Multicore Computing Project 3

Joshua BRIONNE - 50241647

1. Compilation & Execution

```
gcc -fopenmp -O2 prob1.c -lm
```

Compilation has been made on a Linux Ubuntu 22.04.3 LTS system using the -02 optimization flag.

2. Environment

Propriété	**Valeur**
Java Version	21
OS Name	Linux
OS Version	6.8.0-52-generic
Architecture	amd64
Available processors (cores)	8
Max memory (MB)	3936

3. Results

[REF] exec time (unit: ms)

Scheduling	**Chunk size**	**1**
None	None	42.008

[REF] Performance (1 / exec time)

Scheduling	**Chunk size**	**1**
None	None	0.023

4. Using OpenMP

exec time (unit: ms)

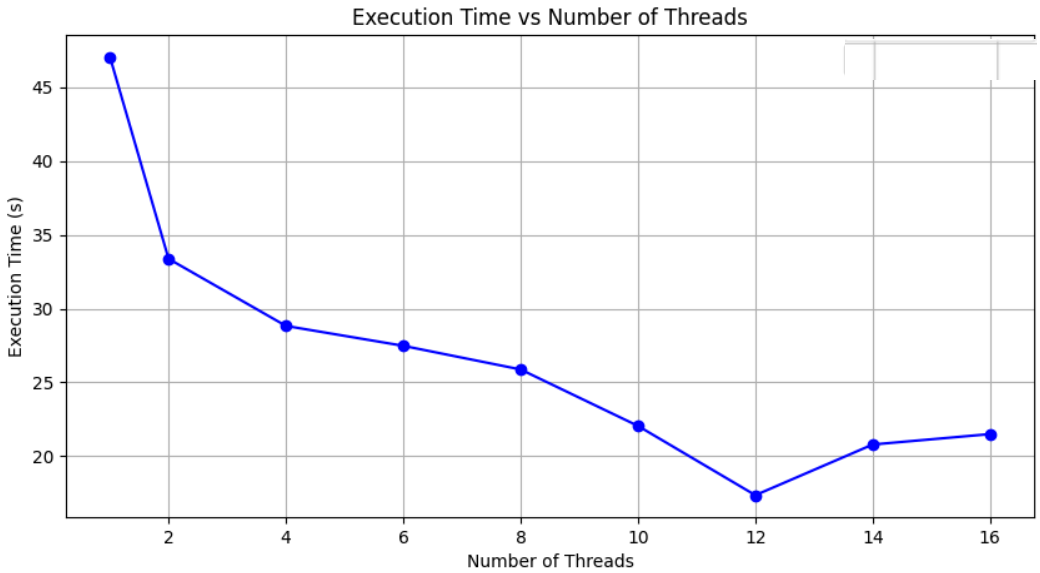
Scheduling	**Chunk size**	**1**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
static	default	47.050	33.371	28.828	27.480	25.877	22.054	17.364	20.796	21.502
dynamic	default	45.403	32.600	29.843	35.035	32.867	26.588	22.836	18.603	19.007
static	10	46.342	39.619	22.122	31.914	29.079	17.740	21.084	20.375	19.017
dynamic	10	47.000	40.733	27.913	22.369	26.388	21.925	19.518	18.699	21.066

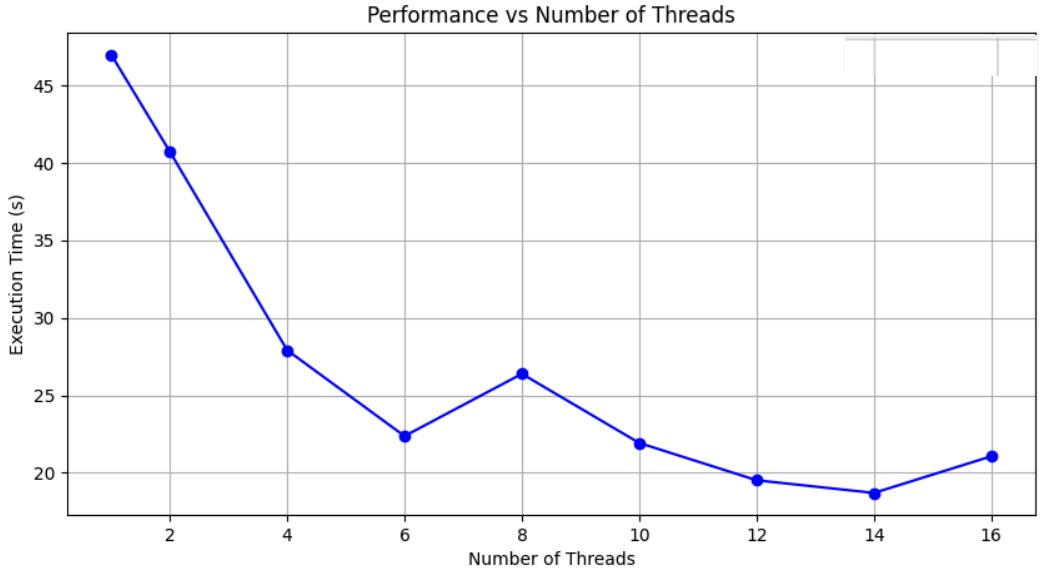
Performance (1 / exec time)

Scheduling	**Chunk size**	**1**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**
static	default	0.021	0.030	0.035	0.036	0.039	0.045	0.058	0.048	0.047
dynamic	default	0.022	0.031	0.034	0.029	0.030	0.038	0.044	0.054	0.053
static	10	0.022	0.025	0.045	0.031	0.034	0.056	0.047	0.049	0.053
dynamic	10	0.021	0.025	0.036	0.045	0.038	0.046	0.051	0.053	0.047

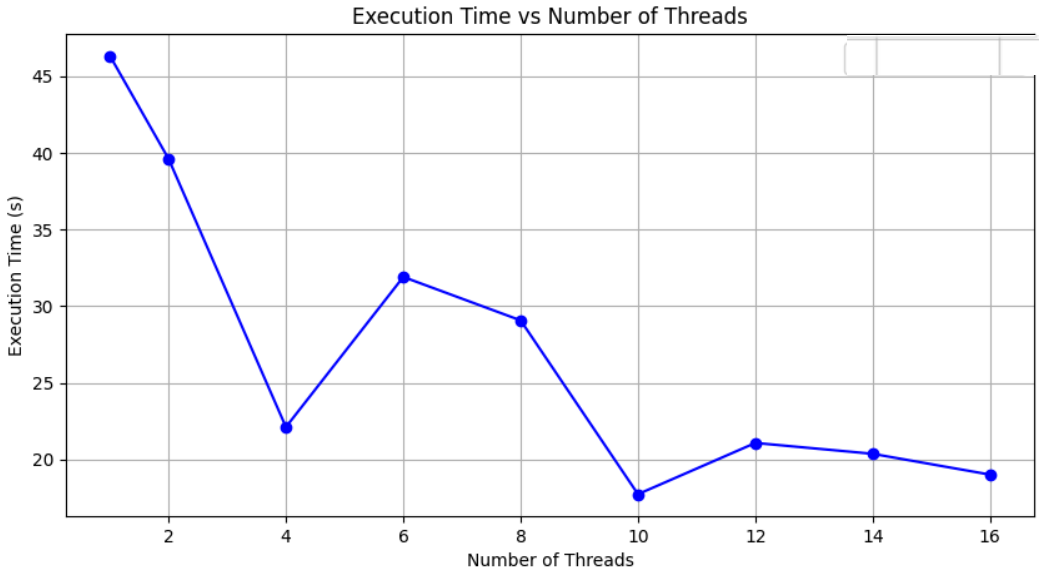
5. Graph:

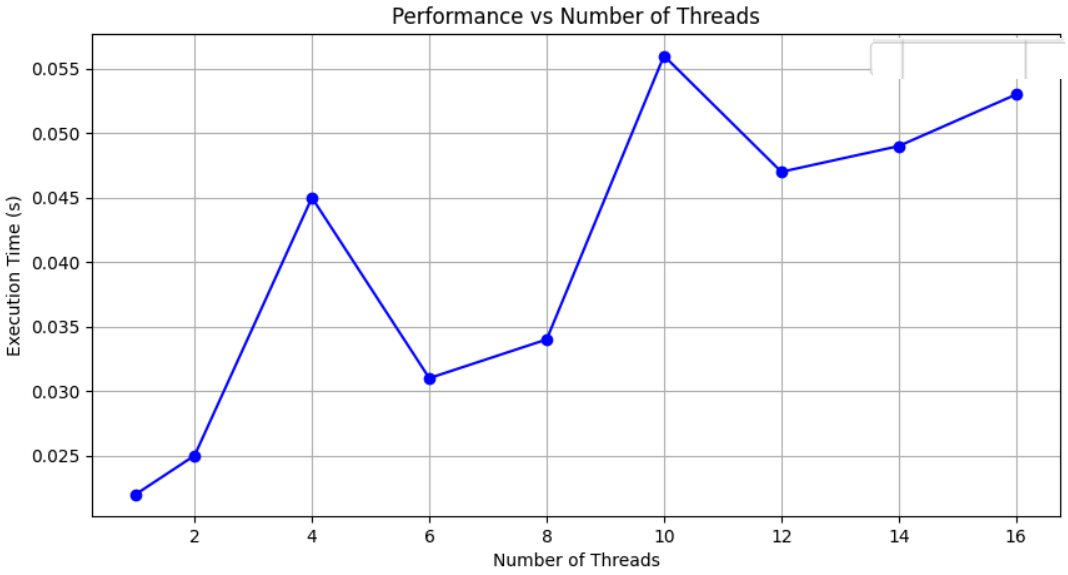
Static default:



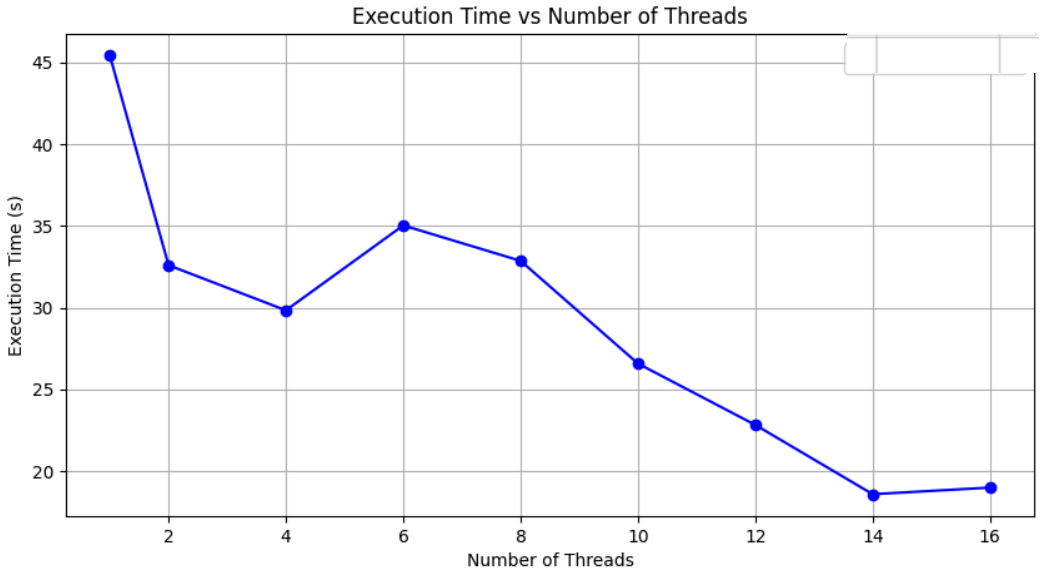


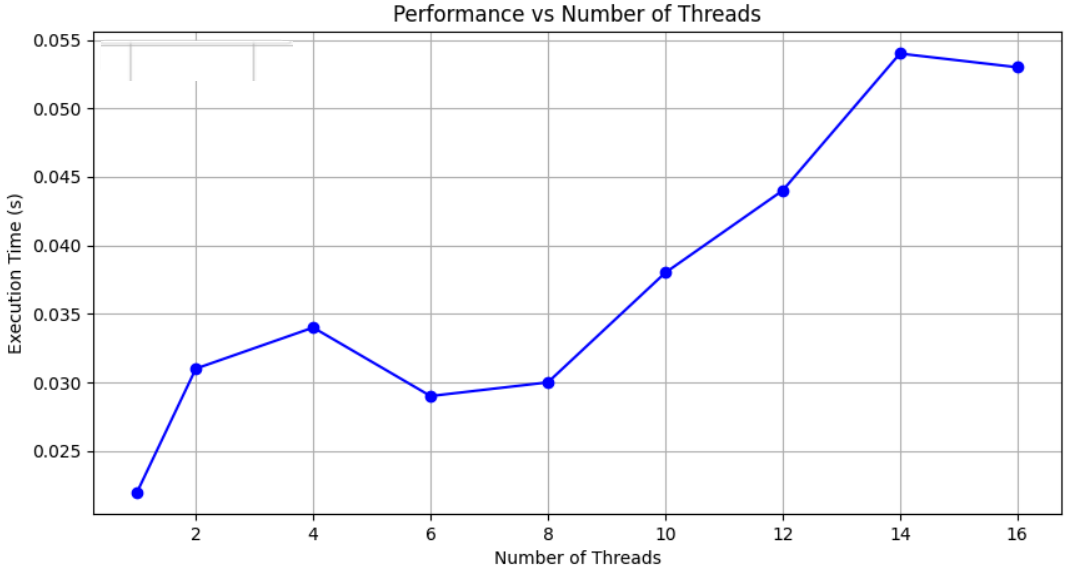
Static,10:


Execution Time vs Number of Threads
Execution Time (s)
45
40
35
30
25
20
2
4
6
8
10
12
14
16
Number of Threads

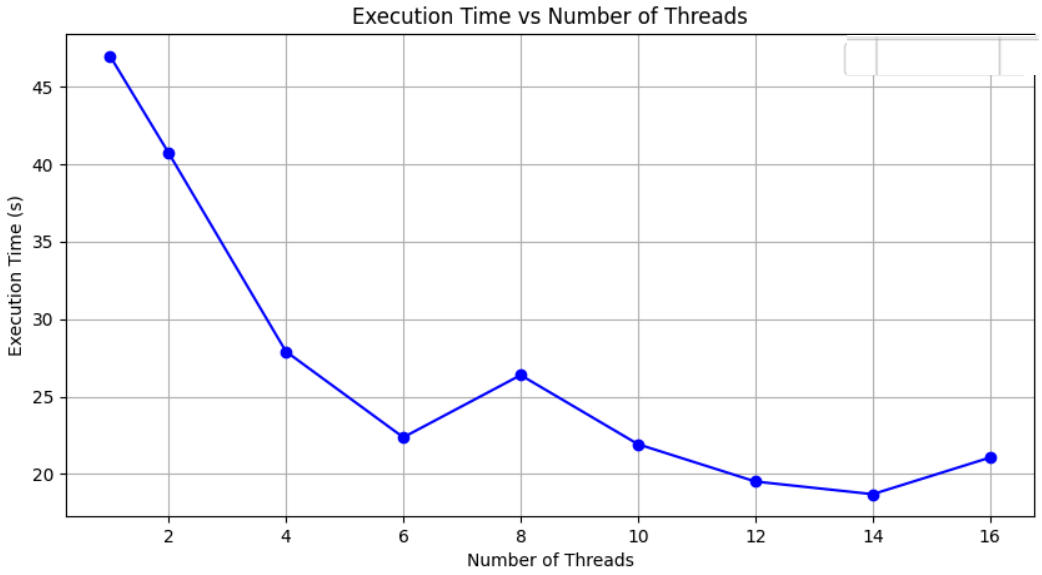

Performance vs Number of Threads
Execution Time (s)
0.055
0.050
0.045
0.040
0.035
0.030
0.025
2
4
6
8
10
12
14
16
Number of Threads

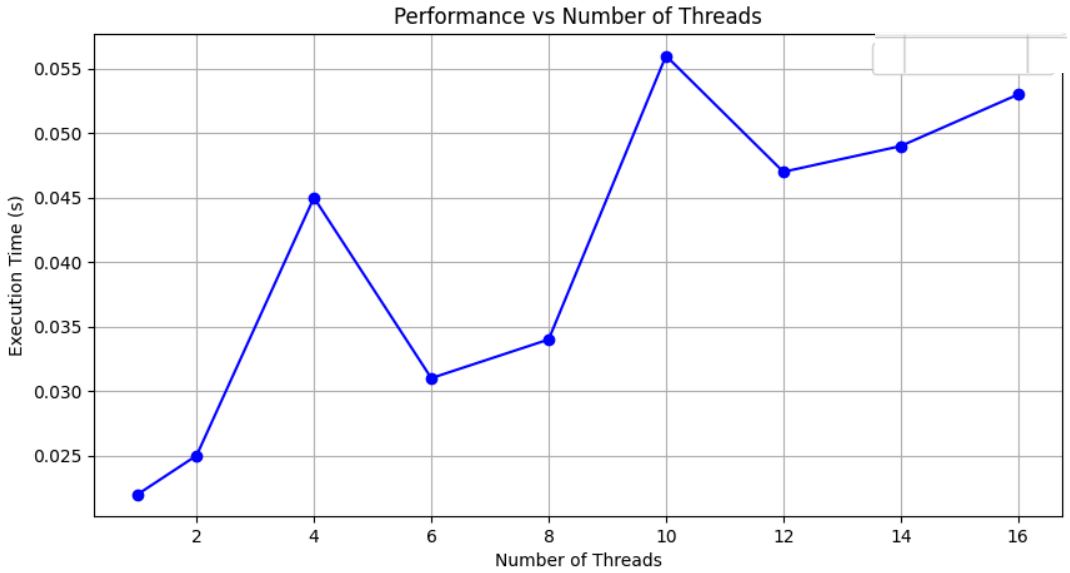
Dynamic default





Dynamic,10:


Execution Time vs Number of Threads
Execution Time (s)
45
40
35
30
25
20
2
4
6
8
10
12
14
16
Number of Threads


Performance vs Number of Threads
Execution Time (s)
0.055
0.050
0.045
0.040
0.035
0.030
0.025
2
4
6
8
10
12
14
16
Number of Threads

6. Scheduling Strategy Analysis

1. `static` (chunk size by default) At compilation time, the iteration space is distributed equally among threads using the `static` scheduling technique. When using 12 threads, the best execution time was reported at about 17.364 ms. This approach constantly performs well as the number of threads increases on a system with 8 physical cores. `static` scheduling works best when jobs are uniformly complex, which is typically the case for primality testing, therefore this is to be expected. Beyond 12 threads, performance somewhat degrades, most likely as a result of the overhead of handling more threads than there are physical cores. This expense may result in CPU time congestion and lower execution speed returns.

2. `dynamic` (chunk size by default) `dynamic` scheduling allocates iterations to threads at runtime, one chunk at a time, as opposed to `static` scheduling. When jobs have different execution times, this improves load balancing; nevertheless, the `dynamic` dispatching adds overhead `dynamic` scheduling performed more erratically in this trial. Although certain setups (such those with four or six threads) showed benefits, the overall advantages were less consistent.This implies that, particularly in a system with constrained memory bandwidth and a moderate RAM capacity of 3.9 GB, the additional scheduling overhead may be substantial in comparison to the computational cost of each iteration.

3. `static,10` This kind of `static` scheduling combines a finer granularity with a predictable distribution by establishing a fixed chunk size of 10.`static`,10 produced some of the top performance results on your 8-core computer, with 10 threads and a minimum execution time of 17.740 ms. Better usage of all cores without overloading any one thread is probably the cause of this increase. Smaller chunks cut down on idle time by allowing threads that finish early to continue on to fresh chunks. A chunk size of 10 seems to be a decent balance in this instance, but if it is too small, the number of context flips may grow.

4. `dynamic,10` This approach aims to minimize scheduling cost while preserving flexibility by combining `dynamic` load balancing with a moderate chunk size.`dynamic`,10 yielded competitive performance on your machine, especially at 6 and 14 threads, where the execution durations were nearly identical to the best recorded. This method's benefit is that it can adjust to changes in the computing load without incurring the fine-grained scheduling expenses associated with `dynamic` scheduling with extremely small chunks. Although the calculation per chunk is still significant enough to amortize the cost of `dynamic` assignment, it works particularly well in situations when workload distribution is unequal.