. Main Thread

The pseudocode for your main thread is given below:

```
Main Thread:
(1) Check command line arguments for validity. Exit in error
(2) Allocate space for matrices A, B and C (NxN integers each)
(3) Read matrix A & B from their respective files and close the files
(4) Start timer (use gettimeofday)
(5) Create worker thread(s) to compute the result (matrix C)
(6) Wait for the termination of the worker threads
(7) Stop the timer (use gettimeofday)
(8) Print the total execution time
(8) Write matrix C to the output file
(8) Free up allocated spaces for matrices A, B and C
```

In step (1) the main thread checks the validity of the command line arguments and terminates the program if the arguments are missing. Assume that all command line arguments provided will be correct. It then allocates space for NxN integers for matrices A, B and C (step 2) and reads the integers for matrix A and B from their respective files (step 3). The main thread then starts the timer (step 4), creates worker thread(s) in (step 5) and then waits for their termination in (step 6). When all threads terminate, the main thread prints the execution time (step 7), write the resulting matrix C to the output file, frees up the allocated space and exits the process in step (step 8).

To compute the total execution time, the main thread starts a timer at step (4) and stops the timer at step (7) and prints the execution time at step (8). You can do this using the C library function "gettimeofday", which returns the current system time in seconds and microseconds. For more information on gettimeofday, type "man gettimeofday" on the command prompt.

Assume that the integers are stored in a text file, one number on each line. You should simply open the text file and read NxN integers from this file and fill them into either A or B depending on which one you are reading.

3. Testing

Name your program mult.c. Your program will be run with the following command line arguments:

```
% mult <numThreads> <N> <filename for A> <filename for B> <filename for C>
% mult 3 100 a.txt b.txt c.txt
```

The first argument to your program is the number of threads you will create to perform the matrix multiplication. This could be a number between 1 and 8. The second argument is the matrix size, N. If N = 100, then the matrix has NxN = 100x100 integers. The third and fourth arguments are the names of the files that contain the values for matrices A and B respectively. The last argument is the name of the file where you will dump the resulting matrix C. You must implement the project as described above. Failure to observe the rules will result in loss of points.

When you write to the output file make sure that each number is written to a separate line. To test for correctness, I will simply compare the output from my program to the output from your program using the Linux "**cmp**" utility. For example, if my program output is "c1.txt" and your program output is "c2.txt", I will run the following Linux command:

```
% cmp c1.txt c2.txt
```

If the files are the same, **cmp** terminates without printing any output. If there are discrepancies between the files, **cmp** prints the lines where they are different. So make sure that your program produces the correct result. For you to test your code, I have provided you with 3 files: a.txt, b.txt and c.txt each containing matrices of size 100x100. I also provided you with genNumbers.c that I use to generate the matrices. You should use this utility to generate larger (e.g., 1000x1000) matrices to test your application. You will only observe speedups with large matrices.

4. Submission

Submit your application source code via Brightspace. This project is due on Sunday 4/20/2025 11:59pm. I will be testing your application on a **Linux** machine.