CS 301
High-Performance Computing

Lab4 - Performance Analysis of Parallel Computing Implementations

Pratham Patel (202201485)
Ramkumar Patel (202201509)

Contents

1 Introduction

This report presents the performance of parallel computing implementations for solving computational problems. The primary objective is to analyze execution time and speedup across different problem sizes and hardware configurations. The study covers three computational problems:

- 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 parallely, 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:**
  - **Serial:** $O(N)$, where N is the number of terms in the series.
  - **Parallel:** $O(N/P)$, where P is the number of processing units.
- **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.
- **Optimization Strategy:** Optimization focuses on enhancing efficiency by using loop unrolling, vectorized operations, and OpenMP for parallelization.

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 (Parallel) Curve

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

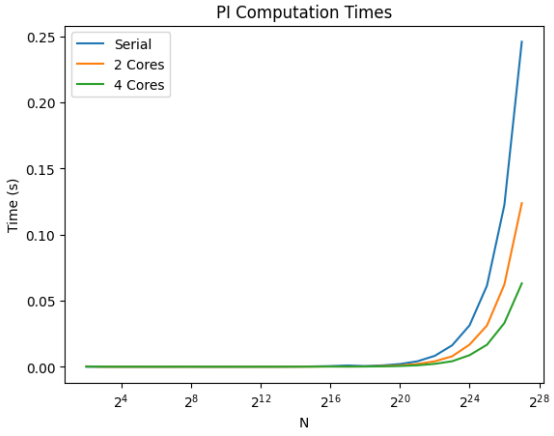

Figure 1: Time vs Problem Size for Pi Computation on Lab PC

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

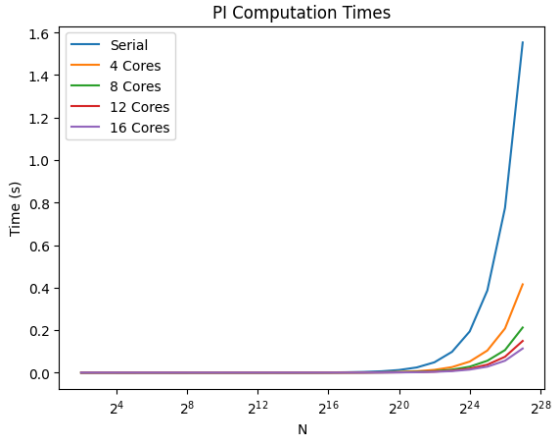

Figure 2: Time vs Problem Size for Pi Computation on HPC Cluster

3.4 Problem Size vs Speedup (for different cores) Curve

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

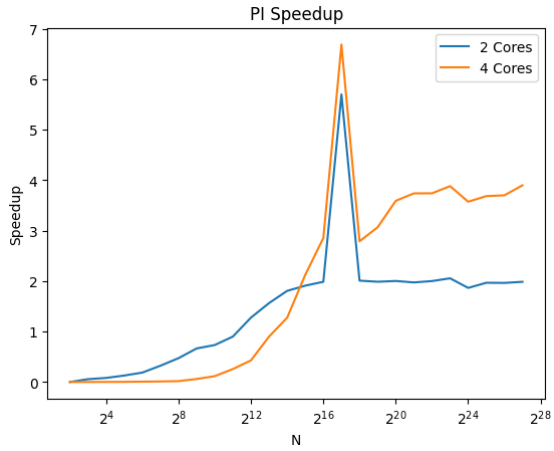

Figure 3: Speedup vs Problem Size for Pi Computation on Lab PC

- **Observations:** As cores increases, the speedup increases, but for small problem size it shows different behavior due parallel overhead.

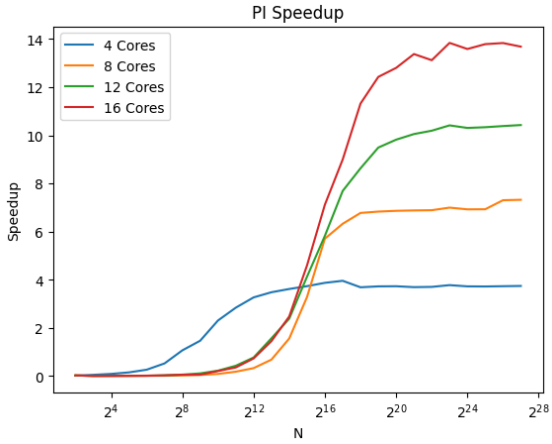

Figure 4: Throughput vs Problem Size for Pi Computation on HPC Cluster

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:**
  - **Serial:** $O(N)$, where N is the size of the vectors.
  - **Parallel:** $O(N/P)$ using OpenMP or SIMD-based vectorization.
- **Profiling Information:** Profiling with gprof shows time spent in multiplication and addition operations.
- **Compute-to-Memory Access Ratio:** The problem is memory-bound, as frequent memory accesses affect performance.
- **Optimization Strategy:** Optimization can be achieved by using SIMD (Single Instruction Multiple Data) for parallelization and improving memory locality to enhance cache efficiency.

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 (Parallel) Curve

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

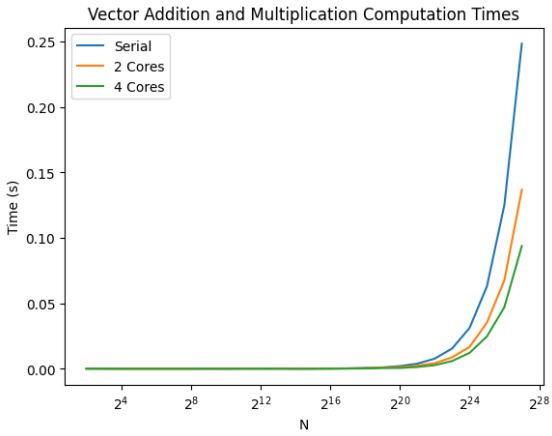

Figure 5: Time vs Problem Size for Vector Addition and Multiplication on Lab PC

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

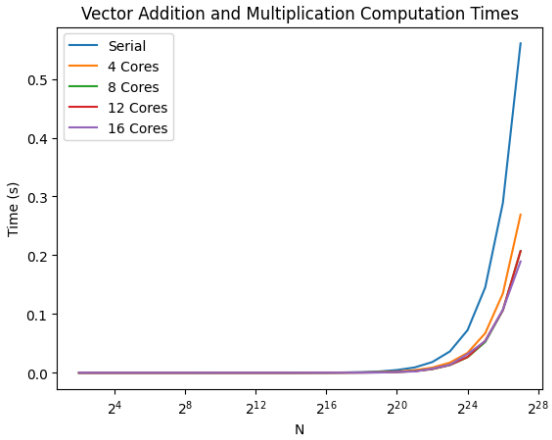

Figure 6: Time vs Problem Size for Vector Addition and Multiplication on HPC Cluster

4.4 Problem Size vs Speedup (for different cores) Curve

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

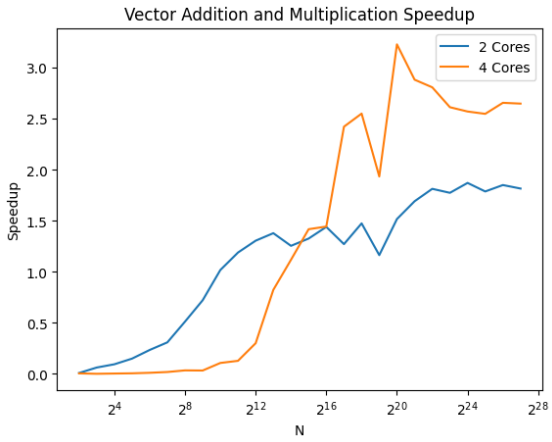

Figure 7: Speedup vs Problem Size for Vector Addition and Multiplication on Lab PC

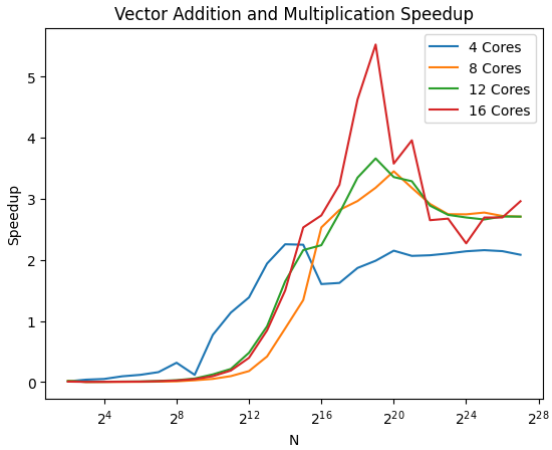

Figure 8: Throughput vs Problem Size for Vector Addition and Multiplication on HPC Cluster

- **Observations:** As cores increases, the speedup increases, but for small problem size it shows different behavior due parallel overhead.

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:**

- **Serial:** $O(N \log N)$.
- **Parallel:** $O(N \log N/P)$ using parallel merge sort with OpenMP or MPI.

- **Profiling Information:** Profiling with gprof highlights recursive function calls in the Merge Sort algorithm.
- **Compute-to-Memory Access Ratio:** The problem is memory-bound for large inputs, as frequent data movement impacts efficiency.
- **Optimization Strategy:** Optimization can be achieved by using parallel merge operations, reducing memory transfers, and implementing in-place merging techniques.

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 (Parallel) Curve

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

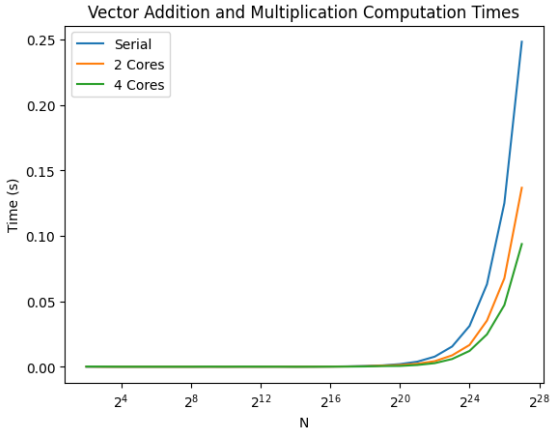

Figure 9: Time vs Problem Size for Merge Sort on Lab PC

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

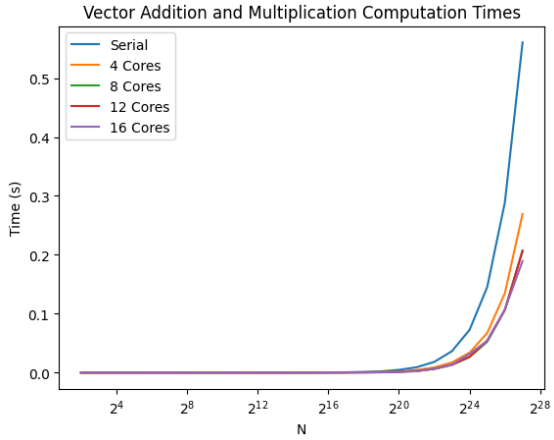

Figure 10: Time vs Problem Size for Merge Sort on HPC Cluster

5.4 Problem Size vs Speedup (for different cores) Curve

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

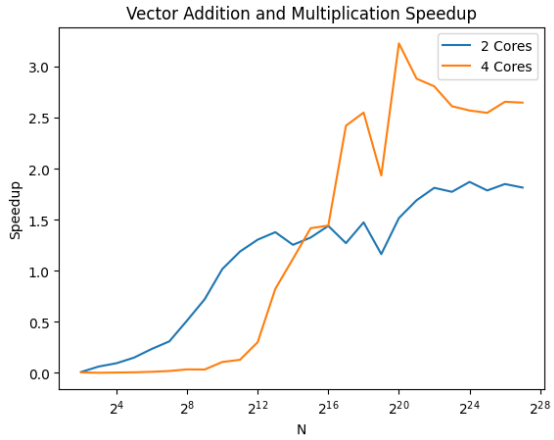

Figure 11: Speedup vs Problem Size for Merge Sort on Lab PC

- **Observations:** As cores increases, the speedup increases, but for small problem size it shows different behavior due parallel overhead.

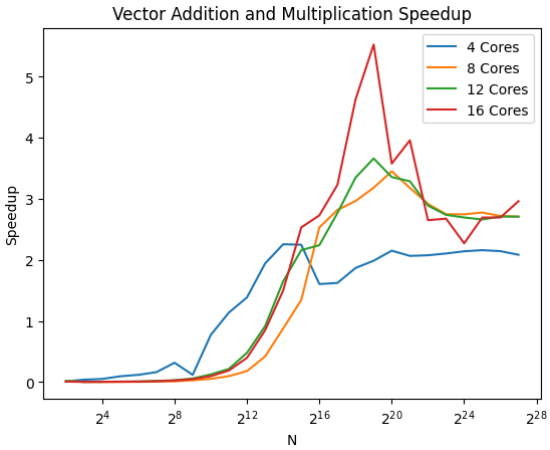

Figure 12: Throughput vs Problem Size for Merge Sort on HPC Cluster

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. We extended our analysis to parallel implementations, evaluating their scalability and efficiency across multiple processing units. Based on profiling results, we have provided insights into optimization strategies, such as vectorization, OpenMP parallelization, and memory locality improvements, to enhance algorithmic performance on different hardware platforms.