CS 301
High-Performance Computing

Lab 01 - CPU Architecture, Memory Hierarchy, and Performance Measurement

Visha Sitapara (202301414)
Dhruv Patel (202301095)

February 3, 2026

Contents

1 Introduction **3**

2 Hardware Details **3**
- 2.1 Hardware Details for LAB207 PCs . . . 3
- 2.2 Hardware Details for HPC Cluster (Node gics0) . . . 4

3 Problem Description **5**

4 Benchmarking Methodology **5**

5 Graphical Results **7**
- 5.1 Vector Copy Operation : $a[i] = b[i]$ . . . 7
  - 5.1.1 Throughput and Bandwidth using Lab PC . . . 7
  - 5.1.2 Throughput and Bandwidth using HPC Cluster . . . 8
- 5.2 Vector Scaling Operation : $a[i] = k * b[i]$ . . . 9
  - 5.2.1 Throughput and Bandwidth using Lab PC . . . 9
  - 5.2.2 Throughput and Bandwidth using HPC Cluster . . . 10
- 5.3 Vector Sum Operation : $a[i] = b[i] + c[i]$ . . . 11
  - 5.3.1 Throughput and Bandwidth using Lab PC . . . 11
  - 5.3.2 Throughput and Bandwidth using HPC Cluster . . . 12
- 5.4 Vector Triad Operation : $a[i] = b[i] + c[i] * d[i]$ . . . 13
  - 5.4.1 Throughput and Bandwidth using Lab PC . . . 13
  - 5.4.2 Throughput and Bandwidth using HPC Cluster . . . 14

6 Analysis Tasks **15**
- 6.1 Accurate Runtime Measurement and Timing Functions . . . 15
- 6.2 Bandwidth vs Working Data Size . . . 15
- 6.3 Performance vs Working Data Size . . . 15
- 6.4 Computation and Memory Time Analysis for the Triad Kernel . . . 15
- 6.5 Comparison with Theoretical Peak Performance . . . 16
- 6.6 Impact of Cache Sizes on Performance . . . 16

7 Conclusion **16**

1 Introduction

The objective of this assignment is to understand how the CPU architecture and the memory hierarchy influence the program performance. Students will study this using simple loop-based computational kernels similar to the STREAM benchmark and analyze memory bandwidth and computation performance.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Address sizes: 46 bits physical, 48 bits virtual
- 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 E5-2640 v3 @ 2.60GHz
- Stepping: 2
- CPU MHz: 2907.656
- 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

We consider a simple particle-based computation:

- A 2D square domain of size 1 cm $\times$ 1 cm.
- N_p particles are randomly distributed within this domain.
- A conceptual mesh of size $N_x \times N_y$ defines the spatial extent, but no mesh-based computation is performed in this assignment.

Each particle has:

- Position (x, y)
- Velocity v

All particles have the same constant mass m. The kinetic energy of each particle is computed as:

$$E = \frac{1}{2}mv^2$$

The purpose of this setup is to generate a memory-intensive workload that allows performance analysis across different working data sizes.

4 Benchmarking Methodology

The following benchmark kernels, similar to the STREAM benchmark, are implemented:

- **Copy:** $x[p] = y[p]$
- **Scale:** $x[p] = a \times y[p]$
- **Add:** $s[p] = x[p] + y[p]$
- **Triad:** $s[p] = x[p] + a \times y[p]$
- **Energy Kernel (optional):** $E[p] = \frac{1}{2}mv[p]^2$

For simplicity, the particle mass is taken as $m = 2$. Here, a denotes a scalar constant.

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

- Execution time for varying problem sizes.

- Throughput (GB/s) and Bandwidth for each operation.
- Differences in performance between the Lab PC and the Cluster.

For each case, we generated plots of:

- Throughput (Bytes/s) vs. Problem Size

$$Throughput = \frac{sizeof(double) \times N \times Total}{alg\_time}$$

  where, N is the number of vectors actually used in the algorithm.

- Bandwidth vs. Problem Size

$$Bandwidth = \frac{sizeof(double) \times M \times Total}{alg\_time \times 10^9}$$

  where, M is the number of vectors theoretically used in the algorithm.

5 Graphical Results

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

5.1.1 Throughput and Bandwidth using Lab PC

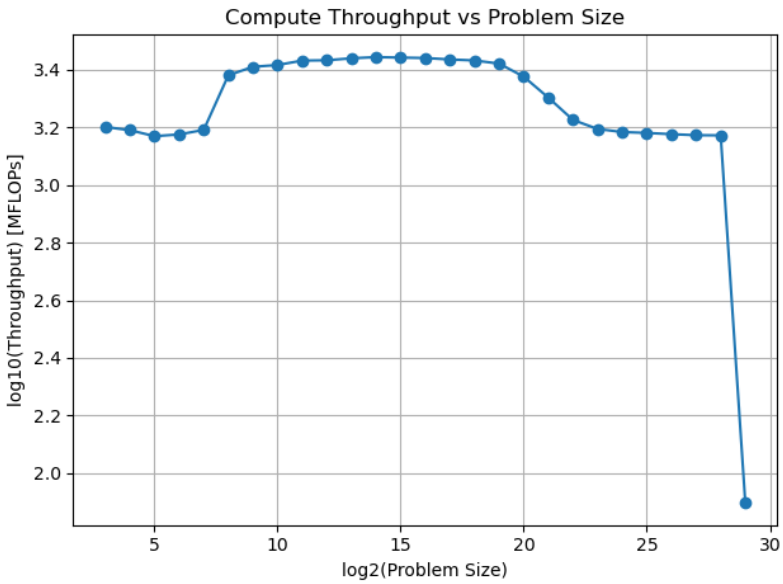

Figure 1: Throughput vs. Problem Size for the vector copy operation using Lab PC.

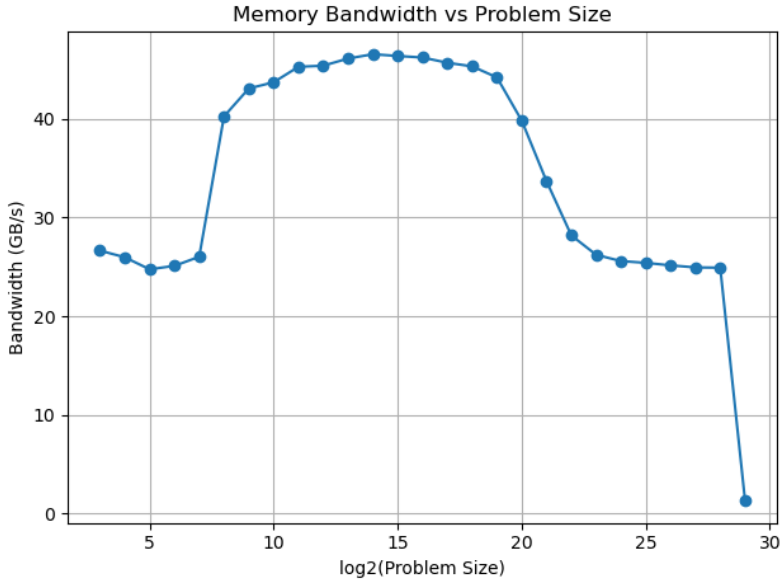

Figure 2: Bandwidth vs. Problem Size for the vector copy operation using Lab PC.

5.1.2 Throughput and Bandwidth using HPC Cluster

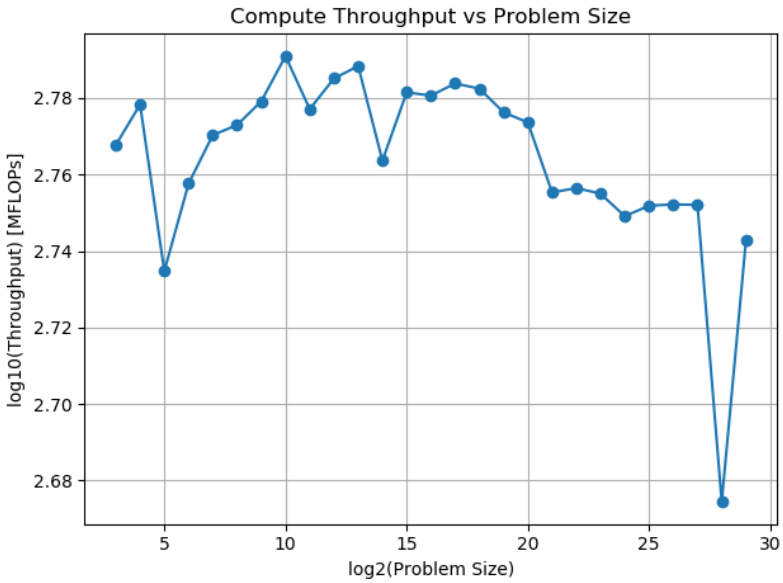

Figure 3: Throughput vs. Problem Size for the vector copy operation using HPC Cluster.

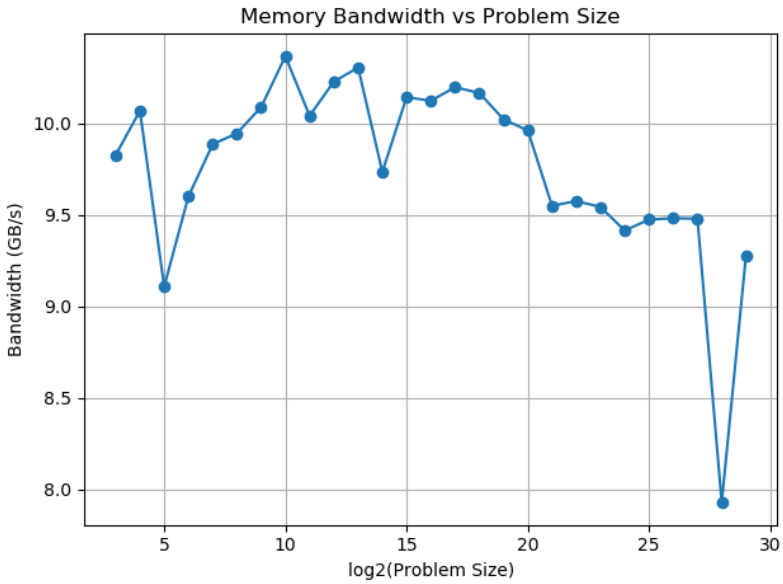

Figure 4: Bandwidth vs. Problem Size for the vector copy operation using HPC Cluster.

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

5.2.1 Throughput and Bandwidth using Lab PC

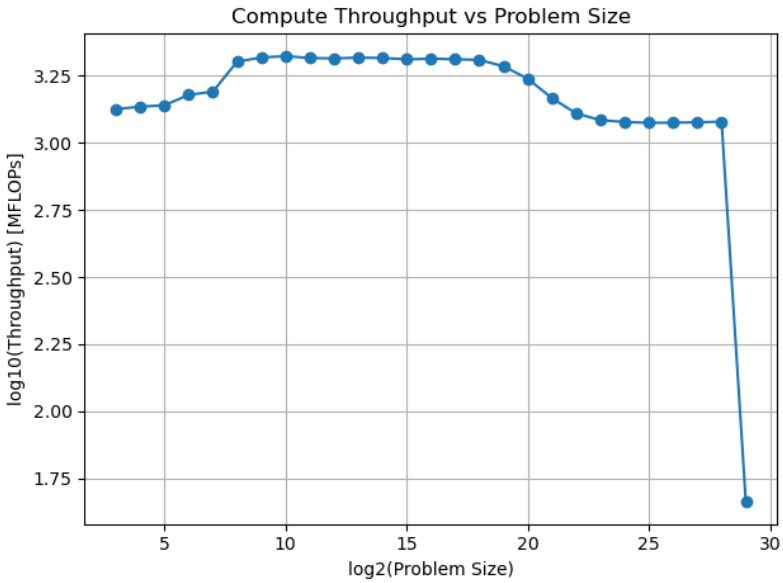

Figure 5: Throughput vs. Problem Size for the vector scaling operation using Lab PC.

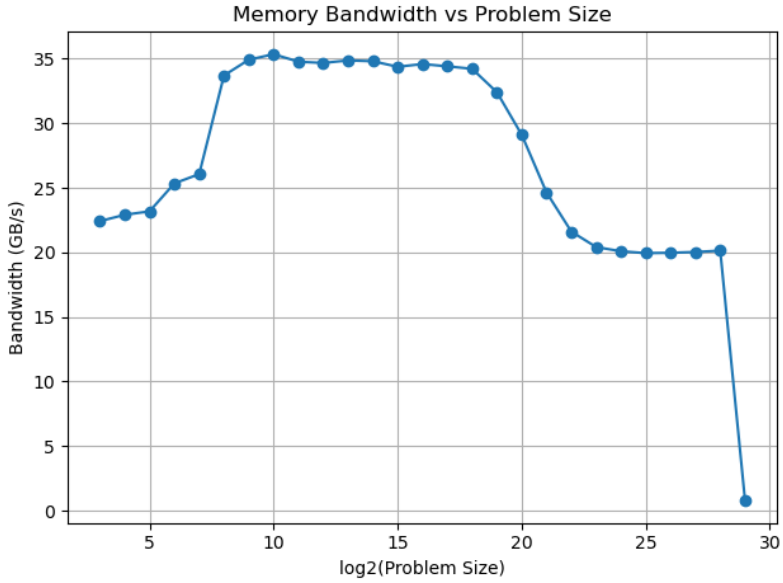

Figure 6: Bandwidth vs. Problem Size for the vector scaling operation using Lab PC.

5.2.2 Throughput and Bandwidth using HPC Cluster

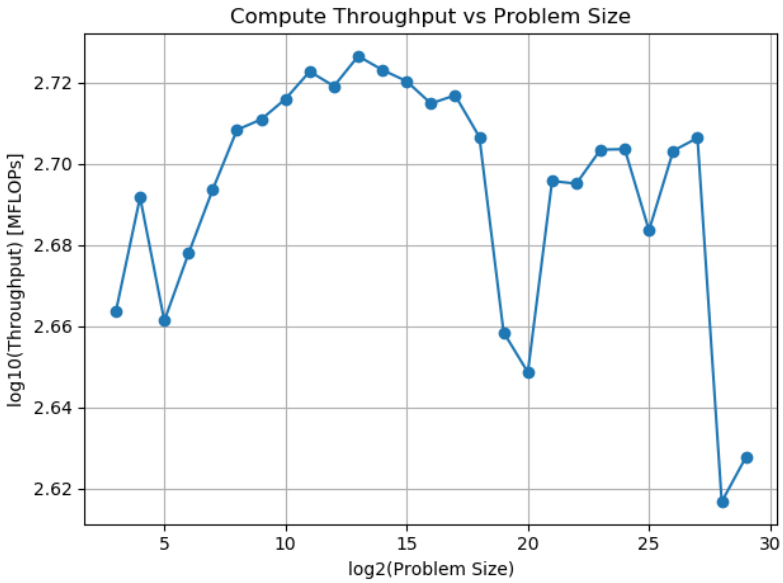

Figure 7: Throughput vs. Problem Size for the vector triad operation using HPC Cluster

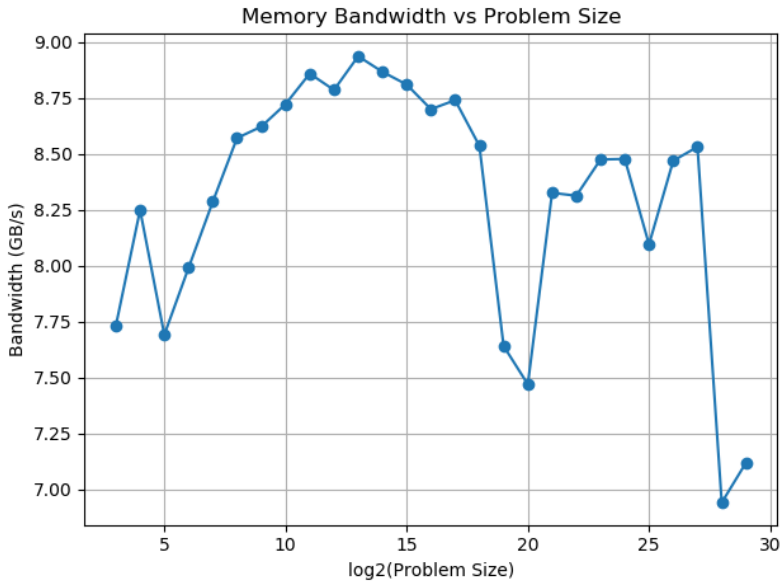

Figure 8: Bandwidth vs. Problem Size for the vector triad operation using HPC Cluster.

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

5.3.1 Throughput and Bandwidth using Lab PC

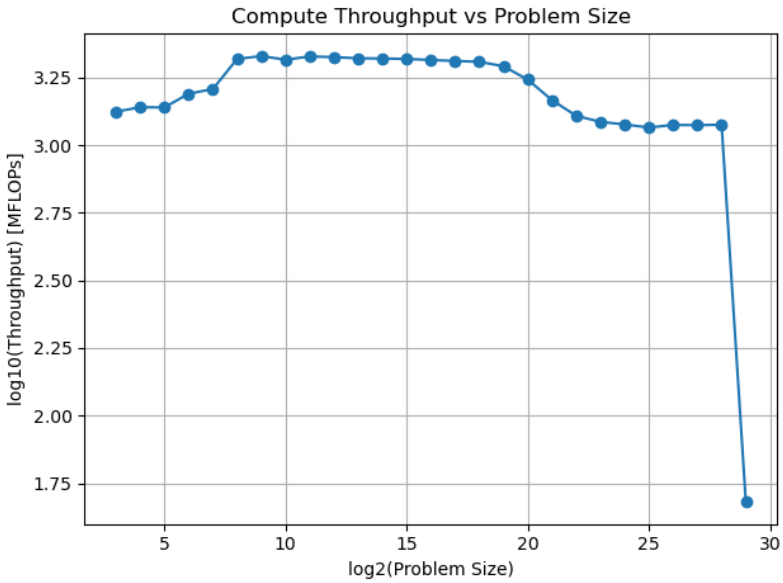

Figure 9: Throughput vs. Problem Size for the vector sum operation using Lab PC.

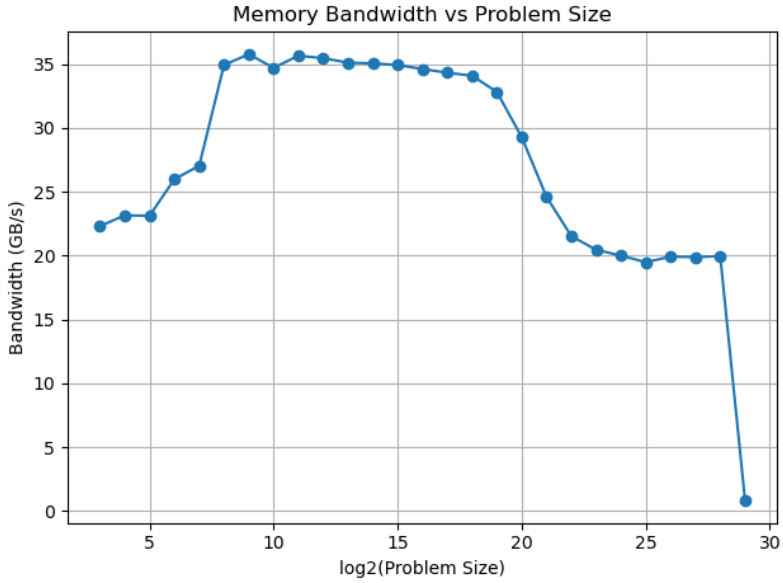

Figure 10: Bandwidth vs. Problem Size for the vector sum operation using Lab PC.

5.3.2 Throughput and Bandwidth using HPC Cluster

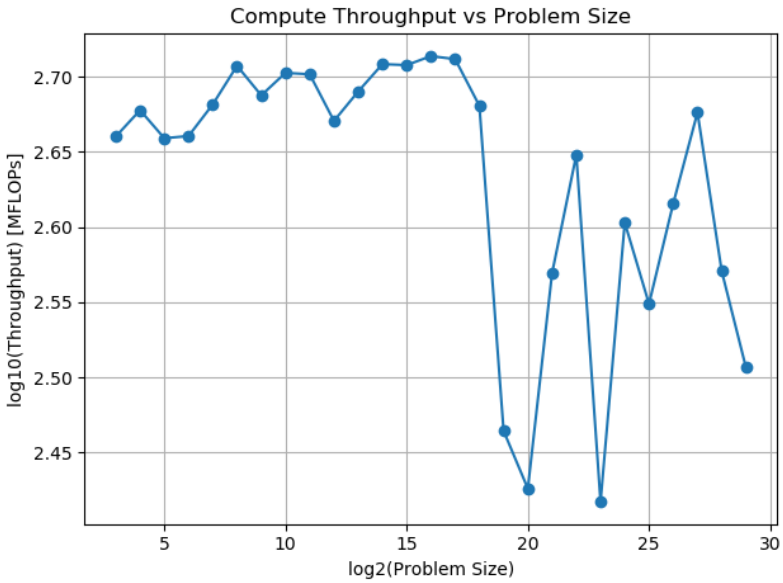

Figure 11: Throughput vs. Problem Size for the vector sum operation using HPC Cluster

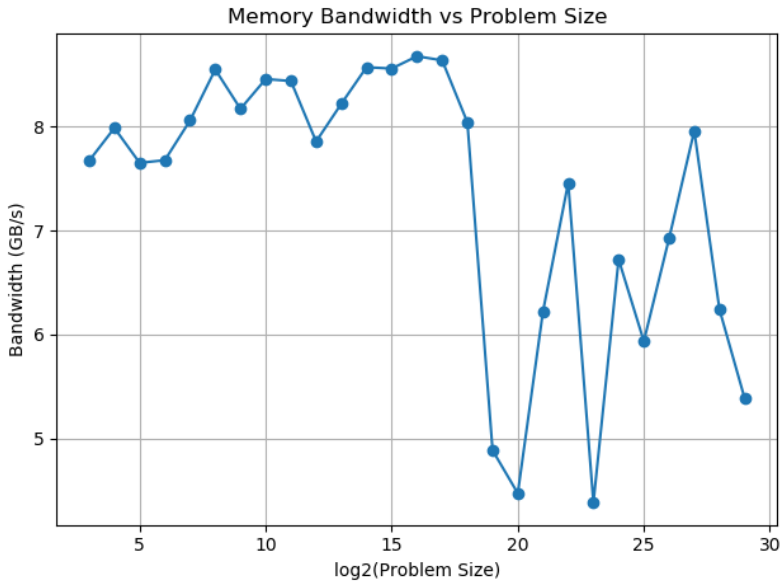

Figure 12: Bandwidth vs. Problem Size for the vector sum operation using HPC Cluster.

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

5.4.1 Throughput and Bandwidth using Lab PC

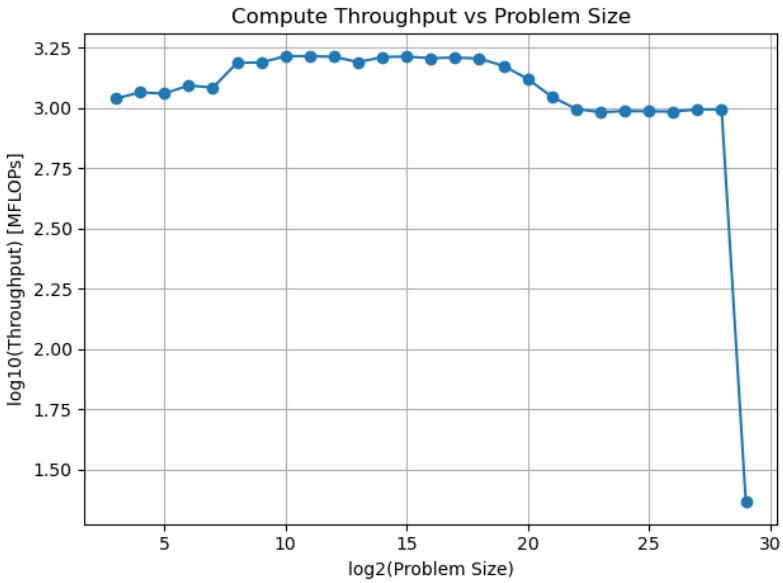

Figure 13: Throughput vs. Problem Size for the vector triad operation using Lab PC.

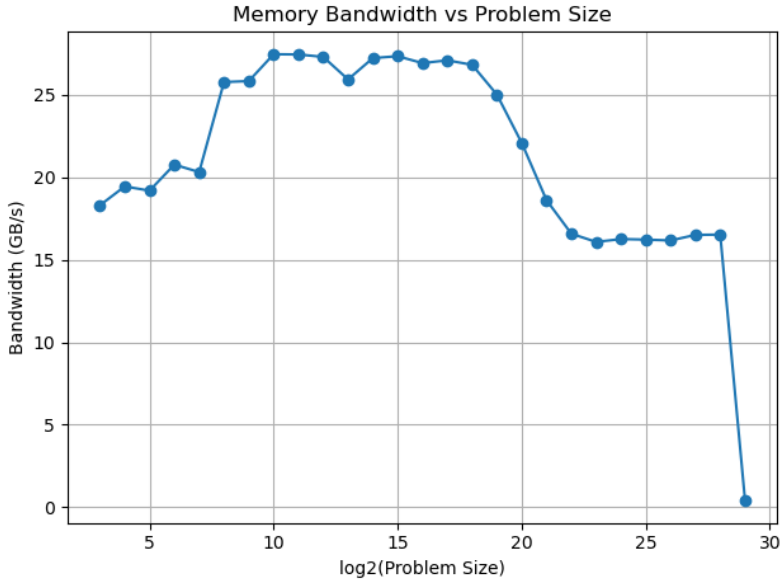

Figure 14: Bandwidth vs. Problem Size for the vector triad operation using Lab PC.

5.4.2 Throughput and Bandwidth using HPC Cluster

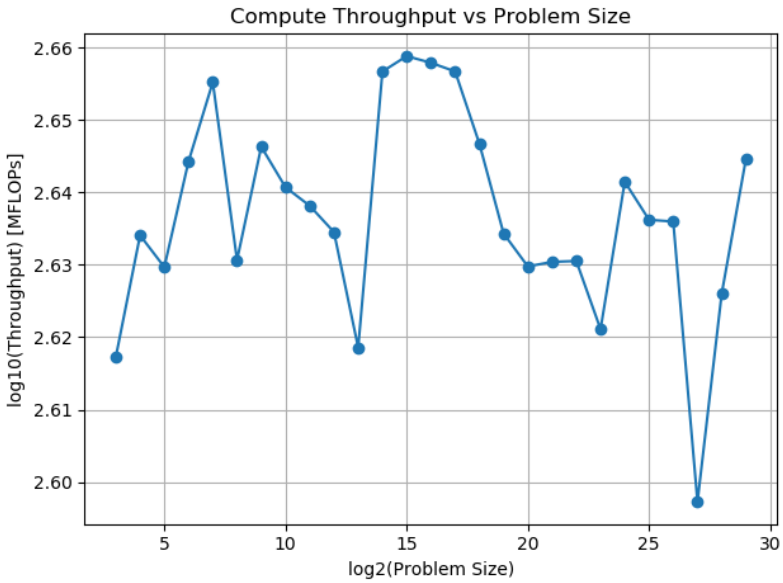

Figure 15: Throughput vs. Problem Size for the vector triad operation using HPC Cluster.

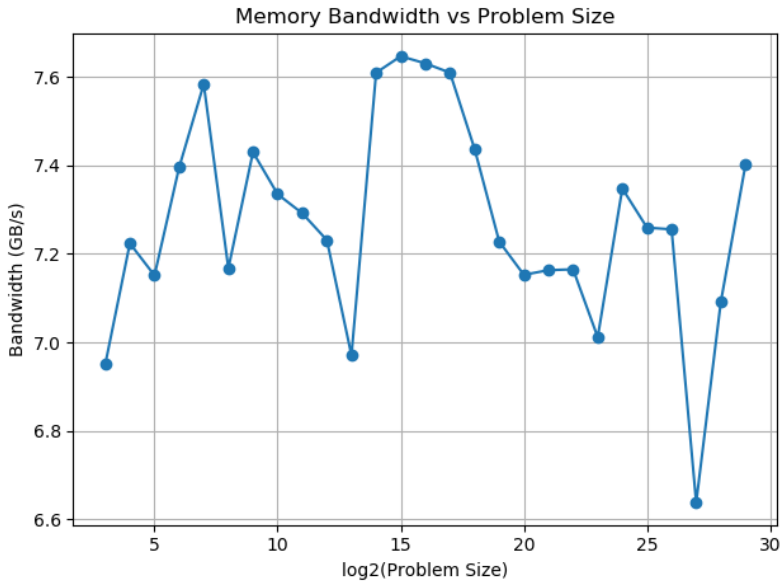

Figure 16: Bandwidth vs. Problem Size for the vector triad operation using HPC Cluster.

6 Analysis Tasks

6.1 Accurate Runtime Measurement and Timing Functions

To accurately measure runtime, the code records the start and end time of kernel execution and computes the difference. Commonly used timing functions are explained below:

- `clock()`: This function measures the CPU time consumed by a program. It has low resolution and does not accurately capture delays due to memory access, making it unsuitable for memory-intensive workloads.
- `clock_gettime()`: This function measures wall-clock time with nanosecond resolution. When used with `CLOCK_MONOTONIC`, it provides accurate and reliable timing unaffected by system clock changes. Therefore, it is preferred for performance benchmarking in this lab.

6.2 Bandwidth vs Working Data Size

Bandwidth was computed for each STREAM kernel as the amount of data transferred per unit time. The bandwidth is calculated using the expression:

$$Bandwidth = \frac{sizeof(double) \times M \times Total}{alg\_time \times 10^9}$$

where, M is the number of vectors theoretically used in the algorithm. The Copy and Scale kernels involve one read and one write operation per element, while the Add and Triad kernels involve two reads and one write per element.

6.3 Performance vs Working Data Size

Performance was measured in terms of throughput, for each STREAM kernel operation. The performance was computed as:

$$Throughput = \frac{sizeof(double) \times N \times Total}{alg\_time}$$

where, N is the number of vectors actually used in the algorithm.The Triad kernel exhibited higher performance compared to the Add kernel due to its higher computational intensity.

6.4 Computation and Memory Time Analysis for the Triad Kernel

In the Triad kernel, each iteration performs one multiplication and one addition, resulting in two floating-point operations. At the same time, it accesses multiple memory locations, including three reads and one write.

The computation time was estimated using the total number of floating-point operations and the theoretical peak compute performance of the CPU. The memory access time was estimated using the total data transferred and the measured memory bandwidth.

6.5 Comparison with Theoretical Peak Performance

In practice, the observed performance was significantly lower than the theoretical peak. This discrepancy arises due to memory bandwidth limitations, cache misses, and instruction overhead. Since the STREAM kernels are memory-intensive, they are unable to fully utilize the computational capabilities of the processor.

6.6 Impact of Cache Sizes on Performance

When the working data size fits within the L1 or L2 cache, low memory latency results in higher bandwidth and performance. As the working data size exceeds the cache capacity, cache misses increase, leading to frequent accesses to slower memory levels.

Once the working set exceeds all cache levels, performance stabilizes at a lower level determined by the main memory bandwidth.

7 Conclusion

The performance of all four vector operations was measured on both the lab system and the HPC cluster. At smaller problem sizes, performance remained almost constant because the data could be stored in the CPU cache. As the problem size increased, performance decreased gradually due to increased reliance on main memory access, which is slower than cache access. Even though the processor is capable of fast computation, memory bandwidth limits the overall performance. Consequently, the measured performance is lower than the theoretical peak. Sudden drops in performance indicate that the working data size has exceeded a cache level.