CS 301
High-Performance Computing

Lab 01 - CPU architecture, Triad and Measuring Performance

Diya Patel (202301216)
Yesha Joshi (202301462)

February 2026

Contents

1 Introduction

In this lab, we will measure the CPU performance of both our Lab PC and the Cluster. We will also study the system architecture by implementing the vector triad code and its variations: copy, scale, and sum. These operations were discussed in our theory class, and now we will test them in practice to understand their impact on performance.

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): 12
- On-line CPU(s) list: 0-11
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 151
- Model name: 12th Gen Intel(R) Core(TM) i5-12500
- Stepping: 5
- CPU max MHz: 4600.0000
- CPU min MHz: 800.0000
- BogoMIPS: 5990.40
- Virtualization: VT-x
- L1d cache: 288K
- L1i cache: 192K
- L2 cache: 7.5M

- L3 cache: 18M
- NUMA node0 CPU(s): 0-11
- Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm epb invpcid_single tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts

2.2 Hardware Details for HPC Cluster (Node gics0)

- 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: 2066.812
- 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 Description

In this lab, we aim to measure the CPU performance of both the Lab PC and the Cluster by running the Stream Benchmark. This benchmark evaluates memory bandwidth using four operations: Copy, Scale, Sum, and Triad. These operations involve reading and writing large arrays, and their performance is limited by memory bandwidth rather than computational power.

To analyze performance, we measured throughput (GB/s) vs. problem size and Bandwidth vs. problem size for each operation. The results were plotted separately for the Lab PC and the Cluster.

4 Benchmarking Methodology

We executed the operations on both the Lab PC and the Cluster, measuring:

- Execution time for varying problem sizes.
- Throughput and Bandwidth for each operation.
- Differences in performance between the Lab PC and the Cluster.

For each case, we generated plots of: The memory bandwidth is calculated as

$$\text{Bandwidth (GB/s)} = \frac{\text{Bytes per element} \times N_p \times \text{RUNS}}{T_{\text{algo}} \times 10^9}$$

The computational throughput is calculated as

$$\text{Throughput (MFLOPs)} = \frac{\text{FLOPs per element} \times N_p \times \text{RUNS}}{T_{\text{algo}} \times 2^{20}}$$

5 Graphical Results

5.1 Vector Copy Operation : $a[i] = b[i]$

5.1.1 Bandwidth using Lab PC

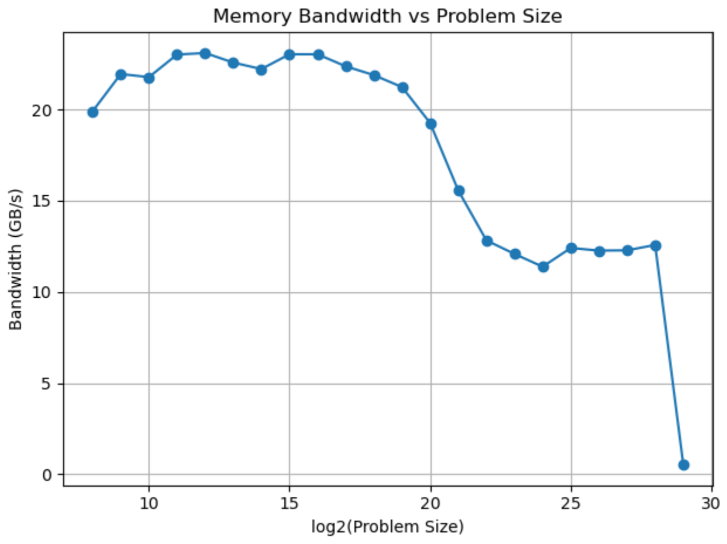

Figure 1: Bandwidth vs. Problem Size for the vector copy operation using Lab PC.

5.1.2 Bandwidth using HPC Cluster

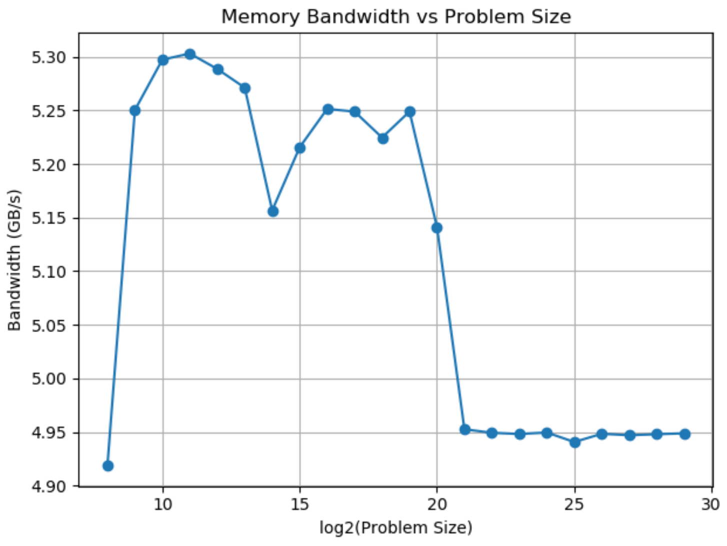

Figure 2: Bandwidth vs. Problem Size for the vector copy operation using HPC Cluster.

5.2 Vector Scaling Operation : $a[i] = k * b[i]$

5.2.1 Throughput and Bandhwidth using Lab PC

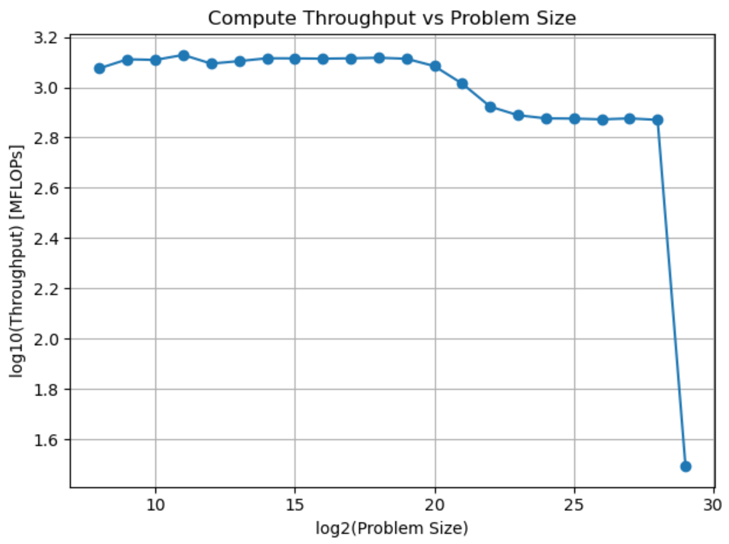

Figure 3: Throughput vs. Problem Size for the vector scaling operation using Lab PC.

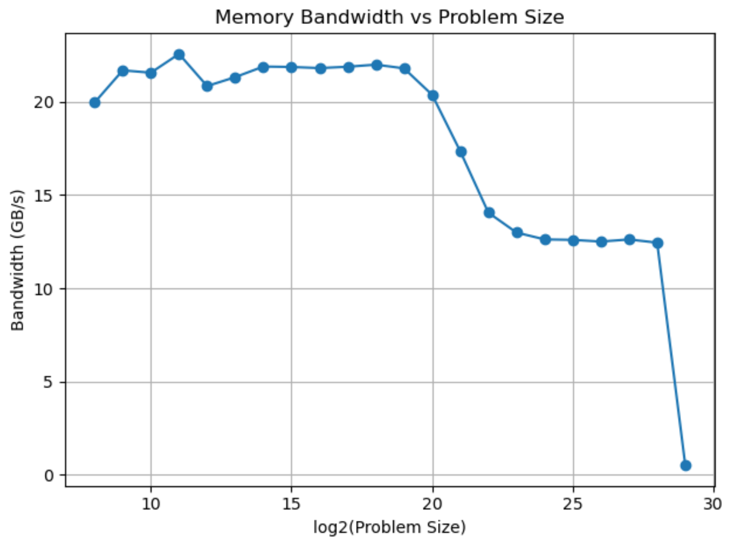

Figure 4: Bandwidth vs. Problem Size for the vector scaling operation using Lab PC.

5.2.2 Throughput and Bandwidth using HPC Cluster

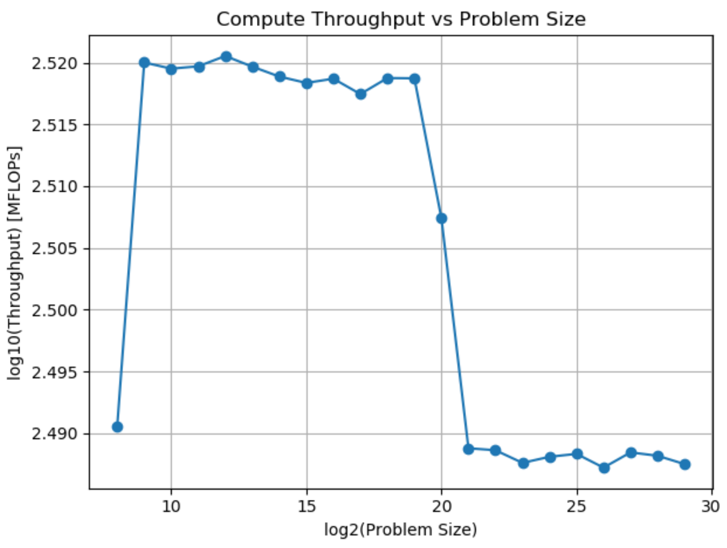

Figure 5: Throughput vs. Problem Size for the vector triad operation using HPC Cluster

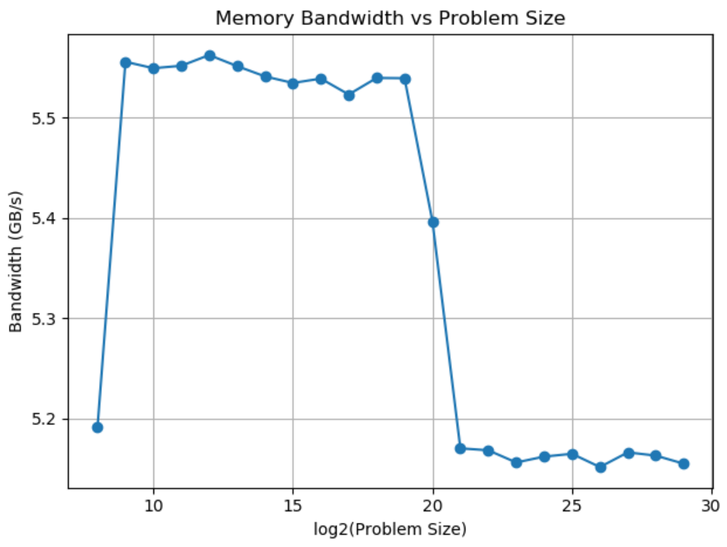

Figure 6: Bandwidth vs. Problem Size for the vector triad operation using HPC Cluster.

5.3 Vector Sum Operation : $a[i] = b[i] + c[i]$

5.3.1 Throughput and Bandwidth using Lab PC

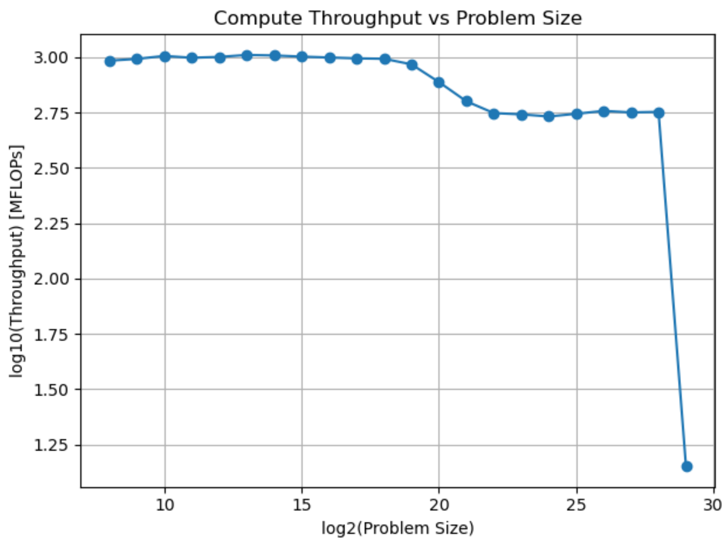

Figure 7: Throughput vs. Problem Size for the vector sum operation using Lab PC.

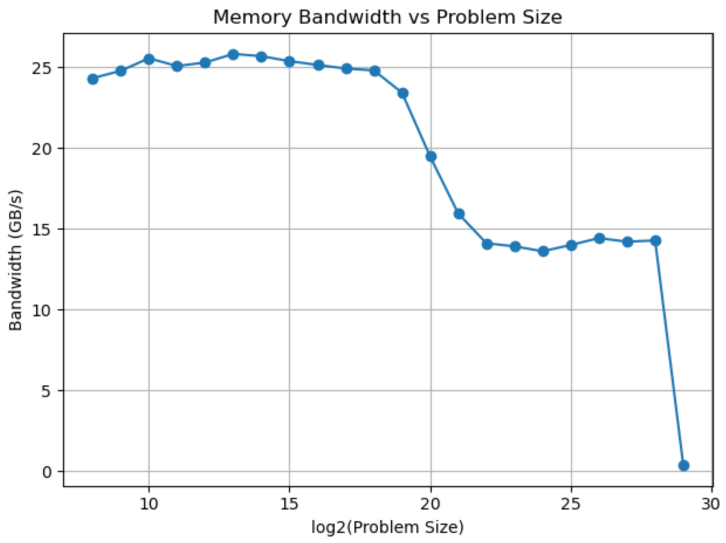

Figure 8: Bandwidth vs. Problem Size for the vector sum operation using Lab PC.

5.3.2 Throughput and Bandwidth using HPC Cluster

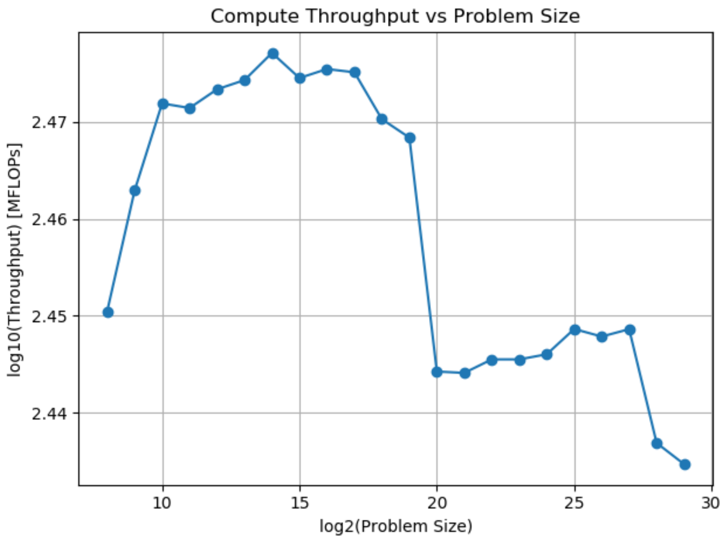

Figure 9: Throughput vs. Problem Size for the vector sum operation using HPC Cluster

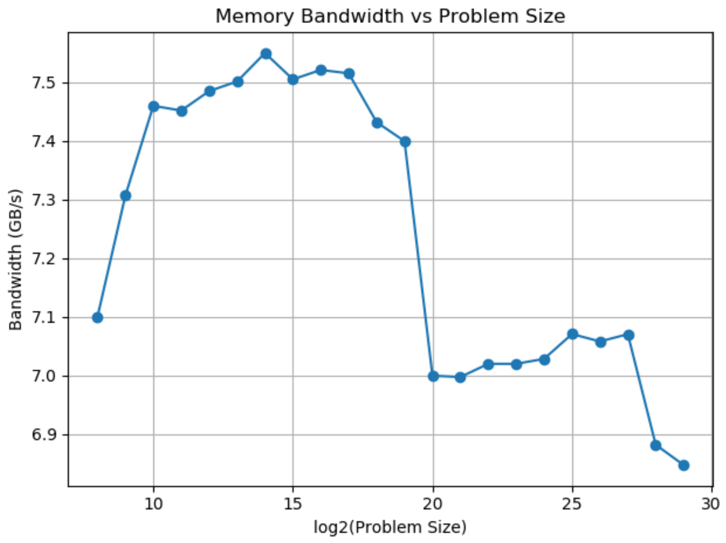

Figure 10: Bandwidth vs. Problem Size for the vector sum operation using HPC Cluster.

5.4 Vector Triad Operation : $a[i] = b[i] + c[i] * d[i]$

5.4.1 Throughput and Bandwidth using Lab PC

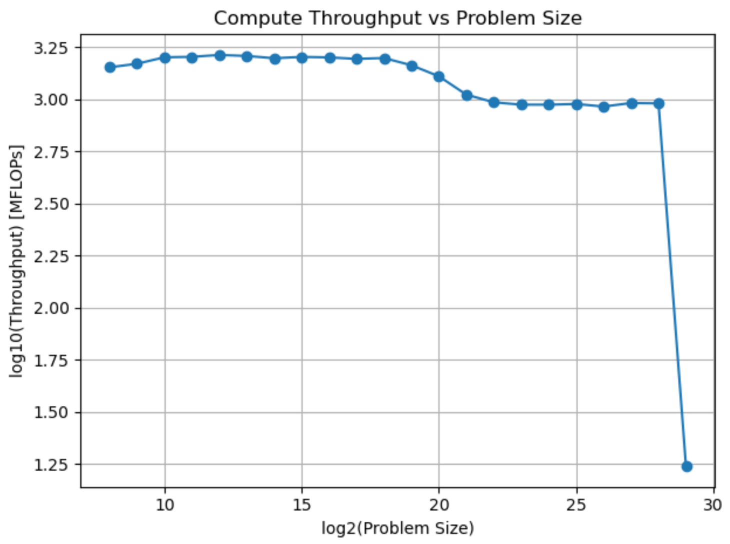

Figure 11: Throughput vs. Problem Size for the vector triad operation using Lab PC.

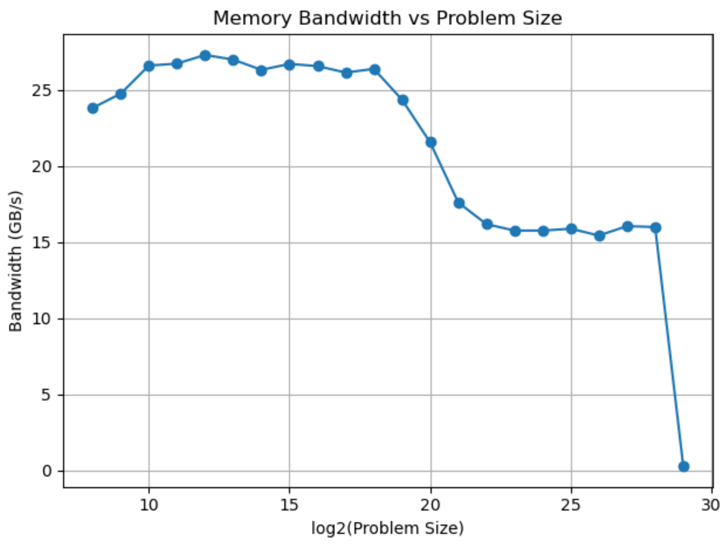

Figure 12: Bandwidth vs. Problem Size for the vector triad operation using Lab PC.

5.4.2 Throughput and FLOPs using HPC Cluster

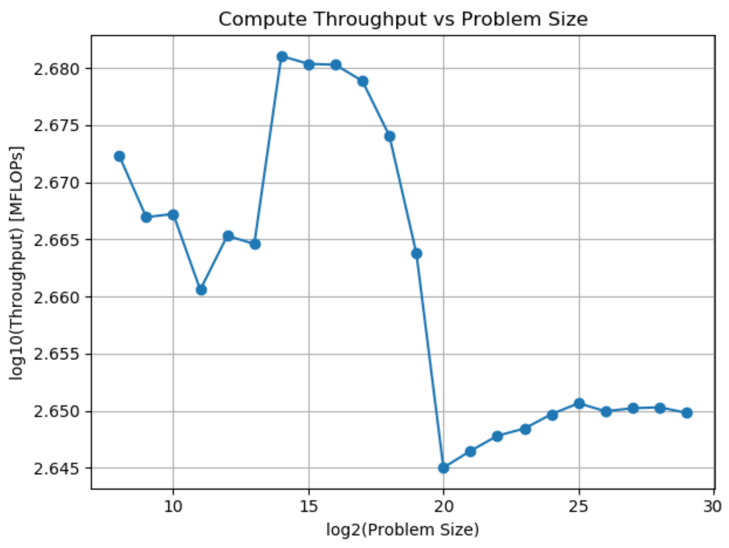

Figure 13: Throughput vs. Problem Size for the vector triad operation using HPC Cluster.

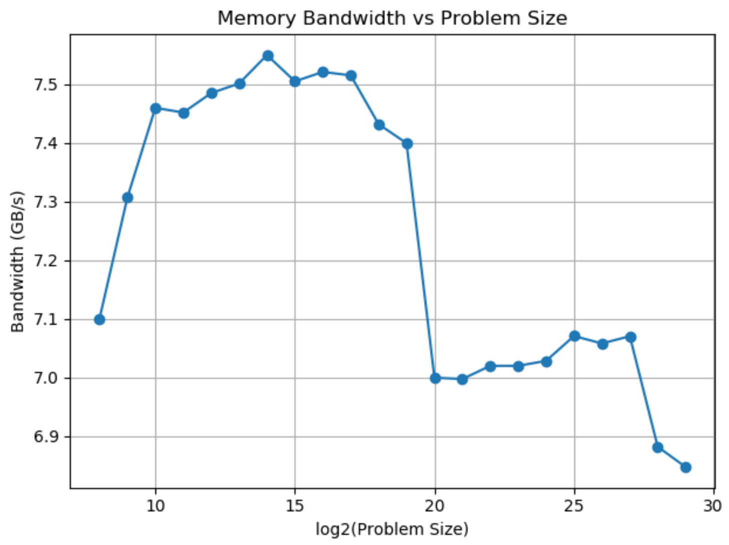

Figure 14: FLOPs vs. Problem Size for the vector triad operation using HPC Cluster.

6 Analysis Task

6.1 Timing Functions

The runtime is measured using `clock_gettime()` with the `CLOCK_MONOTONIC` clock. Two timings are recorded: the end-to-end time and the algorithm time. The algorithm time measures only the execution of the vector triad kernel, excluding initialization and memory allocation overhead, ensuring accurate performance analysis.

The `clock_gettime()` function provides high-resolution wall-clock timing and is not affected by system time changes, making it suitable for HPC benchmarking. Another timing function is `clock()`, which measures CPU time but has lower resolution and does not account for memory stalls, making it less suitable for this experiment.

6.2 Comparison of Measured and Theoretical Peak Performance

The memory bandwidth is calculated as

$$\text{Bandwidth (GB/s)} = \frac{\text{Bytes per element} \times N_p \times \text{RUNS}}{T_{\text{algo}} \times 10^9}$$

The computational throughput is calculated as

$$\text{Throughput (MFLOPs)} = \frac{\text{FLOPs per element} \times N_p \times \text{RUNS}}{T_{\text{algo}} \times 2^{20}}$$

The theoretical peak compute performance of the Lab PC is approximately 31.2 GFLOPs, and that of the HPC cluster is approximately 57.6 GFLOPs, calculated using the number of physical cores, clock frequency, and floating-point operations per cycle. However, the measured compute throughput obtained from the experiments is much lower than these peak values 3 MFLOPs, indicating underutilization of the CPU's computational capability.

In contrast, the measured memory bandwidth reaches a stable plateau at large problem sizes and closely matches the peak sustainable bandwidth observed in the bandwidth plots (approximately 15–16 GB/s for the Lab PC and 7–7.5 GB/s for the HPC cluster). This confirms that the performance of the vector operations is limited by memory bandwidth rather than computation, and the kernels are memory-bound.

6.3 Impact of Cache Sizes on Performance

Cache hierarchy significantly affects performance for small problem sizes, where the working data fits within the L1 and L2 caches, resulting in higher throughput and bandwidth. As the problem size increases and exceeds cache capacity, cache misses increase and data must be accessed from main memory, causing a drop in performance.

Once the working size exceeds the last-level cache, the bandwidth stabilizes and becomes independent of problem size, indicating memory-bound execution.

7 Conclusion

We tested all four operations on the Lab PC as well as HPC cluster. Performance was stable for small problem sizes but dropped as the size increased. This drop happens because memory bandwidth becomes a bottleneck. The system performs well in computing, but memory access slows it down. The actual performance is lower than the theoretical peak due to these memory limits. A sudden large drop in performance likely means the problem size exceeded a cache limit, forcing slower memory access.