CS 301
High-Performance Computing

Lab 01 - Architecture, Memory Hierarchy, and Performance Measurement

Prem Kukadiya (202301452)
Suryadeepsinh Gohil (202301463)

January 31, 2026

Contents

1 Introduction

In this lab, we study how a computer's architecture and its memory levels affect how fast a program runs. We focus on the CPU and how it talks to the memory. To do this, we use simple math tasks called kernels: Copy, Scale, Add, and Triad. These are similar to the STREAM benchmark used in high-performance computing. By running these tasks on both our Lab PC and the HPC Cluster, we can measure how much data the memory can handle (bandwidth) and how many math operations the CPU can do (performance). This comparison helps us understand the performance differences between consumer-grade hardware and enterprise-level HPC systems.

2 Part A: CPU Architecture Study

The system architecture was examined using the `lscpu` command. The following tables summarize the hardware specifications of both the Lab PC and the HPC Cluster node used for benchmarking.

2.1 Lab PC Hardware Specifications

Table 1: Hardware Specifications for Lab PC (12th Gen Intel i5)

Parameter	**Details**
Architecture	x86_64
CPU Model Name	12th Gen Intel(R) Core(TM) i5-12500
Total CPUs (Threads)	12
Cores per Socket	6
Threads per Core	2
Socket(s)	1
NUMA node(s)	1
CPU Max Frequency	4600.0000 MHz
CPU Min Frequency	800.0000 MHz
L1d Cache	288 KiB (6 instances)
L1i Cache	192 KiB (6 instances)
L2 Cache	7.5 MiB (6 instances)
L3 Cache	18 MiB (1 instance)

2.2 HPC Cluster Hardware Specifications (Node gics1)

Table 2: Hardware Specifications for HPC Cluster Node (Intel Xeon E5-2620 v3)

Parameter	**Details**
Architecture	x86_64
CPU Model Name	Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz
Total CPUs (Threads)	24
Cores per Socket	6
Threads per Core	2
Socket(s)	2
NUMA node(s)	2
CPU Frequency	3183.187 MHz
BogoMIPS	4804.69
L1d Cache	32 KiB
L1i Cache	32 KiB
L2 Cache	256 KiB
L3 Cache	15360 KiB (15 MiB)
NUMA node0 CPU(s)	0-5, 12-17
NUMA node1 CPU(s)	6-11, 18-23

2.3 Theoretical Peak Memory Bandwidth

2.3.1 Lab PC

The theoretical peak memory bandwidth is the maximum rate at which data can be read from or written to the system memory by the processor. Assuming the system is equipped with dual-channel DDR4-3200 memory:

- **Memory Clock Speed:** 3200 MT/s
- **Data Bus Width:** 8 Bytes (64-bit)
- **Number of Channels:** 2

The calculation for the theoretical peak bandwidth is as follows:

$$\text{Theoretical Bandwidth} = 3200\ \text{MT/s} \times 8\ \text{Bytes} \times 2\ \text{Channels} = 51.2\ \text{GB/s} \quad (1)$$

This value represents the upper limit for the memory-intensive STREAM kernels on the Lab PC.

2.3.2 HPC Cluster

For the HPC Cluster node with dual-socket configuration and DDR4 memory per socket, the theoretical bandwidth would be higher due to the dual-socket NUMA architecture, allowing for potentially doubled memory bandwidth when both sockets are utilized efficiently.

3 Problem Description

The main goal of this lab is to measure and compare the CPU performance of both the Lab PC and the HPC Cluster by running the STREAM benchmark. This benchmark measures how fast data can move between the CPU and the memory, which is known as memory bandwidth.

We test four specific math operations:

- **Copy:** Moving data from one list to another.
- **Scale:** Multiplying a list by a constant number.
- **Add:** Adding two lists together.
- **Triad:** A mix of adding and multiplying lists.

These tasks are "memory-intensive," meaning their speed depends more on how fast the memory works than on how fast the CPU can calculate. We analyze the results by looking at the data transfer speed (Throughput in GB/s) and the calculation speed (Performance in FLOPs) across different data sizes for both systems.

4 Benchmarking Methodology

To perform these tests, we wrote the STREAM kernels in C++. We tested a wide range of data sizes, starting from a small number of elements (2^8) and doubling it until we reached a very large size (2^{29}).

The process for our benchmark is as follows:

1. **Setup:** We create large arrays (lists of numbers) and fill them with initial values.
2. **Timing:** For each data size, we use high-resolution timers to measure exactly how long the CPU takes to finish the task.
3. **Repetition:** We repeat the tasks many times for smaller sizes to make sure our measurements are accurate and consistent.
4. **Calculation:** We calculate the final results using these formulas:
    - **Throughput (GB/s):** $\frac{\text{Total Bytes Accessed}}{\text{Execution Time}}$.
    - **FLOPs:** $\frac{\text{Total Math Operations}}{\text{Execution Time}}$.

Finally, we create graphs to show how the speed changes as the data size grows, especially when the data becomes too large for the CPU's internal cache memory. These tests are executed on both the Lab PC and the HPC Cluster to compare their performance characteristics.

5 Part B: Benchmarking and Performance Analysis

5.1 Code Implementation and Logic

The benchmarking system is built using C++ for the core math and Python for automation. The logic follows a simple flow: first, memory is allocated and filled with random numbers for the vectors. Then, the code runs the math operations (like the Triad kernel) many times across a wide range of data sizes. We use high-precision timers to measure exactly how long the math takes. Finally, a Python script runs the program multiple times to find the average speed and creates graphs to show how the performance changes as the data size grows.

5.2 Timing Methodology

To measure the performance accurately, we must use timers that can track very small fractions of a second. In our implementation, we focus on measuring the "Algorithm Time," which is the time spent only on the math operations, excluding the time taken to set up the data or allocate memory.

5.2.1 Explanation of Timing Functions

As per the lab requirements, we examined two different timing functions used in C/C++:

- `clock_gettime()`: A modern, high-resolution timer that can provide nanosecond-resolution timestamps. We use it with `CLOCK_MONOTONIC`, which guarantees the clock is monotonic (only moves forward) and is not affected by changes to the system wall-clock time, making it the preferred choice for benchmarking.
- `clock()`: The older C standard function that reports the CPU time used by the program (the returned value divided by `CLOCKS_PER_SEC` gives seconds). It is easy to use but typically has much coarser resolution (often milliseconds), so it is less suitable for timing very fast code. Because it measures CPU time rather than elapsed (wall-clock) time, results can be misleading on multi-threaded or multi-core workloads.

5.2.2 Accurate Measurement Strategy

To get the most accurate results, our code follows these steps:

1. We start the timer exactly before the loop of math operations begins.
2. We stop the timer immediately after the loop ends.
3. For small data sizes, we repeat the math thousands of times (`RUNS`) so the total time is long enough to be measured reliably.
4. We subtract the start time from the end time and convert the result into seconds.

6 Graphical Results

In this section, we present the results for the four STREAM kernels. We plot Bandwidth (GB/s) to show memory performance and Throughput (MFLOPs) to show computational performance against the problem size (N_p).

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

6.1.1 Lab PC Results

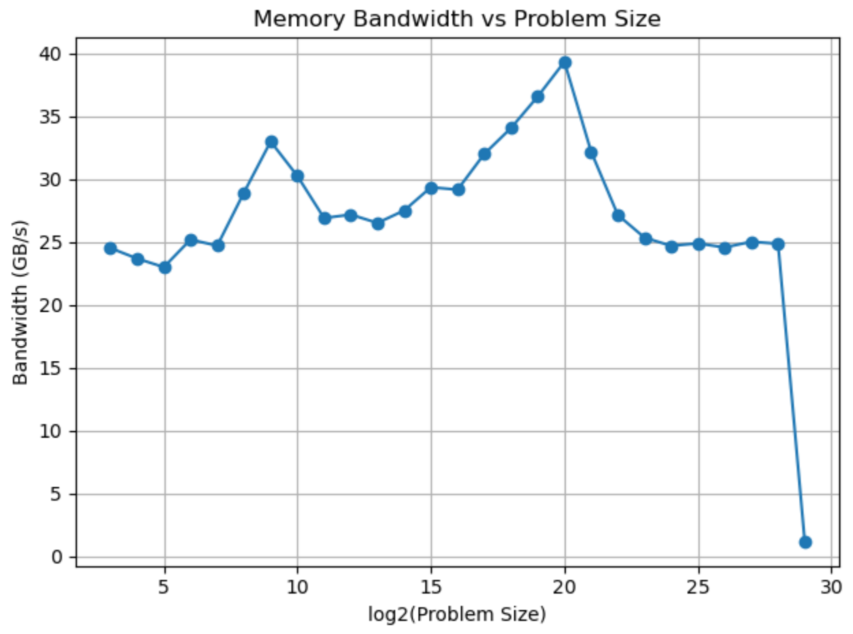

Figure 1: Memory Bandwidth (GB/s) vs. Problem Size for Copy kernel on Lab PC.

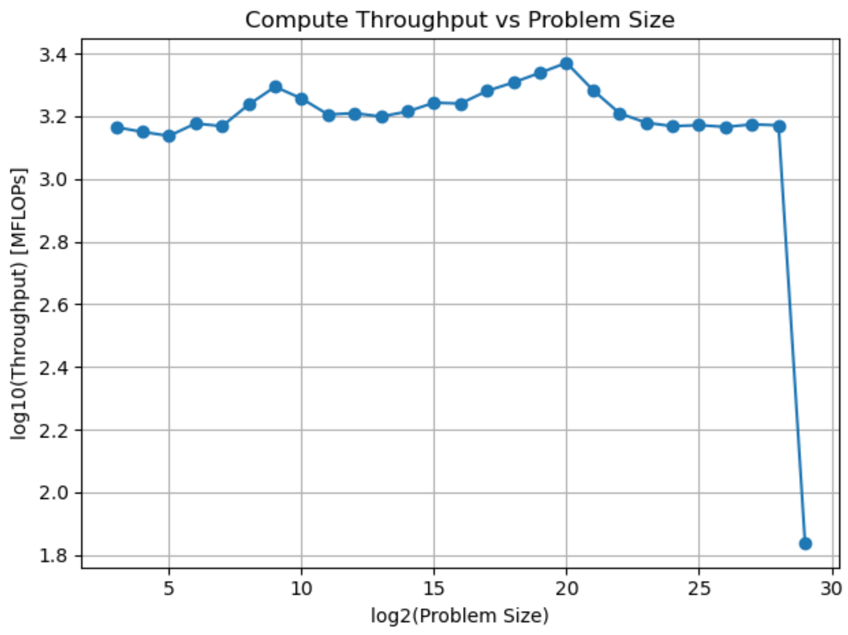

Figure 2: Compute Throughput (MFLOPs) vs. Problem Size for Copy kernel on Lab PC.

6.1.2 HPC Cluster Results

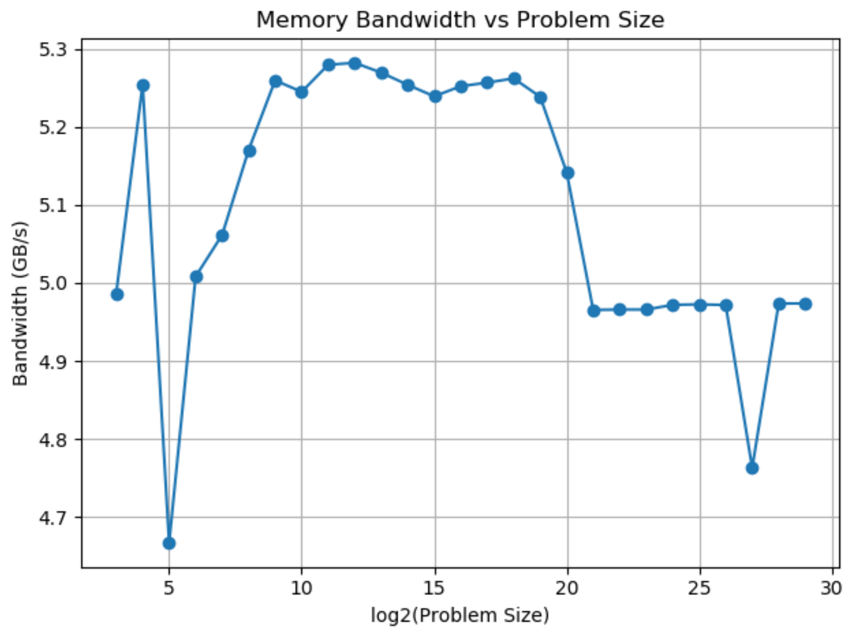

Figure 3: Memory Bandwidth (GB/s) vs. Problem Size for Copy kernel on HPC Cluster.

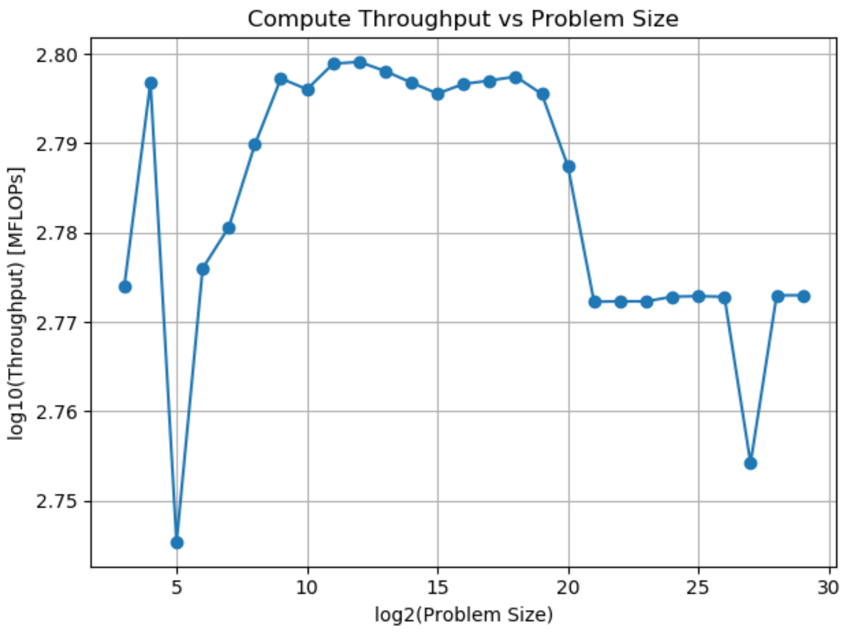

Figure 4: Compute Throughput (MFLOPs) vs. Problem Size for Copy kernel on HPC Cluster.

6.2 Vector Scale Operation: $a[i] = k \times b[i]$

6.2.1 Lab PC Results

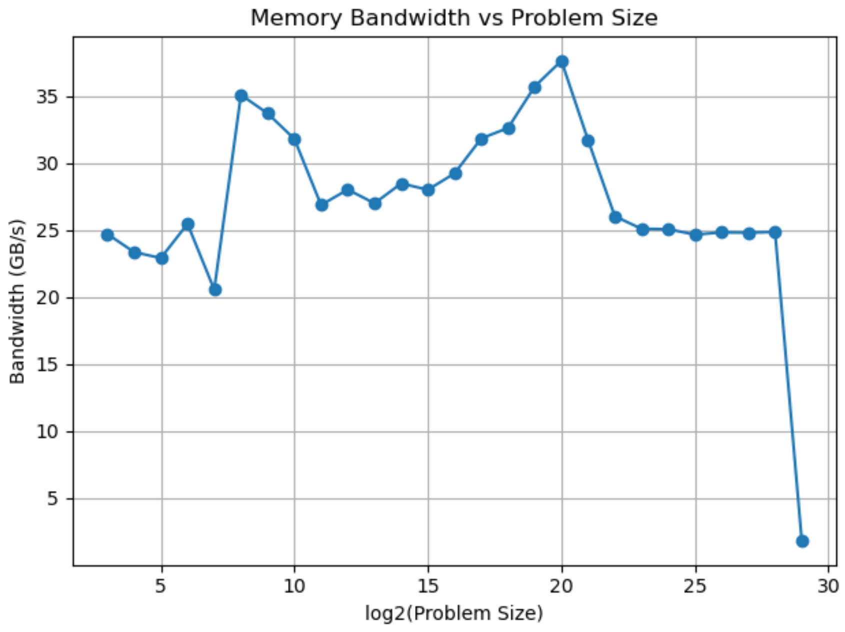

Figure 5: Memory Bandwidth (GB/s) vs. Problem Size for Scale kernel on Lab PC.

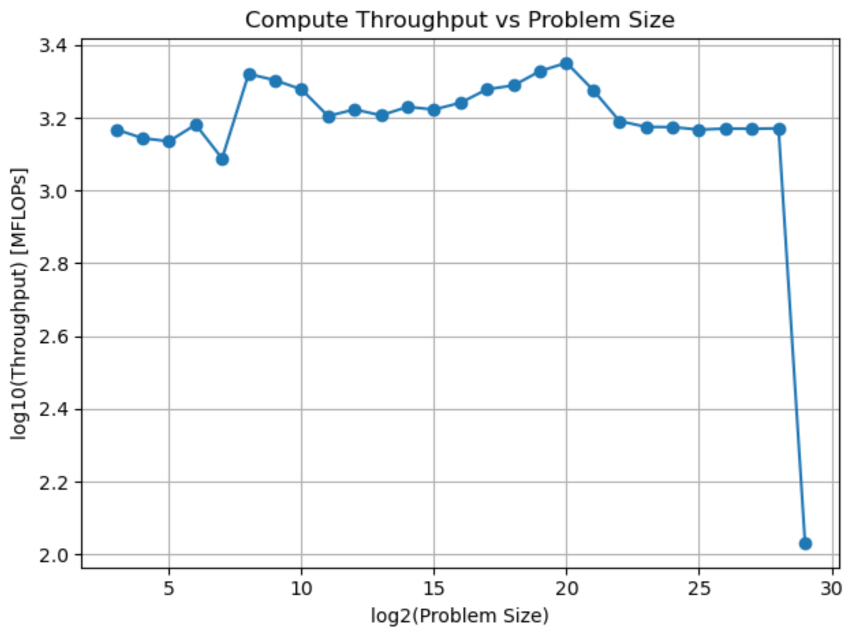

Figure 6: Compute Throughput (MFLOPs) vs. Problem Size for Scale kernel on Lab PC.

6.2.2 HPC Cluster Results

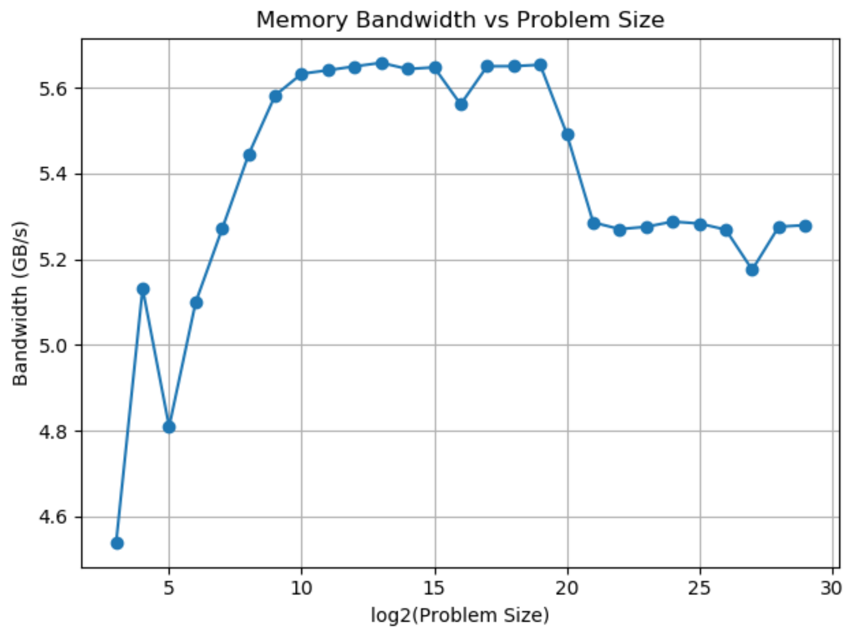

Figure 7: Memory Bandwidth (GB/s) vs. Problem Size for Scale kernel on HPC Cluster.

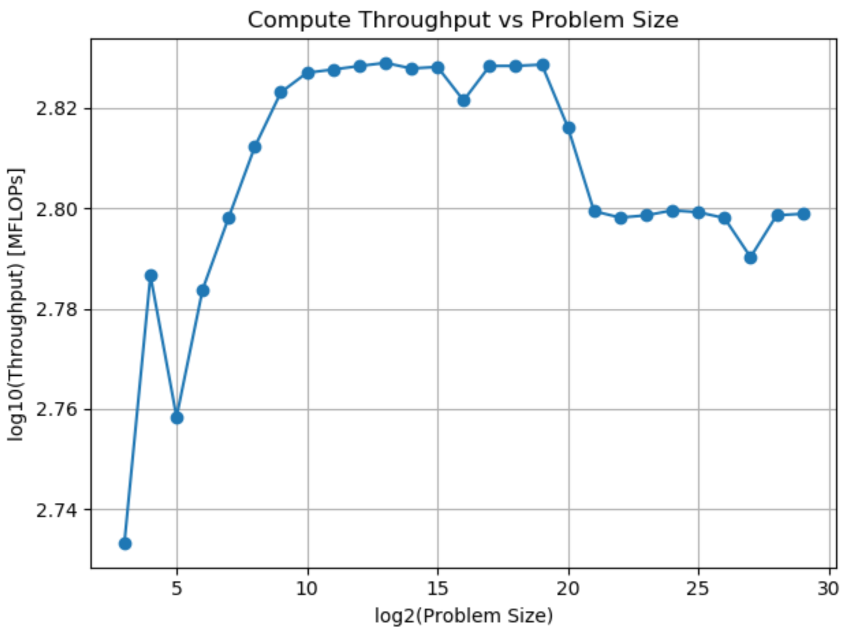

Figure 8: Compute Throughput (MFLOPs) vs. Problem Size for Scale kernel on HPC Cluster.

6.3 Vector Add Operation: $a[i] = b[i] + c[i]$

6.3.1 Lab PC Results

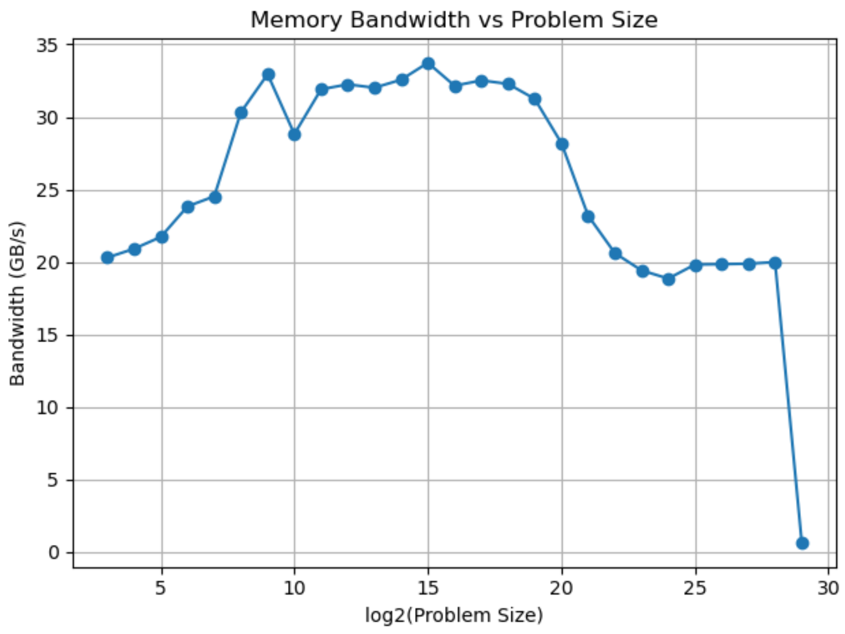

Figure 9: Memory Bandwidth (GB/s) vs. Problem Size for Add kernel on Lab PC.

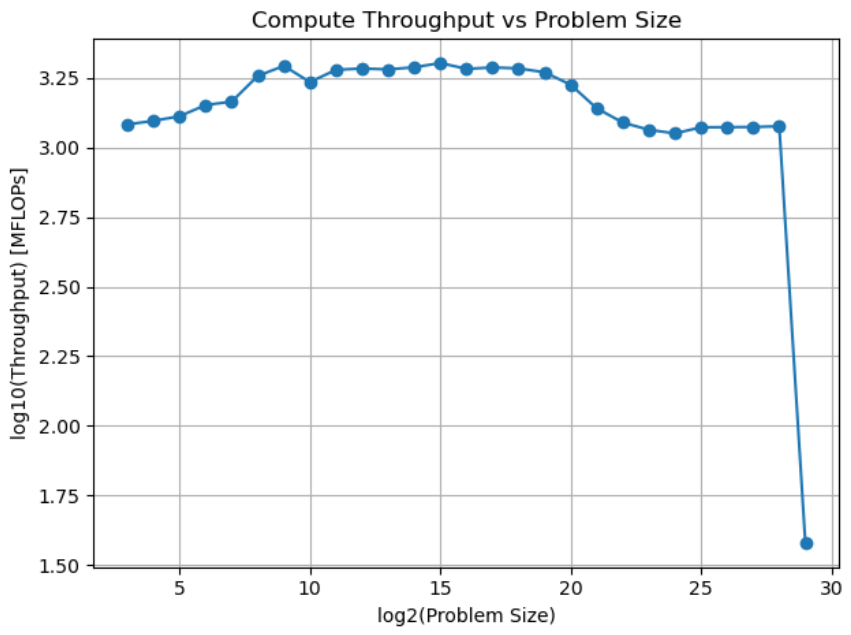

Figure 10: Compute Throughput (MFLOPs) vs. Problem Size for Add kernel on Lab PC.

6.3.2 HPC Cluster Results

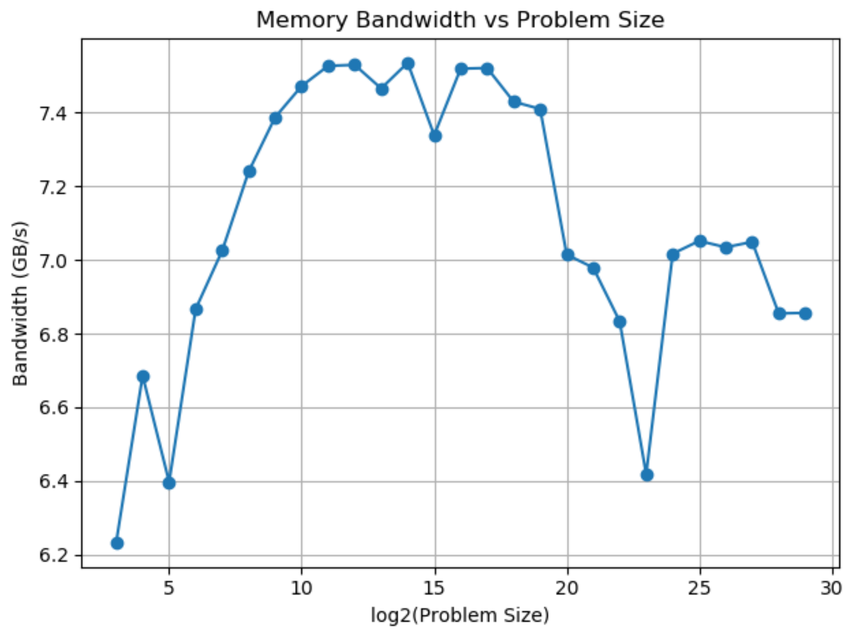

Figure 11: Memory Bandwidth (GB/s) vs. Problem Size for Add kernel on HPC Cluster.

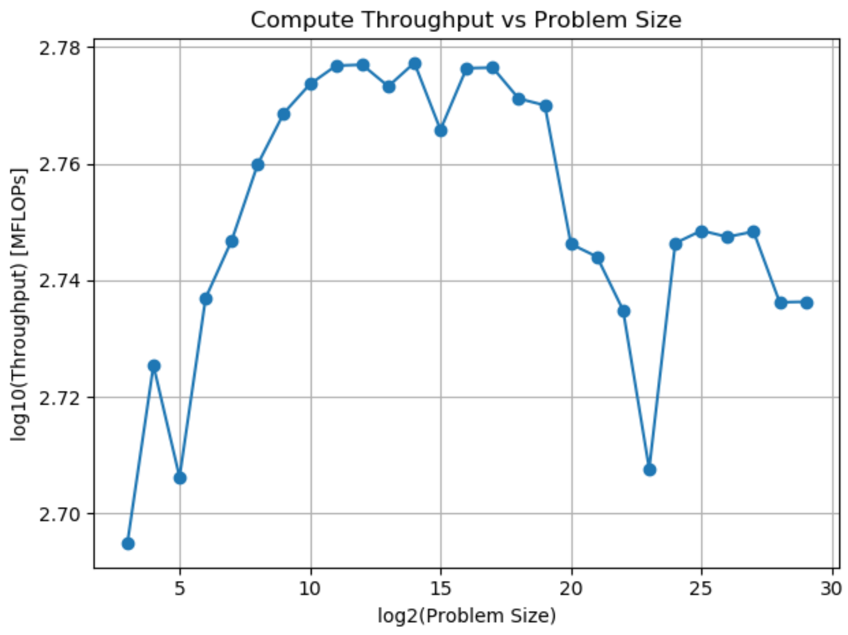

Figure 12: Compute Throughput (MFLOPs) vs. Problem Size for Add kernel on HPC Cluster.

6.4 Vector Triad Operation: $a[i] = b[i] + k \times c[i]$

6.4.1 Lab PC Results

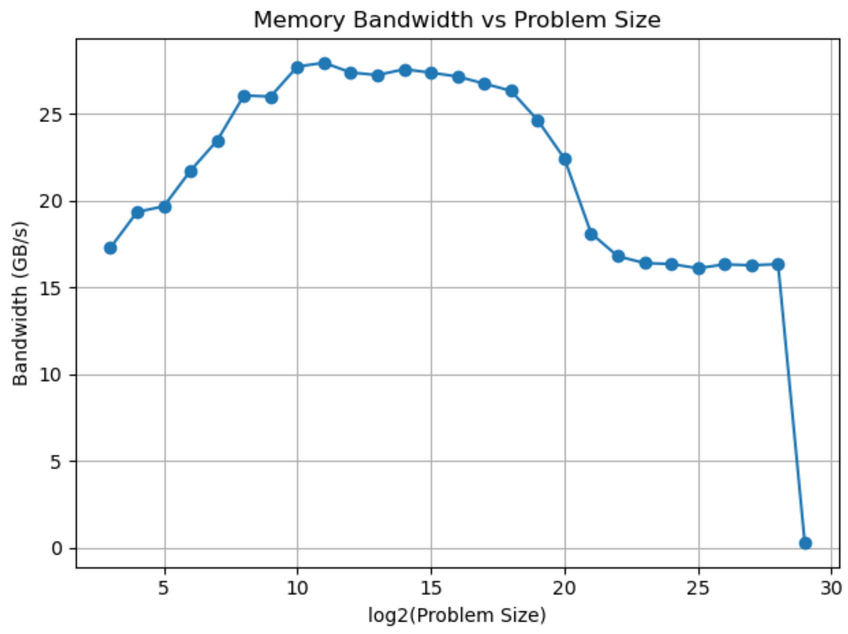

Figure 13: Memory Bandwidth (GB/s) vs. Problem Size for Triad kernel on Lab PC.

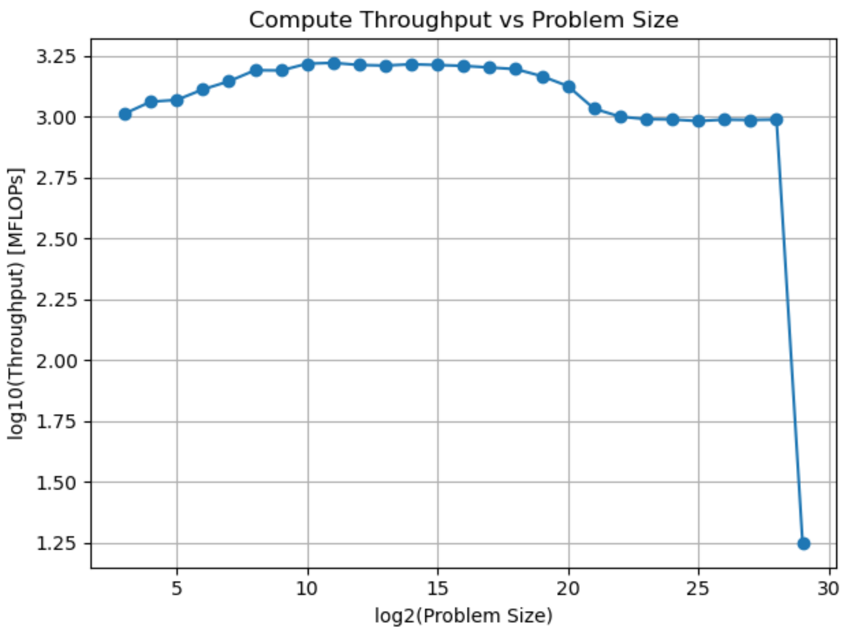

Figure 14: Compute Throughput (MFLOPs) vs. Problem Size for Triad kernel on Lab PC.

6.4.2 HPC Cluster Results

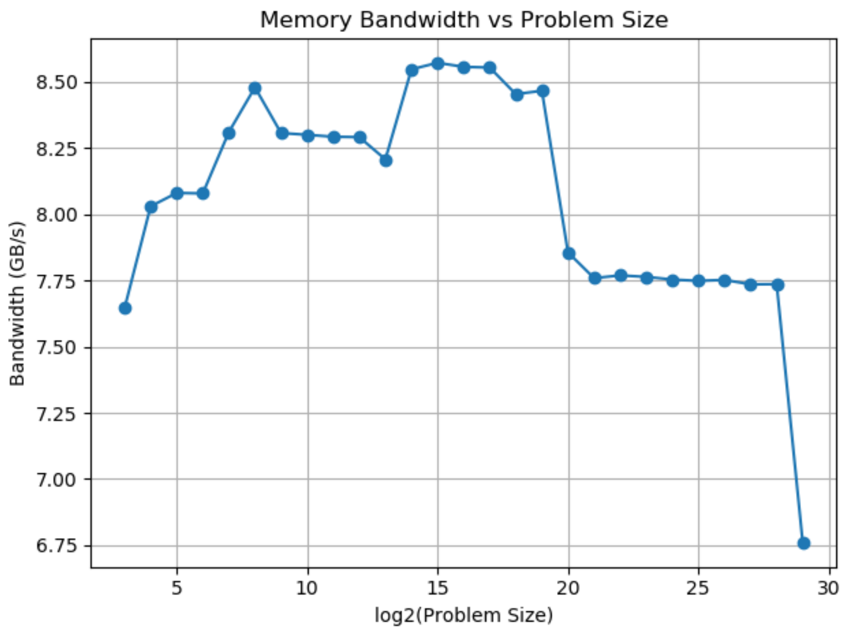

Figure 15: Memory Bandwidth (GB/s) vs. Problem Size for Triad kernel on HPC Cluster.

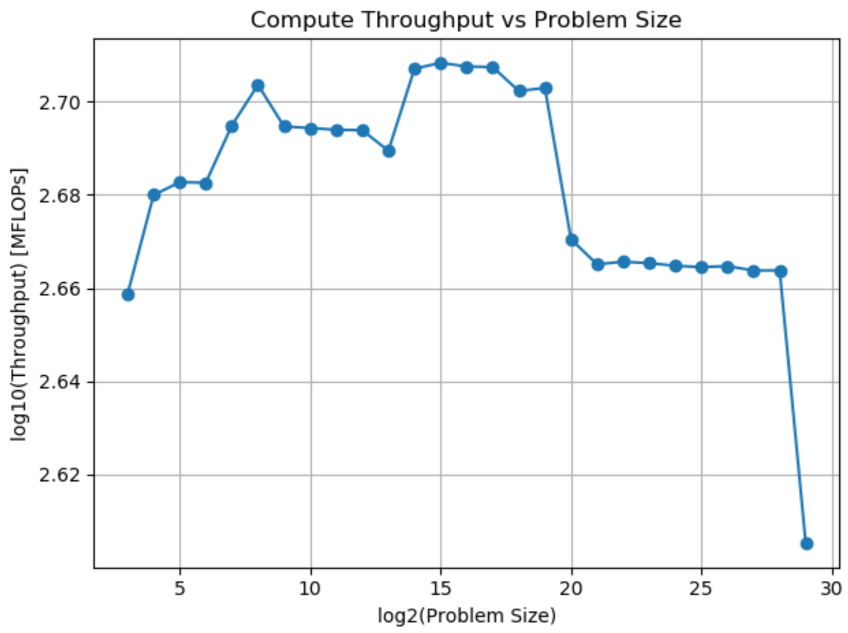

Figure 16: Compute Throughput (MFLOPs) vs. Problem Size for Triad kernel on HPC Cluster.

7 Analysis and Comparison

7.1 Comparison with Theoretical Peak Performance

In Part A, we calculated that the **theoretical peak memory bandwidth** for our Lab PC is **51.2 GB/s**. This represents the absolute maximum speed the hardware can achieve under ideal conditions.

After running our benchmarks, the highest bandwidth we measured was approximately **27.93 GB/s** during the Vector Triad operation. This corresponds to about **54.5%** of the theoretical peak performance.

There are several reasons why the measured bandwidth is lower than the theoretical maximum:

- **Memory Overhead:** The theoretical calculation does not account for memory controller overheads such as command scheduling and DRAM refresh cycles.
- **System Background Tasks:** The operating system and background processes consume a portion of the available memory bandwidth.
- **Data Movement:** The CPU must handle address calculations, cache coherence, and data consistency, introducing small delays in each memory operation.

7.2 Impact of Cache Sizes on Performance

The results clearly illustrate how different cache levels (L1, L2, and L3) influence performance. The observed bandwidth depends strongly on whether the working data set fits within these cache hierarchies.

1. **The Cache Region ($N_p \leq 2^{18}$):** For small problem sizes, the data fits entirely within the CPU caches (L1, L2, and the 18 MiB L3 cache on Lab PC or 15 MiB L3 on HPC Cluster). In this region, the performance remains high and stable. Since the CPU does not need to access main memory, data can be processed very efficiently.
2. **The Transition Region ($2^{18} < N_p < 2^{21}$):** As the number of elements increases, the data size grows from approximately 8 MiB to 64 MiB. Because the L3 cache capacity is limited, the working set no longer fits entirely in cache. This results in a noticeable drop in bandwidth.
3. **The RAM Region ($N_p \geq 2^{21}$):** For very large problem sizes, the CPU must access main memory for nearly every operation. The bandwidth stabilizes at a lower value, representing the sustained, real-world memory bandwidth of the system for large data sets.

The sharp performance drop observed at the final data point is likely due to the data size exceeding the available physical memory, causing the system to rely on disk swapping, which is significantly slower than RAM access.

7.3 Lab PC vs. HPC Cluster Comparison

Comparing the results between the Lab PC and HPC Cluster reveals interesting architectural differences:

- **Cache Architecture:** The Lab PC has a larger L3 cache (18 MiB) compared to the HPC Cluster node (15 MiB), which may contribute to better performance for medium-sized datasets that fit within the cache.
- **Multi-Socket Architecture:** The HPC Cluster features a dual-socket, dual-NUMA node configuration, which can provide higher aggregate bandwidth when workloads are optimized for NUMA-aware execution.
- **Performance Stability:** Both systems show similar performance patterns with clear transitions between cache and memory-bound regions, demonstrating the universal importance of memory hierarchy in HPC applications.

8 Conclusion

In this lab, we measured the performance of both our Lab PC and the HPC Cluster using the four STREAM benchmark operations. By testing different data sizes on both systems, we gained insights into how different hardware configurations handle varying workloads. Our tests showed that performance is very fast for small data sizes because the data fits inside the CPU's high-speed caches. However, once the data becomes larger than the L3 cache, the speed drops significantly because the system must use the slower main RAM.

We tested all four operations (Copy, Scale, Add, and Triad) on both the Lab PC and 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.

The comparison between the Lab PC and HPC Cluster revealed that while both systems exhibit similar memory hierarchy effects, the HPC Cluster's dual-socket architecture and NUMA configuration offer potential advantages for large-scale parallel workloads. We also found that our actual measured bandwidth was lower than the theoretical peak of 51.2 GB/s due to real-world factors like memory delays and system overhead. Overall, this lab demonstrates that the speed of moving data through the memory hierarchy is often more important than the calculation speed of the CPU, and understanding these performance characteristics is crucial for optimizing high-performance computing applications.