CS 301
High-Performance Computing

Lab 4 - Performance evaluation of codes

Problem B-2: Multiplication of Two vectors followed by summation.

Jay Dobariya (202101521)
Akshar Panchani (202101522)

February 28, 2024

Contents

1 Introduction

Here in this code performance we would do the multiplication of the two vectors and find out the complexity and its detailed perspective in terms of high performance computing. The mathematical terms and its case study is given further in the report.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 6
- On-line CPU(s) list: 0-5
- Thread(s) per core: 1
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 155
- Model name: Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz
- Stepping: 10
- CPU MHz: 799.992
- CPU max MHz: 4100.0000
- CPU min MHz: 800.0000
- BogoMIPS: 6000.00
- Virtualization: VT-x
- L1d cache: 192KB
- L1i cache: 192KB
- L2 cache: 1.5MB
- L3 cache: 9MB
- NUMA node0 CPU(s): 0-5

2.2 Hardware Details for HPC Cluster (Node gics1)

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 24
- On-line CPU(s) list: 0-23
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 63
- Model name: Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
- Stepping: 2
- CPU MHz: 2642.4378
- BogoMIPS: 4804.69
- Virtualization: VT-x
- L1d cache: 32K
- L1i cache: 32K
- L2 cache: 256K
- L3 cache: 15360K
- NUMA node0 CPU(s): 0-5,12-17
- NUMA node1 CPU(s): 6-11,18-23

3 Problem B2

3.1 Brief description with its algorithm

In this question, we add and multiply two vectors vector-wise, then total up the results. $(A(i) \cdot B(i)) + (A(i) + B(i))$ is the definition of the operation, where A and B are input vectors of size N. In the serial version, addition and multiplication are carried out element-by-element while a single master thread iteratively covers the whole issue size.

1. Initialize variables: $result = 0$.
2. For i from 1 to N (size of vectors):
   (a) Perform element-wise addition and multiplication: $temp = A(i) \cdot B(i) + A(i) + B(i)$.
   (b) Add $temp$ to $result$.
3. The final result is the summation of $(A(i) \cdot B(i)) + (A(i) + B(i))$ for all i.

3.2 The complexity of the algorithm (serial)

The time complexity of the described algorithm for vector-wise addition and multiplication followed by summation in a serial implementation is $O(N)$, where N is the size of the input vectors.

3.3 Information about parallel implementation

The provided code implements matrix multiplication using a block approach in parallel using OpenMP directives. Here's a brief explanation of the parallel implementation:

- The program takes two command-line arguments: n (size of the input array) and p (number of processors).
- It initializes two arrays a and b of size n, and then computes the element-wise multiplication and addition of corresponding elements of a and b.
- The computation is parallelized using OpenMP directives. The loop iterating over the elements of the arrays is parallelized with the 'omp parallel for' directive, and reduction is used to compute the final result in parallel.
- The number of threads (p) determines the level of parallelism, with each thread computing a subset of the total iterations.
- The result of the computation is stored in a variable c.

3.4 The complexity of the algorithm (Parallel)

The complexity of the algorithm can be analyzed as follows:

- **Parallelism:** The parallel implementation uses OpenMP to distribute the computation of matrix elements across multiple threads, allowing for concurrent execution on multiple CPU cores. The number of threads (p) determines the level of parallelism, with each thread computing a subset of the total elements. However, the overhead of parallelization and synchronization may impact performance for very small values of n or large values of p.

3.5 Profiling using HPC Cluster (with gprof)

The screenshots of profiling using the HPC Cluster are given below

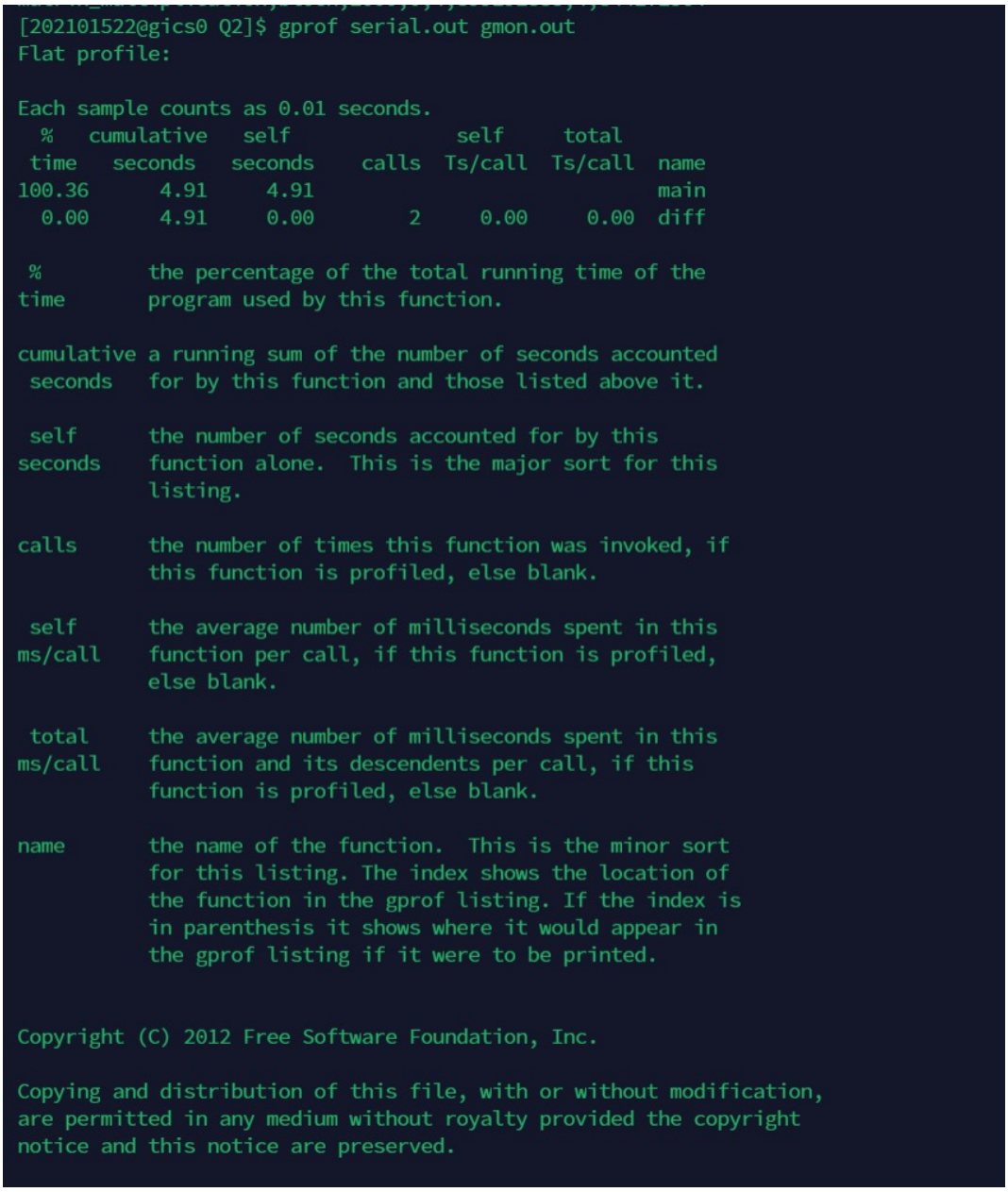

Figure 1: Screenshot of terminal from HPC Cluster

3.6 Graph of Problem Size vs Runtime

3.6.1 Graph of Problem Size vs Totaltime for LAB207 PCs

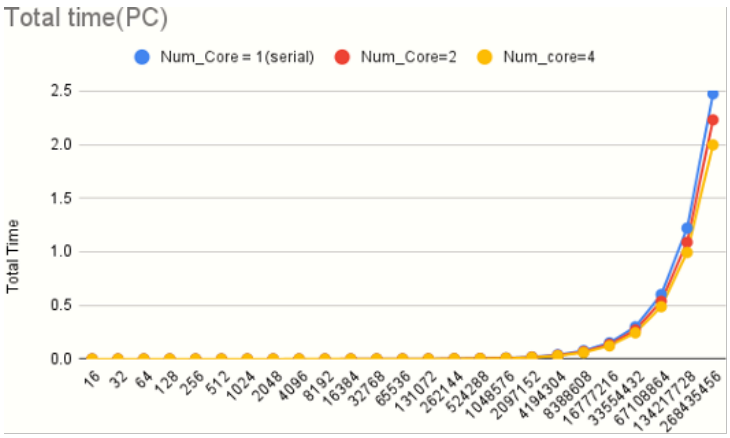

Figure 2: Total mean execution time (Total time) vs Problem size plot for **problem size 10^8 (Hardware: LAB207 PC, Problem: MUL_VECTORS)**.

3.6.2 Graph of Problem Size vs Algorithmtime for LAB207 PCs

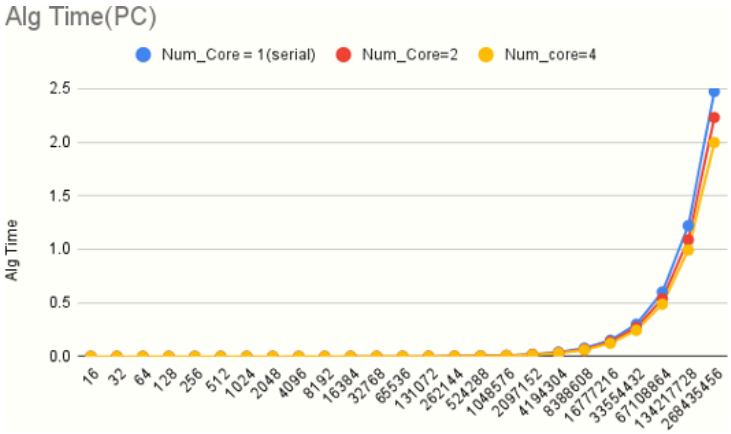

Figure 3: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 10^8 (Hardware: LAB207 PC, Problem: MUL_VECTORS)**.

3.6.3 Graph of Problem Size vs Totaltime for HPC Cluster

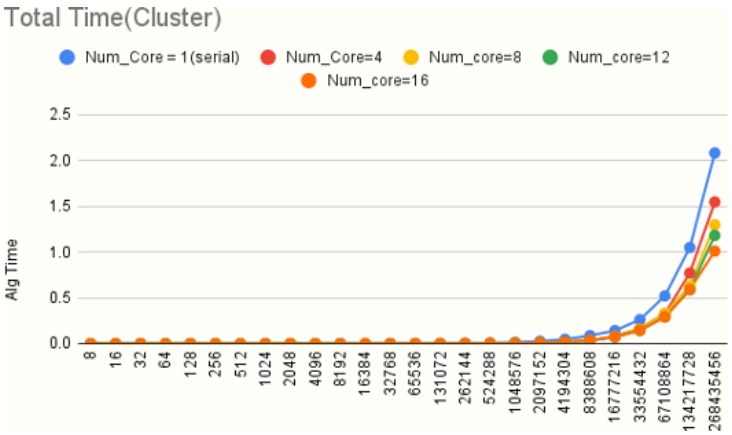

Figure 4: Total mean execution time (Total time) vs Problem size plot for **problem size 10^8.(Hardware: HPC Cluster, Problem: MUL_VECTORS)**.

3.6.4 Graph of Problem Size vs Algorithmtime for HPC Cluster

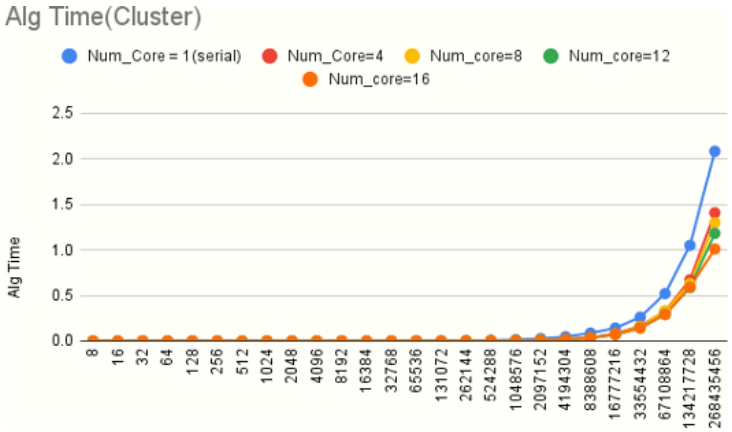

Figure 5: Total mean execution time (Algorithm time) vs Problem size plot for **problem size 10^8.(Hardware: HPC Cluster, Problem: MUL_VECTORS)**.

4 Conclusions

- Here we carried out the complex multiplication on both the lab PC and the cluster and we differentiated the result through the above resulted graphs. This shows the flavors of the computing with its time and space complexity that how a matrix multiplication and its addition is carried out.
- The cluster machine executes the given equation faster than the LAB207 machine for equivalent problem sizes.

- As the problem size increases its running time also increases. This shows its disparity.
- Overall, the parallel implementation improves the performance of the matrix multiplication algorithm by utilizing multiple threads to compute the elements of the resulting matrix concurrently.