IN3030 assignment 2

marksverdhei

February 2020

Introduction

In this assignment, I parallelize the naïve matrix multiplication algorithm for square matrices, and measure speedup.

Matrix multiplication

$$A \times B = C$$

is computed as following

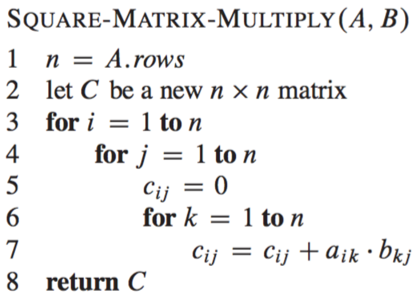
$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Hardware information:

- Intel Core i5-4210U CPU @ 1.70GHz
- 2 cores, 4 threads

Sequential implementation

The sequential implentation is based on the following algorithm:



```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        $c_{ij} = 0$
        for k = 1 to n
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C
```

Figure 1: pseudocode for naïve matrix multiplication

This algorithm is known as the naïve, or classical algorthim for matrix multiplication. It consists of three nested for loops and runs in cubic time: $\mathcal{O}(n^3)$.

It is worth mentioning that there are faster algorithms for solving this problem. For the transposed variants, i only had to swap the indices, so that $b_{kj} \to b^T_{jk}$ and $a_{ik} \to a^T_{ki}$

Parallel implementation

For the parallelization, the amount of threads made is based on how many cores are availible for the jvm, so the number of threads is the closest higher power of 4 to the p number of cores there are. This is to maximally reduce the overhead of creating more threads than needed. The algorithm recurisvely partitions the matrix into partitions of 4 until there are more threads than cores.

Benchmarks

n	default		b transposed		a transposed	
	sequential	parallel	sequential	parallel	sequential	parallel
$n = 100$	4.55ms	5.04ms	3.74ms	1.65ms	3.9ms	2.07ms
speedup	0.9		2.27		1.88	
$n = 200$	12.98ms	8.99ms	8.82ms	6.19ms	29.38ms	14.77ms
speedup	1.44		1.42		1.99	
$n = 500$	336.2ms	165.08ms	142.11ms	50.58ms	560.66ms	1.29s
speedup	2.04		2.81		0.43	
$n = 1000$	13.59s	8.3s	1.35s	632.76ms	32.36s	19.82s
speedup	1.64		2.13		1.63	

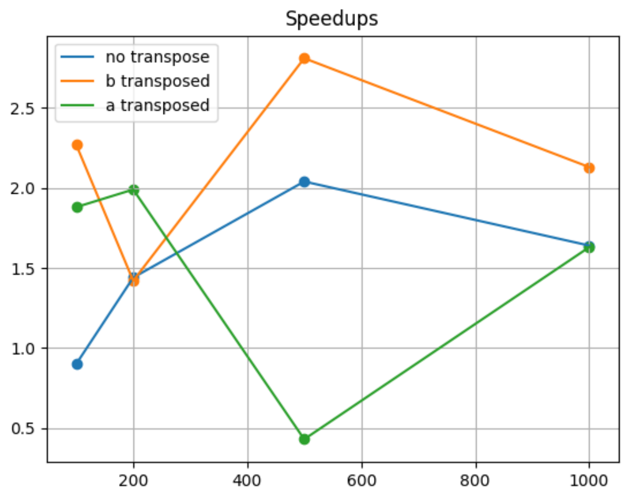


Figure 2: Speedups

Discussion

The benchmark results demonstrate how important array alignment is for performance. Row indexing is likely optimized, while consecutive column indexing is expensive in terms of performance. Because of that, matrix multiplication with b transposed is clearly the fastest, and matrix multiplication with a transposed is slower. I cannot really explain the sudden downspike for speedup where a is transposed and n=500, but it seems to be consistent when i rerun the benchmarks.

User guide

The classes Matrix and ParallelMatrix are the two classes containing the matmul methods. They are designed to be useful modules for other programs, and thus, do not have any main methods themselves. The class TestMatrix is what I have used to test the correctness of the Matrix and ParallelMatrix classes respectively. The class Benchmark is what i wrote and used to measure the time each of the algorithms take on the input sizes $N = \{100, 200, 500, 1000\}$. The matrix class contains some useful utilities for double matrices such as equals, copyOf, toString etc.

To run the tests, compile the classes and run the program (e.g. javac *.javas java -ea TestMatrix). It is important to add the -ea flag when running the tests in order for them to work. To run the benchmark, no additional arguments or flags are needed.

Keep in mind that the test and benchmark classes are dependent of the Oblig2Precode class.

Conclusion

The parallelization of the naïve matrix multiplication algorithm was somewhat successful, providing speedups up to 2.8 though the speedup was less than one in some cases. Speedup was highest when b was transposed, and lowest when a was transposed.