Fall 2023
CS3413
Lab 3
Threads for Data Parallelism

This lab is demonstrating data parallelism using pthreads in C. Matrix multiplication is a computation-intensive task that can be accelerated very well.

Matrix multiplication

The task of matrix multiplication can be described as follows: for every element of matrix C (where the rows are indexed by i , and the columns are indexed by j)

$$C_{i,j} = \sum_{k=0}^{n} A_{i,k} * B_{k,j}$$

In other words, every element of the row i from the first matrix is multiplied by the corresponding element from the column j of the second matrix and the sum is placed in the result matrix at the position i,j.

For example

$$\begin{bmatrix} 5 & 5 \\ 0 & 2 \end{bmatrix} * \begin{bmatrix} 6 & 2 \\ 0 & 9 \end{bmatrix} = \begin{bmatrix} 5*6+5*0 & 5*2+5*9 \\ 0*6+2*0 & 0*2+2*9 \end{bmatrix} = \begin{bmatrix} 30 & 55 \\ 0 & 18 \end{bmatrix}$$

Note that in general cases, K, the number of columns of the first matrix A(NxK), should be equal to the number of rows in the second matrix B(KxM), and the result output will have (NxM) dimensions. In this lab, we are focusing on the square matrices, which makes the task even simpler, as both the input matrices as well as the output matrix have the same dimension NxN.

You can read more about the problem online, or study from the skeleton code.

Skeleton program

You are to familiarize yourself with the program provided on D2L. The program requires two command line arguments: the size of the matrix, as well as the number of threads to create for parallel computation. The first parameter determines the number of rows (and columns), and the second parameter is not used in the skeleton code. You will need to use it for generating this many threads.

1. The program generates two matrices of the specified size (if the size is 2, the size of each matrix will be 2x2) and populates them with random numbers. Note that the seed value (the value specified as a parameter to the srand() call) is fixed, so the “random” numbers will always be the same.
2. Further, the program allocates the result array – the space where the results of multiplication are supposed to be stored.
3. The program continues with invoking the multiply_matrices function that performs the computation.

4. The execution duration of this function is measured by comparing two timestamps, recorded before and after the parameters are prepared and the function is called.
5. The duration of the computation is reported to the console.
6. The program then generates a new array for storing the results of a threaded computation.
7. The program repeats the computation in a parallel way (your task is to):
    - Provide an implementation of the **multiply_matrices_threaded** function, that should be creating a thread per one or more rows of the first matrix. Hint: the implementation is not that different from the single-threaded version.
    - Invoke the function, creating **max_threads** worker threads.
    - Properly terminate the threads.
    - Place this code between the two lines of code recording the timestamps for the threaded execution (see the comments in the skeleton file).
    - Answer the questions from the questions section.
8. After the parallel computation is finished (if the function is implemented correctly and the threads are used properly) the real time it took the threads to perform the computation is reported.
9. Finally, the results of the multi-threaded computation are compared with the results of the single-core computation, and an error is printed if there is a difference.

The program contains an implementation of the print function for the matrices, that is not used by default, but can be used for debugging.

Questions

- Run the program with the problem size of 1000 and 10 threads, what is the approximate speedup you are achieving?
- Is there a problem size / number of threads combination that slows down the computation process? Why do you think it is happening?
- What is the minimum size of the problem that benefits from creating an extra thread?
- Does using the threads always improve execution duration?
- Guesstimate and comment on the nature of growth of the speedup with the number of threads – is it linear, exponential, are there any limits?

Please submit your C file with the solution, as well as with the answers included in the comments within the C file.