

# HiPerGator Performance Analysis with Matrix Multiplication

Anders Jensen

4-22-2024

Dr. El Aarag

CSCI 301

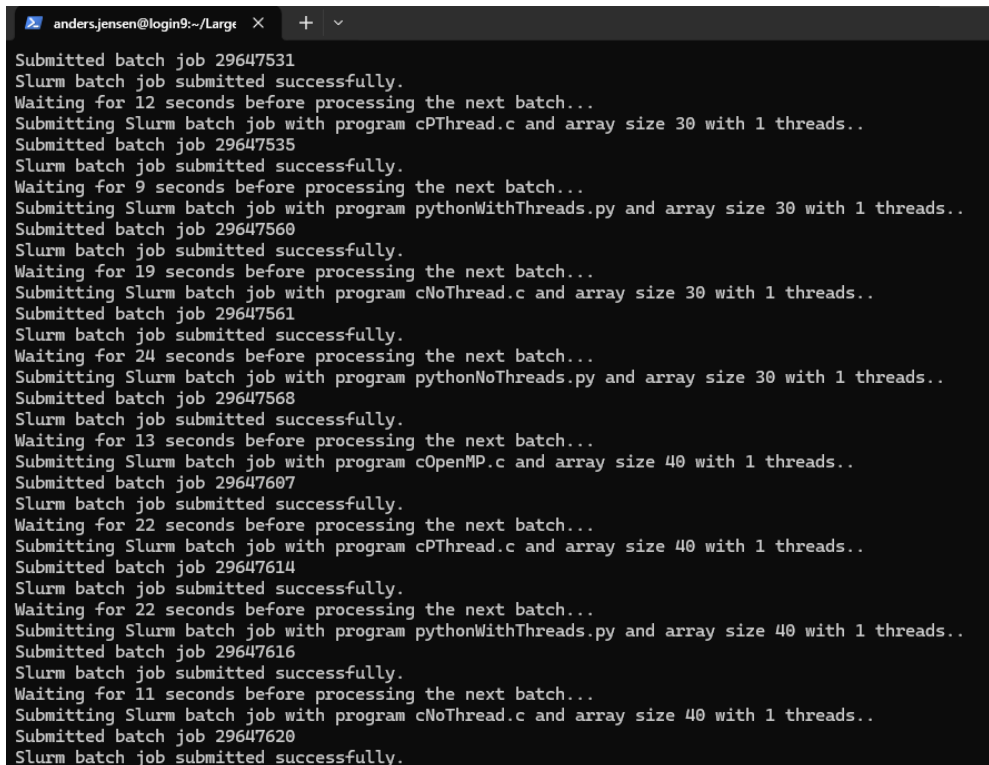
1. Introduction: Matrix multiplication is a fundamental operation in various scientific and computational applications, often demanding high computational resources. This study aims to evaluate the performance of matrix multiplication programs implemented in different programming languages and paradigms on a supercomputer server. By analyzing execution times and thread utilization, I seek to provide insights into the comparative efficiency of these programs in high-performance computing environments.

2. Methodology:

2.1 Experimental Setup:

The experiments were conducted on a supercomputer server equipped with multi-core processors.

Each program was executed multiple times to ensure statistical reliability. See batch job submissions to Slurm in Figure 1 and 2 below:



```
anders.jensen@login9:~/Large X + v
Submitted batch job 29647531
Slurm batch job submitted successfully.
Waiting for 12 seconds before processing the next batch...
Submitting Slurm batch job with program cPThread.c and array size 30 with 1 threads..
Submitted batch job 29647535
Slurm batch job submitted successfully.
Waiting for 9 seconds before processing the next batch...
Submitting Slurm batch job with program pythonWithThreads.py and array size 30 with 1 threads..
Submitted batch job 29647560
Slurm batch job submitted successfully.
Waiting for 19 seconds before processing the next batch...
Submitting Slurm batch job with program cNoThread.c and array size 30 with 1 threads..
Submitted batch job 29647561
Slurm batch job submitted successfully.
Waiting for 24 seconds before processing the next batch...
Submitting Slurm batch job with program pythonNoThreads.py and array size 30 with 1 threads..
Submitted batch job 29647568
Slurm batch job submitted successfully.
Waiting for 13 seconds before processing the next batch...
Submitting Slurm batch job with program cOpenMP.c and array size 40 with 1 threads..
Submitted batch job 29647607
Slurm batch job submitted successfully.
Waiting for 22 seconds before processing the next batch...
Submitting Slurm batch job with program cPThread.c and array size 40 with 1 threads..
Submitted batch job 29647614
Slurm batch job submitted successfully.
Waiting for 22 seconds before processing the next batch...
Submitting Slurm batch job with program pythonWithThreads.py and array size 40 with 1 threads..
Submitted batch job 29647616
Slurm batch job submitted successfully.
Waiting for 11 seconds before processing the next batch...
Submitting Slurm batch job with program cNoThread.c and array size 40 with 1 threads..
Submitted batch job 29647620
Slurm batch job submitted successfully.
```

Figure 1

29647774.ba+	batch	csci301	1	COMPLETED	0:0
29647794	job28 hpg-defau+	csci301	1	FAILED	1:0
29647794.ba+	batch	csci301	1	FAILED	1:0
29647825	job29 hpg-defau+	csci301	1	COMPLETED	0:0
29647825.ba+	batch	csci301	1	COMPLETED	0:0
29647831	job30 hpg-defau+	csci301	1	FAILED	1:0
29647831.ba+	batch	csci301	1	FAILED	1:0
29647832	job31 hpg-defau+	csci301	1	COMPLETED	0:0
29647832.ba+	batch	csci301	1	COMPLETED	0:0
29647838	job32 hpg2-comp+	csci301	1	COMPLETED	0:0
29647838.ba+	batch	csci301	1	COMPLETED	0:0
29647952	job33 hpg2-comp+	csci301	1	FAILED	1:0
29647952.ba+	batch	csci301	1	FAILED	1:0
29647968	job34 hpg2-comp+	csci301	1	COMPLETED	0:0
29647968.ba+	batch	csci301	1	COMPLETED	0:0
29648020	job35 hpg2-comp+	csci301	1	FAILED	1:0
29648020.ba+	batch	csci301	1	FAILED	1:0
29648047	job36 hpg-defau+	csci301	20	PENDING	0:0
29648083	job37 hpg-defau+	csci301	20	PENDING	0:0
29648085	job38 hpg-defau+	csci301	20	PENDING	0:0
29648102	job39 hpg-defau+	csci301	20	PENDING	0:0
29648123	job40 hpg-defau+	csci301	20	PENDING	0:0
29648170	job41 hpg-defau+	csci301	20	PENDING	0:0
29648273	job42 hpg-defau+	csci301	20	PENDING	0:0
29648275	job43 hpg-defau+	csci301	20	PENDING	0:0
29648282	job44 hpg-defau+	csci301	20	PENDING	0:0
29648306	job45 hpg-defau+	csci301	20	PENDING	0:0
29648315	job46 hpg-defau+	csci301	20	PENDING	0:0

Figure 2

## 2.2 Program Implementation:

Matrix multiplication programs were implemented in C (utilizing OpenMP, PThread, and no threading) and Python (with and without threading).

The programs were optimized for parallel execution to leverage the computational capabilities of the supercomputer.

## 2.3 Job Configurations:

Various job configurations were considered, including different matrix dimensions and thread utilization settings.

Matrix dimensions ranged from 10x10 to 2000x2000, with increments at each step. All matrices were filled with 1's to formulate consistent and accurate comparisons, leading to robust conclusions. Matrices of greater dimensions were scheduled as batch jobs but had not completed in the time allotted for unknown reasons.

Thread utilization settings varied from single-threaded execution to multi-threaded execution with up to three threads.

## 2.4 Data Collection:

Execution times for each job configuration were measured and recorded.

Statistical analysis techniques, including mean, median, and standard deviation, were employed to analyze the collected data.

### 3. Results and Discussion:

#### 3.1 Performance Comparison:

Across all job configurations, C OpenMP demonstrated the lowest execution times, followed by C PThread and C NoThread, while Python programs exhibited comparatively higher execution times.

For a 1000x1000 matrix multiplication job, C OpenMP recorded an average execution time of 0.328771 seconds, while Python NoThreads took 110.3648 seconds, highlighting the performance disparity between the two languages.

Python programs with threading (WithThreads) showcased improved performance compared to their non-threaded counterparts (NoThreads) but remained slower than C implementations.

#### 3.2 Effect of Threading:

In C programs, the impact of threading on performance was investigated by varying the number of threads.

OpenMP and PThread implementations exhibited performance gains with increased thread utilization, up to a certain threshold, beyond which the benefits diminished.

For instance, in a 100x100 matrix multiplication job, C OpenMP with three threads recorded an average execution time of 0.004046 seconds, compared to 0.051193 seconds with single-threaded execution.

Figures 3, 4, 5, and 6 below illustrate the clear differences between the performance of each program along with specified number of cores.

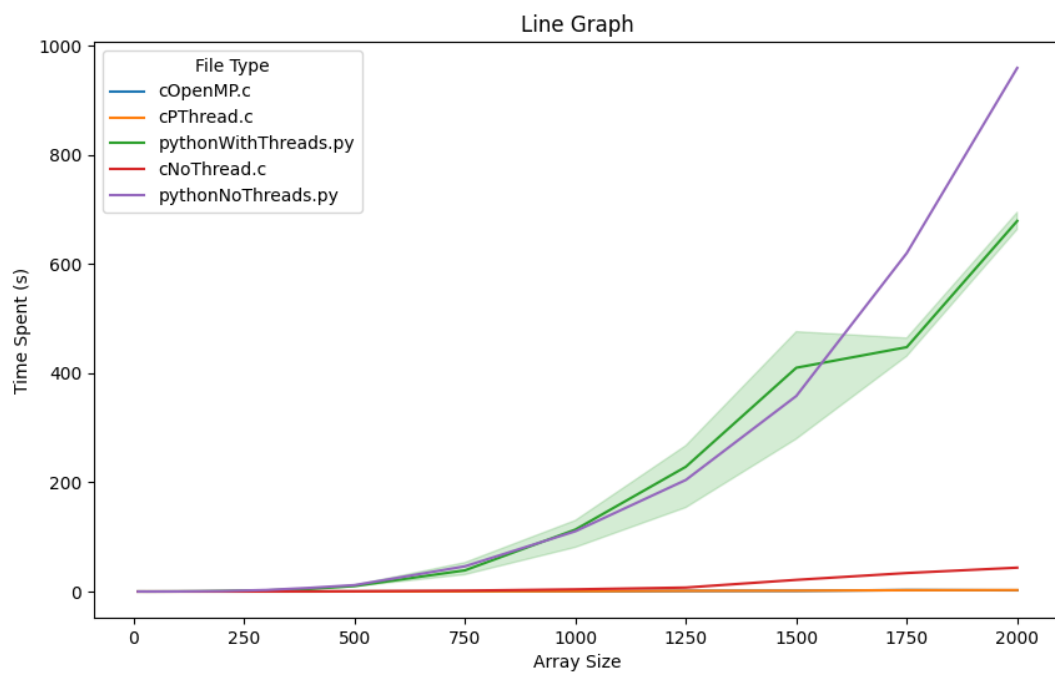


Figure 3

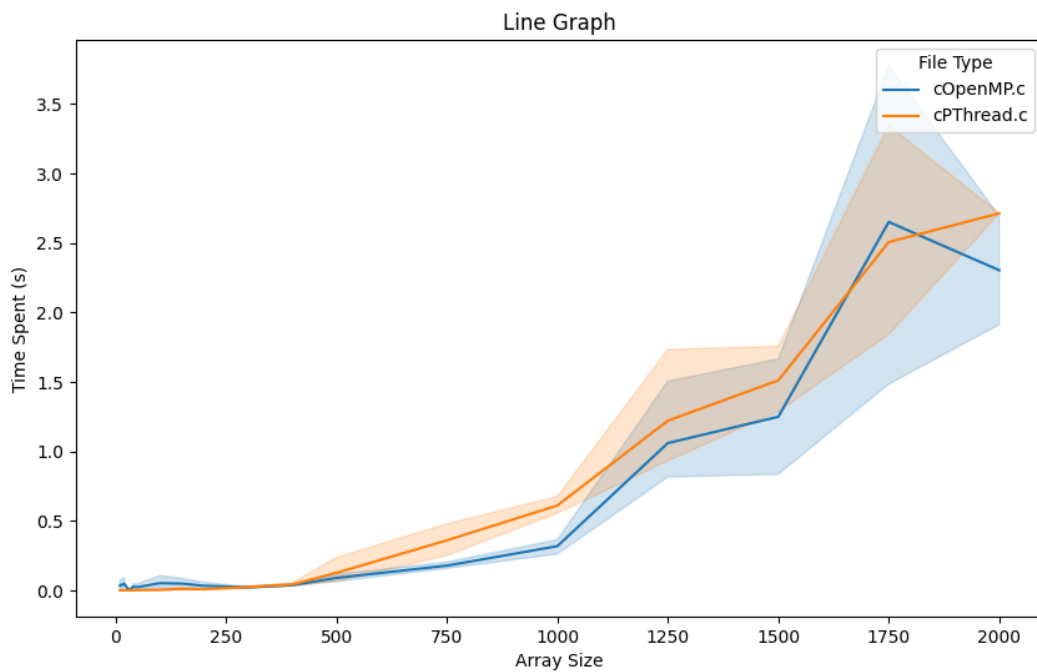


Figure 4

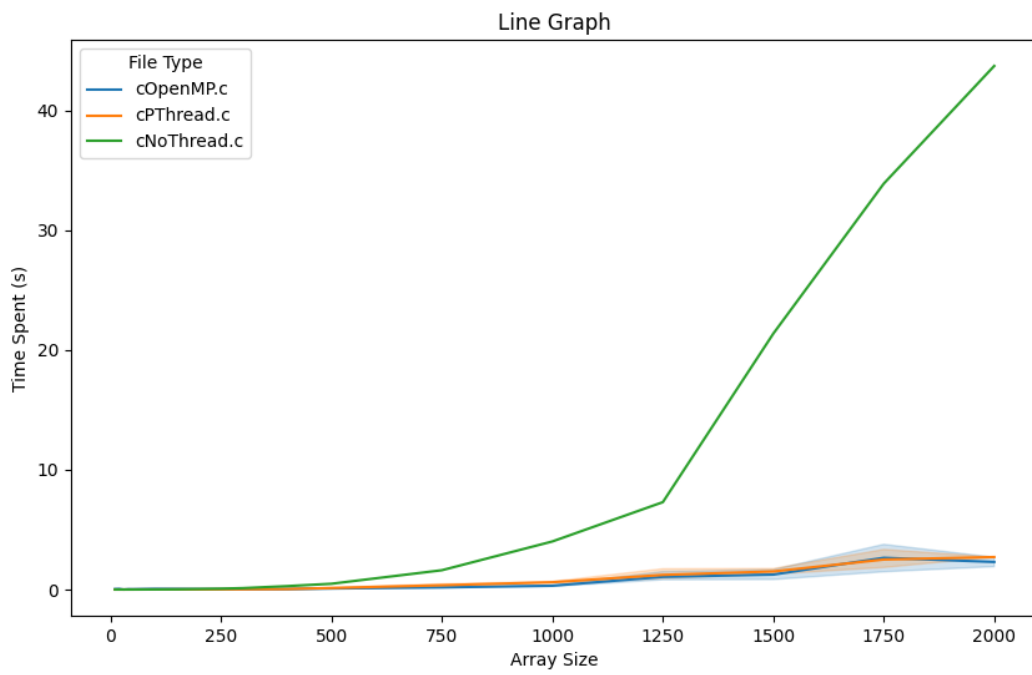


Figure 5

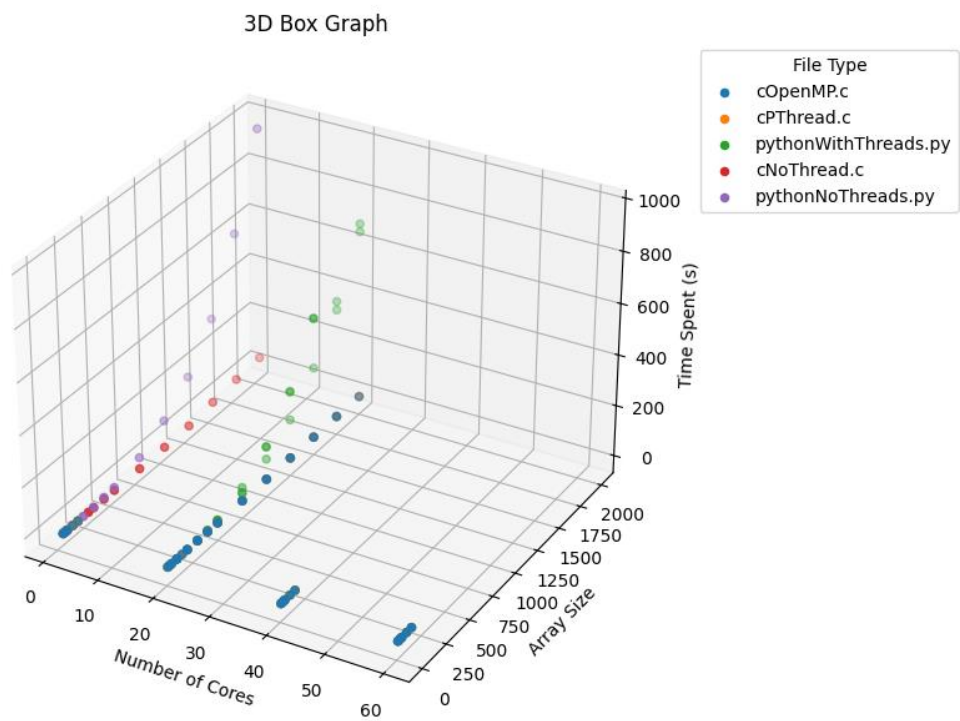


Figure 6

4. Conclusion: This study highlights the significance of programming paradigms and threading strategies in optimizing the performance of matrix multiplication programs on supercomputer servers. While C implementations with OpenMP demonstrated superior performance, Python programs showcased the inherent overhead associated with interpreted languages. The findings underscore the importance of leveraging parallelism and optimizing thread utilization to enhance the efficiency of matrix multiplication algorithms in high-performance computing environments.