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:

A screenshot of a computer program

Description automatically generated

Figure 1

A screenshot of a computer

Description automatically generated

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.

A graph with lines and numbers

Description automatically generated

Figure 3

A graph with blue and orange lines

Description automatically generated

Figure 4

A graph with green line and orange line

Description automatically generated

Figure 5

A graph with different colored dots

Description automatically generated

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.