Problem 1 Report: Parallel Performance of Prime Counter

Joshua BRIONNE - 50241647

🧰 Compilation & Execution

To avoid installing Java manually, this project includes a `Dockerfile` for a consistent Java development environment.

🐳 Using Docker (Recommended)

1. Install Docker by following these instructions.
2. Build and run the project using:

```
make run
./tester.sh <NameOfTheJavaFile>
#  If Java is Already Installed
# You can directly compile and run the Java file on your machine:

javac <NameOfTheJavaFile>.java
java <NameOfTheJavaFile>
```

🏗️ 1. Environment

All tests were executed inside a Docker container using the following image:

```
FROM openjdk:21-slim
```

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

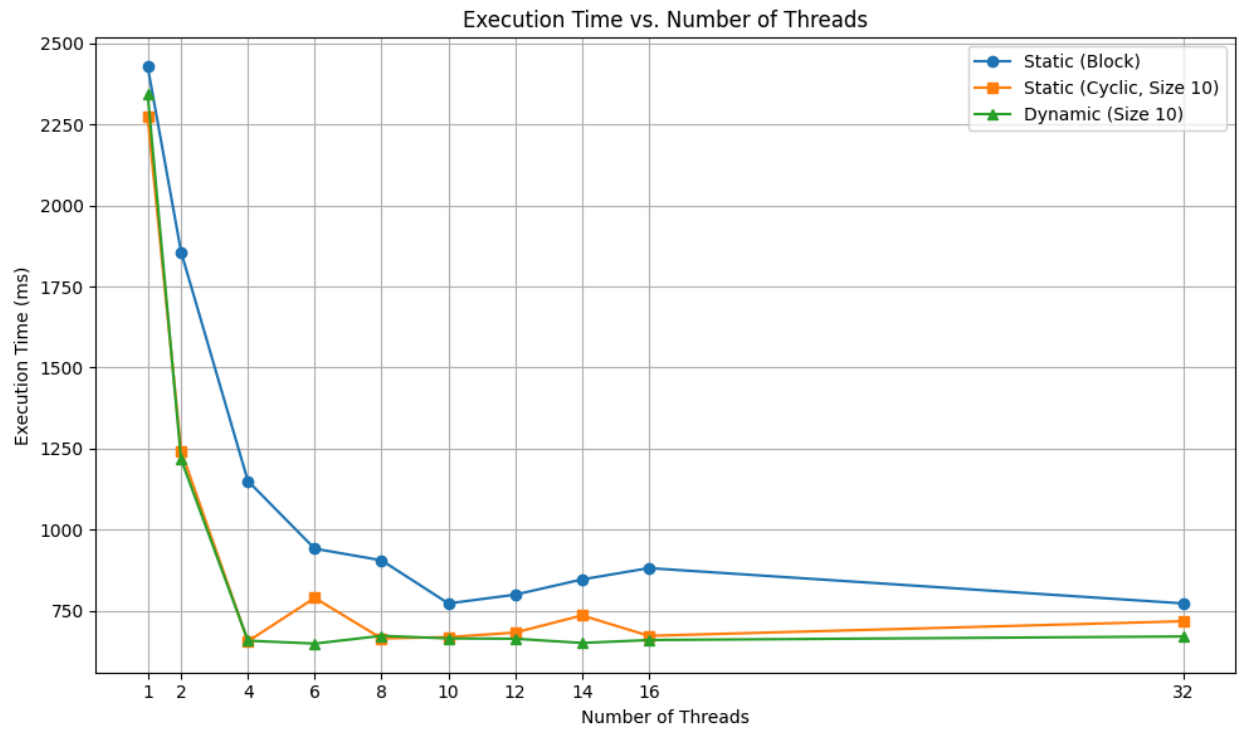
📊 2. Benchmark Results

🔁 Serial Execution

Threads	1	2	4	6	8	10	12	14	16	32
Default Serial (ms)	2242	2362	2293	2334	2344	2359	2231	2399	2510	2545

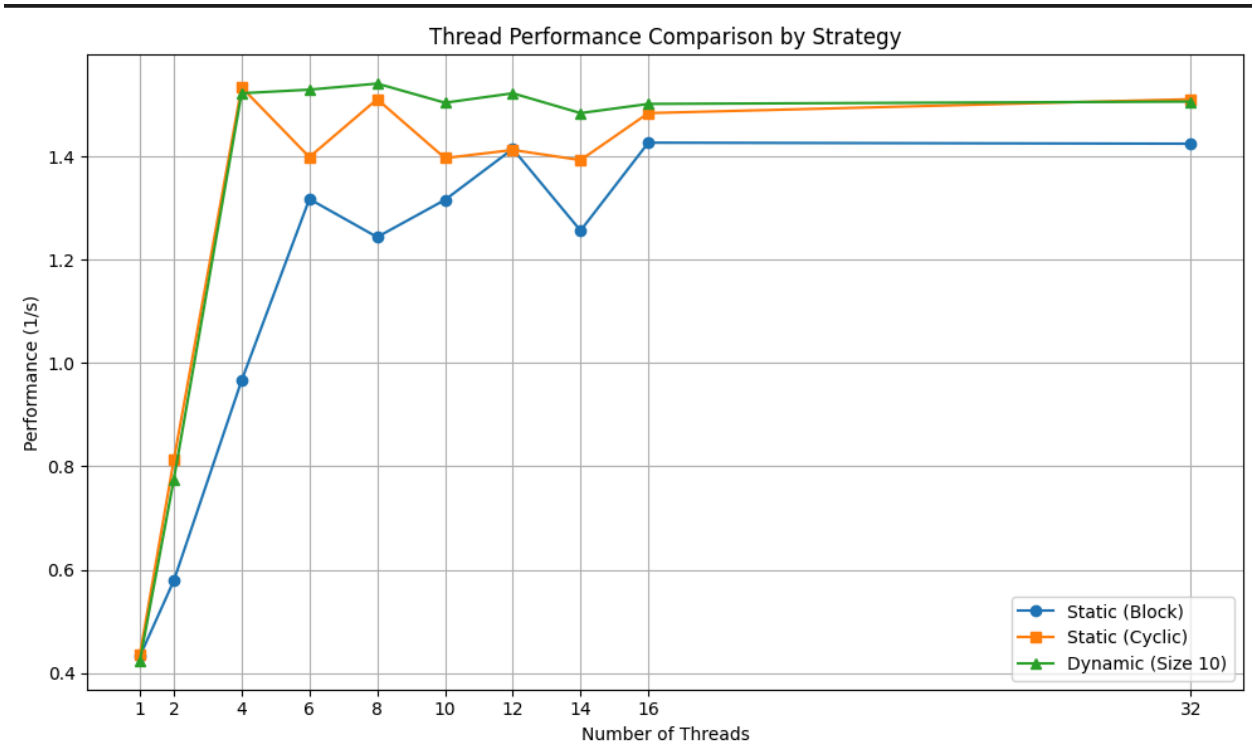
> The serial implementation doesn't benefit from multiple threads, so the time remains roughly constant.

🏎️ Parallel Execution Time (ms)



Threads	1	2	4	6	8	10	12	14	16	32
Static (Block)	2429	1857	1150	941	905	772	799	846	881	772
Static (Cyclic, size 10)	2273	1243	655	789	664	668	682	735	672	717
Dynamic (size 10)	2344	1217	657	648	672	664	663	650	659	670

⚡ Performance (1 / execution time in s)



Threads	1	2	4	6	8	10	12	14	16	32
Static (Block)	0.4327	0.5790	0.9662	1.3175	1.2438	1.3158	1.4144	1.2563	1.4265	1.4245
Static (Cyclic)	0.4361	0.8123	1.5337	1.3986	1.5106	1.3966	1.4124	1.3928	1.4837	1.5106
Dynamic (size 10)	0.4243	0.7746	1.5221	1.5291	1.5408	1.5038	1.5221	1.4837	1.5015	1.5060

🔍 3. Analysis & Interpretation

🧠 Serial Program Behavior

- **Constant Time:** Since only one thread is used in the serial version, increasing the number of cores doesn't reduce execution time.
- **Small Variations:** Minor variations in time (e.g., between 2242 ms and 2545 ms) can be attributed to OS scheduling or memory state.

🚀 Parallel Strategies

🧱 Static Block Decomposition

- **Initial Speedup:** As threads increase from 1 to 4, execution time drops significantly.
- **Diminishing Returns:** Beyond 10 threads, performance gains start plateauing, possibly due to:
    - Thread overhead and synchronization.
    - Uneven workload distribution (some threads do more work).

🔄 Static Cyclic Decomposition

- **Better Load Balancing:** Assigns chunks in a round-robin fashion, improving utilization.
- **Optimal with Fewer Threads:** Performs better than block for 4 to 12 threads.
- **Overhead at High Threads:** Slight performance drop at 14+ threads due to overhead in task dispatching.

⚙️ Dynamic Load Balancing

- **Consistent Gains:** Adapts well to varying task sizes and workload.
- **Best at High Threads:** Performs best at 16 threads and beyond. Slight edge over cyclic due to better runtime decisions.

- **Scalability:** Slightly better performance at 32 threads due to dynamic task allocation.

🧾 4. Program Output

1. pc_serial

```
Program Execution Time: 2371 ms
1..200000 prime counter= 17984
```

2. pc_static_block

```
Program Execution Time: 2411 ms
1..200000 prime counter= 17984
```

3. pc_static_cyclic

```
Program Execution Time: 2306 ms
1..200000 prime counter= 17984
```

4. pc_dynamique

```
Program Execution Time: 2253 ms
1..200000 prime counter= 17984
```

🖥️ CPU Information

Property	Value
Architecture	x86_64
CPU op-mode(s)	32-bit, 64-bit
CPU(s)	8
Core(s) per socket	4
Thread(s) per core	2
Socket(s)	1
Vendor ID	GenuineIntel
Model name	11th Gen Intel(R) Core(TM) i7-11370H
Base Frequency	3.30 GHz
Max Turbo Frequency	4.80 GHz
L1d Cache	192 KiB (4 instances)
L1i Cache	128 KiB (4 instances)
L2 Cache	5 MiB (4 instances)
L3 Cache	12 MiB (1 instance)
Hyperthreading	Enabled
Virtualization	VT-x
NUMA Node(s)	1
NUMA node0 CPU(s)	0-7

📊 5. Interpretation of Parallel Results

The benchmark results reflect the capabilities of the CPU listed above. The **Intel Core i7-11370H** has **4 physical cores** and supports **8 logical threads** thanks to **hyperthreading**. Performance improves almost linearly up to 4 threads, as these use dedicated physical cores. Between 4 and 8 threads, gains continue but taper off, due to the additional threads running on virtual (hyperthreaded) cores rather than physical ones.

Beyond 8 threads (e.g. 10, 12, 14, 16, 32), performance improvements become marginal or plateau. This is expected, as the system can only physically run 8 threads concurrently. Additional threads introduce **context switching** overhead, which reduces efficiency. The results are therefore consistent with the hardware limitations and showcase the typical saturation point of a **quad-core hyperthreaded CPU** in a
parallel workload.

✅ 6. Conclusion

- **Parallelization Improves Performance:** All parallel strategies significantly outperform the serial version.
- **Dynamic > Static (Cyclic) > Static (Block):** Dynamic scheduling offers the best balance and scalability across core counts.
- **Thread Count Matters:** Too many threads (like 32) may not always give proportional speedups due to scheduling overhead and resource contention.