# Introduction

Report about speedup and runtimes for sequential and parallel solutions to matrix multiplication.

# Sequential Matrix Multiplication

For the most part I used a sequential solution like the one shown by Eric in the lesson. I transpose matrix B, using the function he published, and then, for each “square” in the matrix, I calculate its value using the function I called “calculate” in my code.

# Parallel Solution

The calculation of the answer for each square in the matrix is independent from all the others. Because of that, I simply treat the calculation of each square as its own job, and have a queue with these jobs listed, which a pool of worker threads could pull from.

To do this, I created a fixed threadpool, which creates a fixed amount of threads that will do any job submitted to them, and simply pick up a new job whenever the previous job is done. Doing it like this allows me to reuse the threads, not having to go through the overhead of creating threads more often than I have to.

The only synchronization point is at the end, where the main thread waits for the threadpool to finish and shut down. This is efficient, since the program doesn’t have to waste time synchronizing at multiple points.

# Measurements

|  |  |  |  |
| --- | --- | --- | --- |
|  | Kjøretid (ms) | |  |
| N | Sekvensiell | Parallell | Speedup |
| 100 | 1,386 | 6,439 | 0,215 |
| 200 | 7,456 | 16,596 | 0,449 |
| 500 | 104,596 | 161,579 | 0,647 |
| 1000 | 981,316 | 512,979 | 1,913 |

For low values of N (small N x N matrixes), obviously the speedup was below 1, due to the overhead associated with creating threads and having the task queue. Some addition tests show that it breaks even at roughly N = 700, and the speedup can vary from 1,5 to 2 at N = 1000.

# Running the Program

java MatrixMulti N

N = The height and width of the matrixes.

# Summary

The parallel solution was written using a Java Fixed ThreadPool, and it’s very easy to see that the higher values for N gives better speedup. My solution only has a single synchronization point, at the end, where the main thread is waiting for the workers to finish.