CS 301
High-Performance Computing

Lab2 - Performance Analysis and Optimization of Numerical and Sorting Algorithms

Pratham Patel (202201485)
Ramkumar Patel (202201509)

Contents

1 Introduction

This report presents performance analysis and optimization of numerical and sorting algorithms on two different machines: the Lab 207 machine and the HPC Cluster. The three sub-problems include:

- Problem A-2: Calculation of Pi using Series
- Problem B-2: Multiplication of Two Vectors Followed by Summation
- Problem C-2: Sorting - Merge Sort

The objective of this assignment is to implement these algorithms, profile their performance, and analyze them for various inputs and on different machines.

2 Hardware Details

To compare the performance of different architectures, we analyzed two machines: Lab 207 PC and HPC Cluster. The key specifications are summarized in Table 1.

Specification	**Lab 207 PC**	**HPC Cluster**
Architecture	x86_64	x86_64
CPU Model	12th Gen Intel Core i5-12500	Intel Xeon E5-2620 v3 @ 2.40GHz
Clock Speed (GHz)	0.8 - 4.6	2.4
Cores / Threads	6 / 12	12 / 24
L1 Cache (KB)	288 (6 instances)	32 (data + instruction)
L2 Cache (KB)	7.5 MiB (6 instances)	256 KiB
L3 Cache (MB)	18	15
Memory (GB)	8 (DDR4)	64 (DDR4)
Memory Channels	2	2 (NUMA: 2 nodes)
Hyper-Threading	Yes	No

Table 1: Comparison of Hardware Architectures

3 Sub-problem A-2: Calculation of Pi using Series

3.1 Context

- **Brief Description:** This problem involves calculating the value of Pi using a series approximation. The algorithm iterates over the terms in the series to compute an approximation for Pi.
- **Complexity of the Algorithm:** The serial time complexity is $O(N)$, where N is the number of terms in the series.
- **Profiling Information:** Profiling information using gprof shows runtime behavior.
- **Compute-to-Memory Access Ratio:** This problem is compute-bound, as it relies heavily on arithmetic operations with minimal memory access.
- **Memory vs Compute-bound:** Given that the algorithm is based primarily on computation and requires only minimal memory access, it is classified as compute-bound.
- **Optimization Strategy:** Optimization focuses on enhancing efficiency by using loop unrolling, double precision for more accurate results, and exploring vectorization techniques to speed up computation.

3.2 Input Parameters and Output

- **Input:** Large value of N for Pi calculation.
- **Output:** Approximate value of Pi.
- **Accuracy Check:** Compare the result with the exact value of π.

3.3 Problem Size vs Time (Serial) Curve

- Plot of problem size versus time for various values of n.

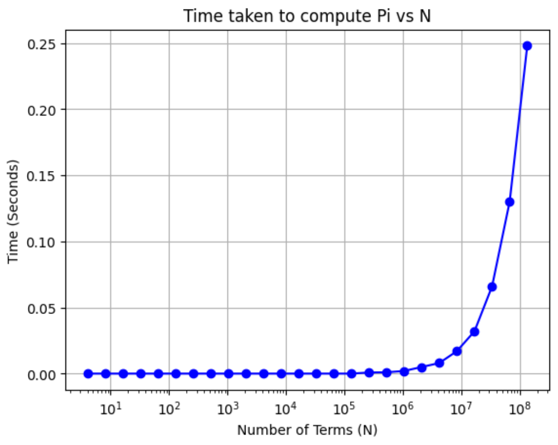

Figure 1: Throughput vs Problem Size for Pi Computation on Lab PC

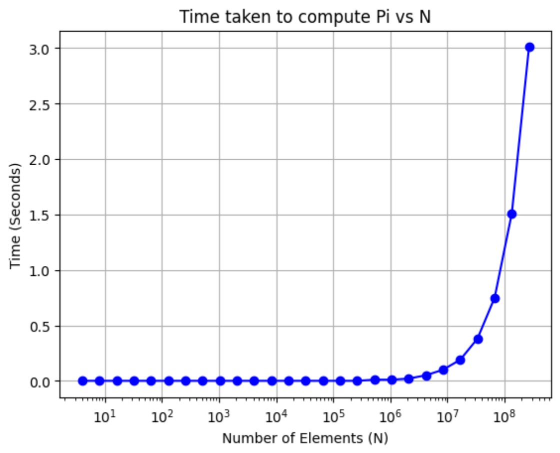

Figure 2: Throughput vs Problem Size for Pi Computation on HPC Cluster

- **Observations:** As n increases, the time taken by the algorithm increases linearly.

4 Sub-problem B-2: Multiplication of Two Vectors Followed by Summation

4.1 Context

- **Brief Description:** This problem involves computing $[A(i) * B(i)] + [A(i) + B(i)]$ for all elements in vectors A and B.
- **Complexity of the Algorithm:** The serial time complexity is $O(N)$, where N is the size of the vectors.
- **Profiling Information:** Profiling with gprof shows time spent in multiplication and addition operations.
- **Compute-to-Memory Access Ratio:** The compute-to-memory access ratio is moderate, as the algorithm involves both arithmetic operations and memory access.
- **Memory vs Compute-bound:** This problem is compute-bound, as it involves performing arithmetic calculations over each element of the vectors.
- **Optimization Strategy:** Optimization can be achieved by using SIMD (Single Instruction Multiple Data) for parallelization, avoiding redundant calculations, and ensuring efficient memory access using cache-friendly operations.

4.2 Input Parameters and Output

- **Input:** Two vectors A and B of length n.
- **Output:** A new vector C where $C[i] = (A[i] * B[i]) + A[i] + B[i]$.
- **Accuracy Check:** Check if the sum matches the expected output for known inputs.

4.3 Problem Size vs Time (Serial) Curve

- Plot the problem size vs time for varying vector sizes.

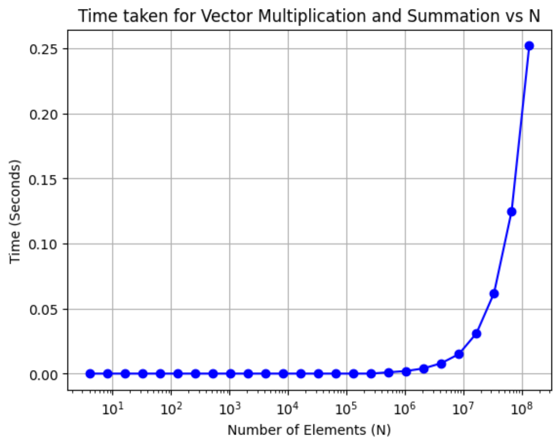

Figure 3: Throughput vs Problem Size for Vector Multiplication and Addition on Lab PC

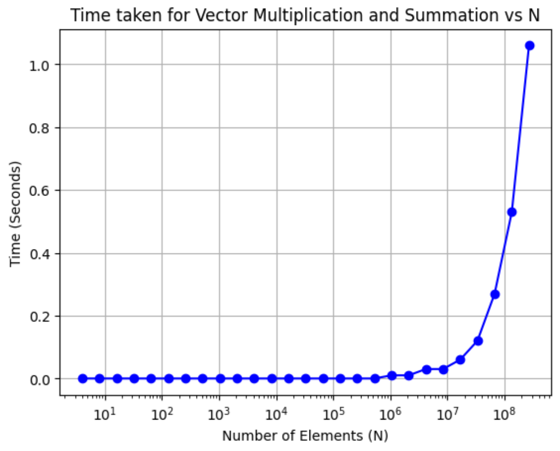

Figure 4: Throughput vs Problem Size for Vector Multiplication and Addition on HPC Cluster

- **Observations:** Larger vectors lead to higher runtime due to more iterations.

5 Sub-problem C-2: Sorting - Merge Sort

5.1 Context

- **Brief Description:** This problem involves sorting an array using the Merge Sort algorithm and analyzing its performance.
- **Complexity of the Algorithm:** The serial time complexity is $O(N \log N)$, where N is the size of the array.

- **Profiling Information:** Profiling with gprof highlights recursive function calls in the Merge Sort algorithm.
- **Compute-to-Memory Access Ratio:** The compute-to-memory access ratio is moderate, as Merge Sort involves both recursive calls and frequent data movement.
- **Memory vs Compute-bound:** The problem is memory-bound due to recursive calls and the frequent data movement during the merging process.
- **Optimization Strategy:** Optimization can be achieved by using iterative Merge Sort to avoid stack overhead, implementing in-place merging where possible, and optimizing memory usage by merging smaller subarrays first.

5.2 Input Parameters and Output

- **Input:** An array of size n to be sorted.
- **Output:** The sorted array.
- **Accuracy Check:** Ensure that the output array is sorted correctly.

5.3 Problem Size vs Time (Serial) Curve

- Plot the problem size vs time for varying array sizes.

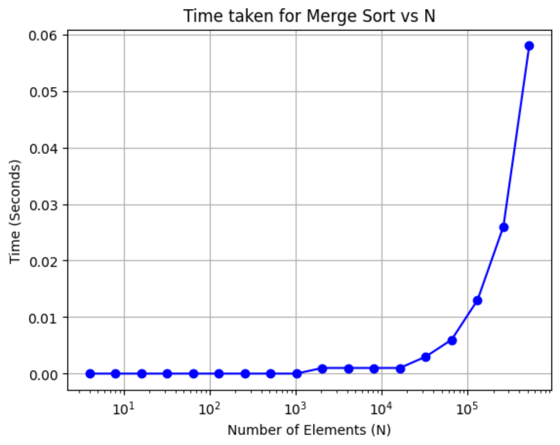

Figure 5: Throughput vs Problem Size for Merge Sort on Lab PC

- **Observations:** The algorithm performs well with increasing problem sizes but has performance bottlenecks with very large arrays.

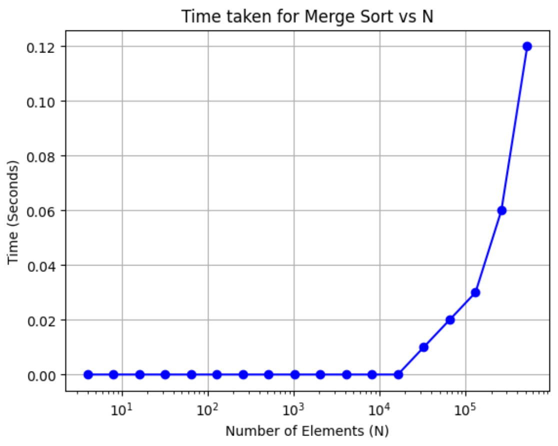

Figure 6: Throughput vs Problem Size for Merge Sort on HPC Cluster

6 Conclusion

In this report, we have implemented and profiled several algorithms, analyzing their performance in terms of time complexity, memory usage, and computational efficiency. Based on profiling results, we have provided insights into the optimization strategies that can improve algorithmic performance on different hardware platforms.