Valentin Gegoux

Multicore Computing
Project #1, Problem 2

I/ Introduction

This document aims to report the performances of a multithreaded matrix multiplication program. This program uses the **static cyclic** strategy.

II/ How to use

Compile and run the program by typing one of the following command lines within the `problem2/` directory of the project:

```
$ java MatmultD.java <number of cores> < <matrix file>
```

III/ Testing environment

RAM: 18 Go
CPU Type: 11-core Apple M3 Pro (6 performance + 5 efficiency)
Hyperthreading: No (1 thread per core)
Clock Speed: Up to 4.06 GHz (P-cores)
Architecture: ARM64 (Apple Silicon)
OS: Sequoia 15.3.2

IV/ Measurements

The multiplied matrices have a size of 1000x1000.

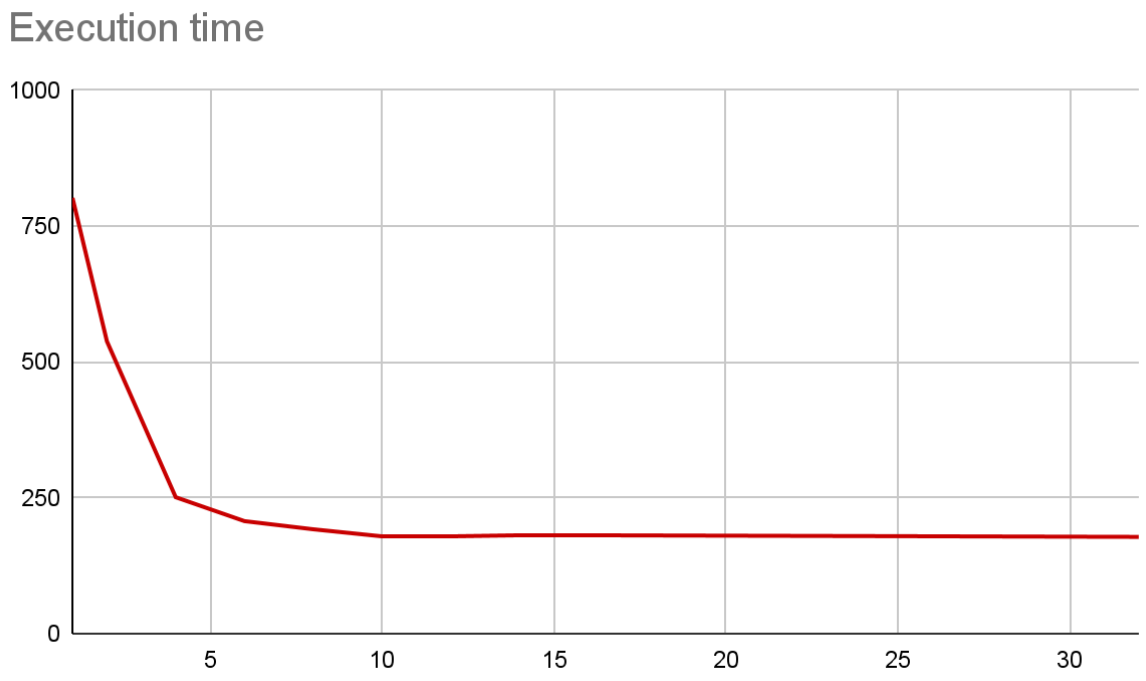
For this test, the chosen task size is 30. It's a good compromise for ideal effectiveness:

- a too low value would increase the overhead: the ratio between calculation time and task finding time would be too low.
- a too high value may lead to loss of CPU resources: if there are more threads than available tasks, some will remain idle.

The maximum number of threads being 32 here, a matrix of 1000x1000 can be calculated by almost all threads with a task size of 30 (32*30 = 960).

Execution time (in ms)

threads	**1**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**32**
exec time (ms)	*802*	*538*	*251*	*207*	*192*	*179*	*179*	*181*	*181*	*178*

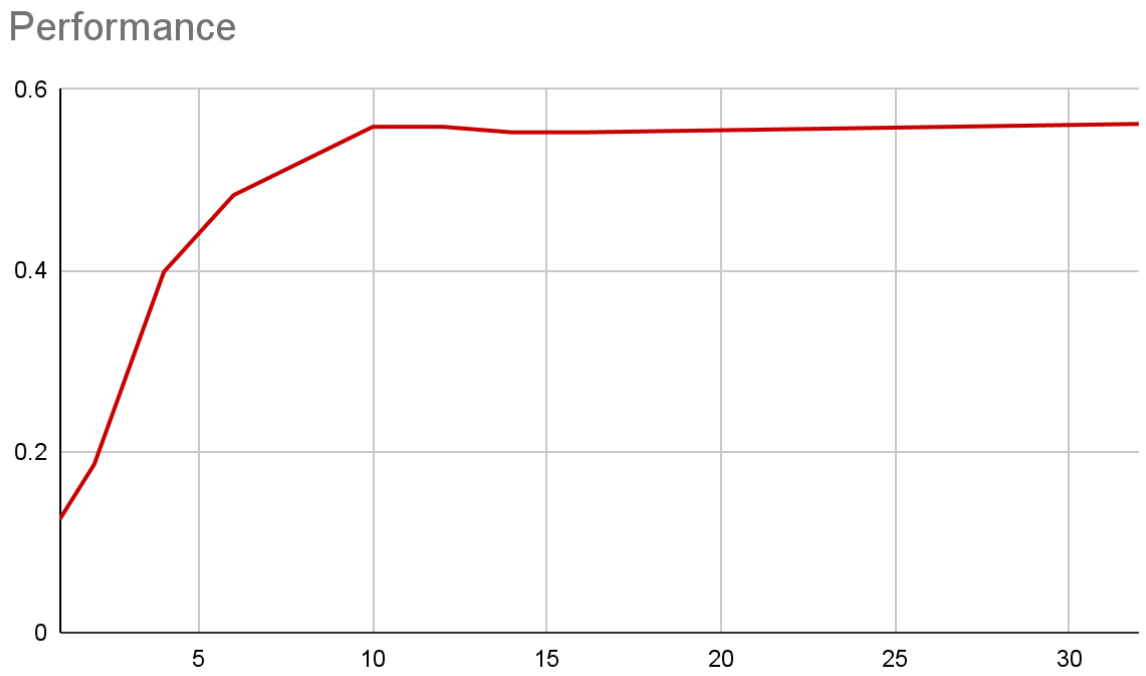


Graph showing the execution time (vertical axis) depending on the number of threads (horizontal axis)

Performance

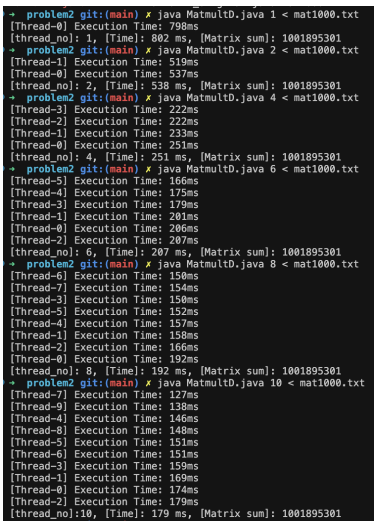
In order to ease readability, I multiplied the performance by 100. It doesn't affect the relationship between each value.

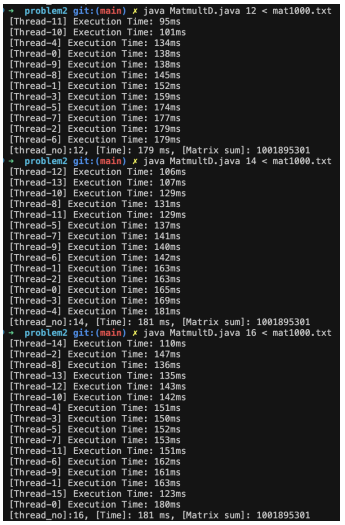
	1	2	4	6	8	10	12	14	16	32
performance (1/exec time)	*0.125*	*0.186*	*0.398*	*0.483*	*0.521*	*0.559*	*0.559*	*0.552*	*0.552*	*0.562*

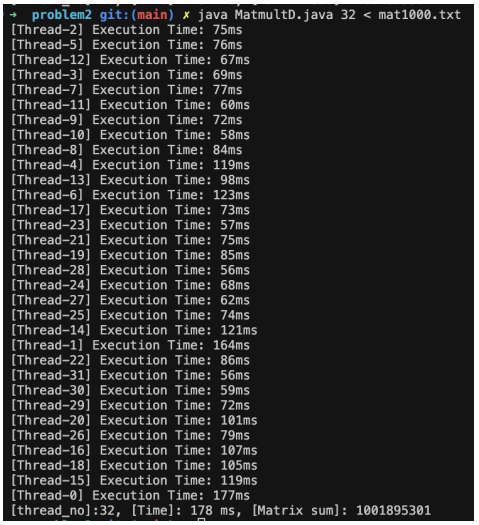


Graph showing the performance (vertical axis) depending on the number of threads (horizontal axis)

V/ Screenshots







VI/ Analysis of the results

As in problem 1, the performance grows as the number of threads increases until it reaches a ceiling - 10 threads here. The shape of the curves look similar to a logarithm function. This ceiling is probably due to the fact that the machine has a limited number of physical cores (11), in addition to overhead and memory bandwidth limitations.

Additionally, we can observe - as in problem 1 - that performance gain slows down from 6 threads. It confirms that the 6 first threads are run by performance cores, and the 5 other ones are run by efficiency cores, which are slower than p-cores.