Lab 3 – Documentation

Goal

Divide a simple task between threads. The task can easily be divided in sub-tasks requiring no cooperation at all. See the caching effects, and the costs of creating threads and of switching between threads.

Requirement

Write several programs to compute the product of two matrices.

Have a function that computes a single element of the resulting matrix.

Have a second function whose each call will constitute a parallel task (that is, this function will be called on several threads in parallel). This function will call the above one several times consecutively to compute several elements of the resulting matrix. Consider the following ways of splitting the work betweeb tasks (for the examples, consider the final matrix being 9x9 and the work split into 4 tasks):

Each task computes consecutive elements, going row after row. So, task 0 computes rows 0 and 1, plus elements 0-1 of row 2 (20 elements in total); task 1 computes the remainder of row 2, row 3, and elements 0-3 of row 4 (20 elements); task 2 computes the remainder of row 4, row 5, and elements 0-5 of row 6 (20 elements); finally, task 3 computes the remaining elements (21 elements).

Each task computes consecutive elements, going column after column. This is like the previous example, but interchanging the rows with the columns: task 0 takes columns 0 and 1, plus elements 0 and 1 from column 2, and so on.

Each task takes every *k*-th element (where *k* is the number of tasks), going row by row. So, task 0 takes elements (0,0), (0,4), (0,8), (1,3), (1,7), (2,2), (2,6), (3,1), (3,5), (4,0), etc.

For running the tasks, also implement 2 approaches:

Create an actual thread for each task (use the low-level thread mechanism from the programming language);

Use a thread pool.

Experiment with various values for (and document the attempts and their performance):

The sizes of the matrix;

The number of tasks (this is equal to the number of threads when not using a thread pool);

The number of threads and other parameters for the thread pool (when using the thread pool)

Implementation

I created 6 different programs.

In the Helper class, there will be 4 different methods:

1. One for generating a random matrix of size n*n
2. One for generating a 0-initialized matrix of size n*n
3. One for getting the value of the element on row r and column c from the resul matrix obtained from the multiplication of firstMatrix and secondMatrix
4. The last one: in the result matrix,elements are numbered from 0 to n*n-1; the function takes the index of the first element in the resultMatrix to be computed and the index of the last one and sequentially computes the value for each element and writes it in the memory

We have 3 ways of splitting the compuatation into tasks and for each of these, I implemented two approaches for running the tasks:

1. Manully creating each thread
2. Using a thread pool that receives multiple Runnable objects

After running the tests, the following results were obtained:

Program 1:

Test #1:
- -matrix size: 100
- -number of tasks: 10
- -number of threads: 10
- -time: 17ms

Test #2:
- -matrix size: 100
- -number of tasks: 20
- -number of threads: 20
- -time: 16ms

Test #3:
- -matrix size: 200
- -number of tasks: 20
- -number of threads: 20
- -time: 24ms

Program 2 – thread pool:

Test #1:
- -matrix size: 100
- -number of tasks: 10
- -number of threads: 5
- -time: 2ms

Test #2:
- -matrix size: 100
- -number of tasks: 20
- -number of threads: 5
- -time: 2ms

Test #3:
- -matrix size: 200
- -number of tasks: 20
- -number of threads: 10
- -time: 2ms

Program 3:

Test #1:
- -matrix size: 100
- -number of tasks: 10
- -number of threads: 10
- -time: 15ms

Test #2:
- -matrix size: 100
- -number of tasks: 20
- -number of threads: 20
- -time: 18ms

Test #3:
- -matrix size: 200
- -number of tasks: 20
- -number of threads: 20
- -time: 22ms

Program 4 – thread pool:

Test #1:
- -matrix size: 100
- -number of tasks: 10
- -number of threads: 5
- -time: 2ms

Test #2:
- -matrix size: 100
- -number of tasks: 20
- -number of threads: 5
- -time: 2ms

Test #3:
- -matrix size: 200
- -number of tasks: 20
- -number of threads: 10
- -time: 2ms

Program 5:

Test #1:
- -matrix size: 100
- -number of tasks: 10
- -number of threads: 10
- -time: 14ms

Test #2:
- -matrix size: 100
- -number of tasks: 20
- -number of threads: 20
- -time: 15ms

Test #3:
- -matrix size: 200
- -number of tasks: 20
- -number of threads: 20
- -time: 23ms

Program 6 – thread pool:

Test #1:
- -matrix size: 100
- -number of tasks: 10
- -number of threads: 5
- -time: 2ms

Test #2:
- -matrix size: 100
- -number of tasks: 20
- -number of threads: 5
- -time: 2ms

Test #3:
- -matrix size: 200
- -number of tasks: 20
- -number of threads: 10
- -time: 2ms