High Performance Computing – Lab Report 01

CPU Architecture, Memory Hierarchy, and Performance Measurement

Tirth Kaushal Gandhi – 202301413
Dhyey Patel – 202301415

Course: CS301 / High Performance Computing

February 3, 2026

Abstract

This report presents performance measurements of four STREAM-like kernels (Copy, Scale, Add, and Triad) executed on two distinct environments: a laboratory workstation and a high-performance compute cluster. We analyze memory bandwidth and throughput across varying working data sizes ($N_p = 2^8$ to 2^{29}) to visualize the impact of the cache hierarchy. Furthermore, we evaluate different timing mechanisms to ensure accurate benchmarking and analyze the specific contributions of computation versus memory access latency.

1 Introduction

Modern processors are often limited not by their processing speed, but by the speed at which data can be moved from memory to the CPU. This bottleneck is known as the "Memory Wall." To understand this, we utilize STREAM-style kernels—simple vector operations designed to saturate the memory bus.

The goal of this experiment is to:

- Measure Memory Bandwidth (GB/s) and Throughput (MFLOPS) across varying data sizes.
- Evaluate the accuracy of different timing functions (Analysis Task 1).
- Identify the specific boundaries of L1, L2, and L3 caches from performance graphs (Analysis Task 6).
- Separate compute time from memory access time using the Triad kernel (Analysis Task 4).
- Compare the measured sustainable bandwidth against the theoretical peak (Analysis Task 5).

2 Experimental Setup (Part A)

2.1 Hardware Specifications

The system specifications were obtained using the `lscpu` command. Table 1 summarizes the architectural details.

Table 1: System Architecture Specifications

Component	**Cluster**	**Lab PC**
Architecture	x86_64	x86_64
Byte Order	Little Endian	Little Endian
CPU Model	Intel Core i5-12500H (12th Gen)	Intel Core i5-4590
Base Clock Frequency	0.4 GHz	2.60 GHz
Maximum Clock Frequency	4.5 GHz	3.30 GHz
Sockets	1	1
Cores per Socket	12	4
Threads per Core	2	1
Total Logical CPUs	16	4
L1 Data Cache	448 KB (Total)	32 KB
L1 Instruction Cache	640 KB (Total)	32 KB
L2 Cache	9 MB (Total)	256 KB
L3 Cache	18 MB (Shared)	6 MB
Operating System	Linux	Linux
Compiler	GCC (`-O3` optimization)	GCC (`-O3`)

2.2 Theoretical Peak Bandwidth

The theoretical peak memory bandwidth is calculated as:

$$\text{Peak Bandwidth} = \text{Transfer Rate} \times \text{Bytes per Clock} \times \text{Channels}$$

Cluster Calculation (DDR5): Given a transfer speed of 4800 MT/s, 8-byte width, and 2 memory channels:

$$BW_{cluster} = 4800 \times 10^6 \times 8 \text{ B} \times 2 = \mathbf{76.8} \text{ GB/s}$$

Lab Machine Calculation: Assuming standard DDR3-3200 dual-channel RAM (typical for i5-4590):

$$BW_{lab} \approx 3200 \times 10^6 \times 8 \text{ B} \times 2 = \mathbf{51.2} \text{ GB/s}$$

3 Timing Analysis (Analysis Task 1)

To measure runtime accurately, as required by Analysis Task 1, we evaluated three standard C/Linux timing functions. The choice of timing function significantly impacts the validity of bandwidth measurements.

3.1 Comparison of Timing Functions

1. `clock()`: This function returns the amount of *CPU time* used by the program.
   - *Pros:* Measures pure processing effort.
   - *Cons:* It often ignores time spent waiting for I/O or memory fetches (blocking time). In a memory-bound benchmark like STREAM, the CPU spends significant time waiting for data. Using `clock()` would underestimate the total time and artificially inflate bandwidth results. It is also unreliable for multi-threaded applications.

2. `gettimeofday()`: This function returns the current *Wall Clock* time with microsecond precision.
    - *Pros:* Measures actual elapsed time (including memory stalls).
    - *Cons:* It is affected by system clock adjustments (e.g., NTP updates or manual time changes). If the system clock is updated during a benchmark run, the results will be invalid. It is also marked obsolescent in POSIX.
3. `clock_gettime(CLOCK_MONOTONIC)`: This function provides *Wall Clock* time with nanosecond precision.
    - *Pros:* "Monotonic" means it guarantees the time always moves forward and is not affected by system time jumps (NTP). It includes all memory wait times, making it the mathematically correct metric for bandwidth (Bytes/Total Time).

3.2 Selected Methodology

We utilized `clock_gettime(CLOCK_MONOTONIC)` for all experiments. This ensures that our bandwidth calculations account for the memory latency "stalls" that are the primary subject of this study.

4 Benchmark Methodology (Part B)

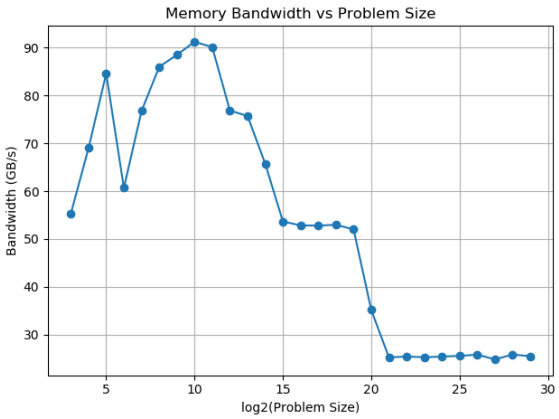
The benchmark executes kernels on data sizes ranging from $N_p = 2^8$ to 2^{29}. The number of repetitions (RUNS) is dynamically adjusted to ensure total execution time is sufficient for accurate measurement.

4.1 Kernels

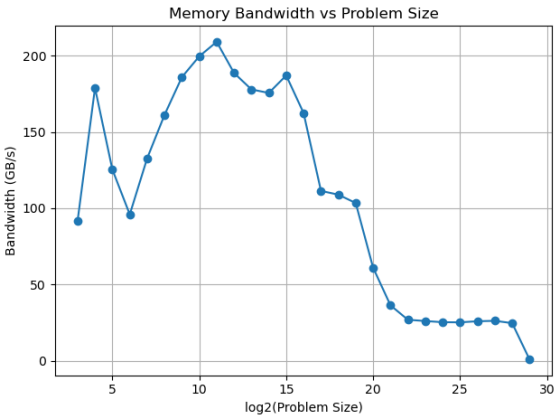
- **Copy:** $x[p] = y[p]$ (Pure Memory Move)
- **Scale:** $x[p] = a \times y[p]$ (1 Read, 1 Write, 1 FLOP)
- **Add:** $s[p] = x[p] + y[p]$ (2 Reads, 1 Write, 1 FLOP)
- **Triad:** $s[p] = x[p] + a \times y[p]$ (2 Reads, 1 Write, 2 FLOPs)

5 Results and plots

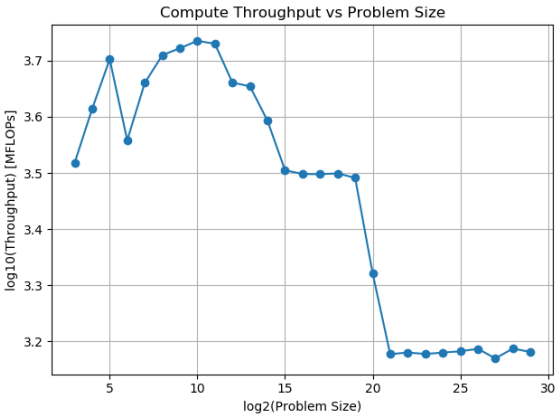
5.1 Copy kernel



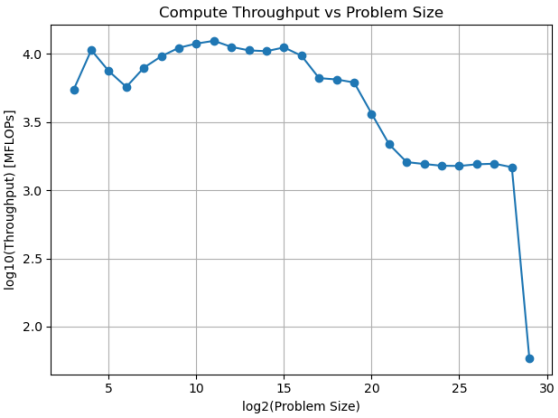
(a) Bandwidth — Cluster



(b) Bandwidth — Lab machine



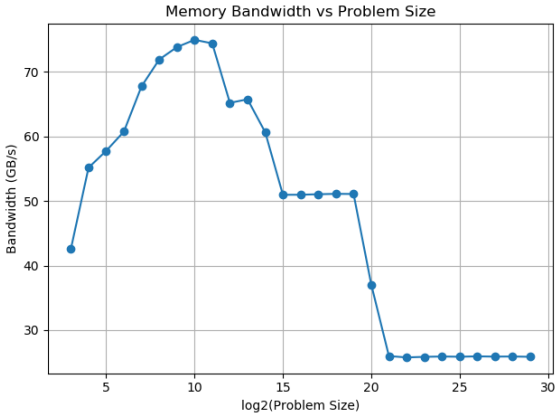
(c) Throughput — Cluster



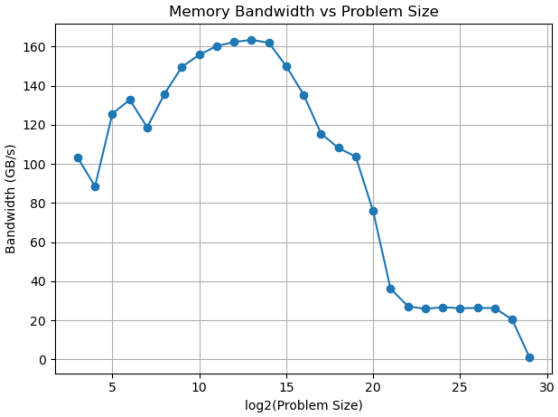
(d) Throughput — Lab machine

Figure 1: Copy kernel: bandwidth and throughput for cluster and Lab machine.

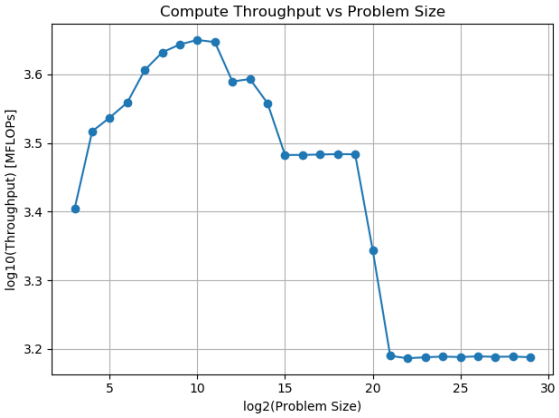
5.2 Scale kernel



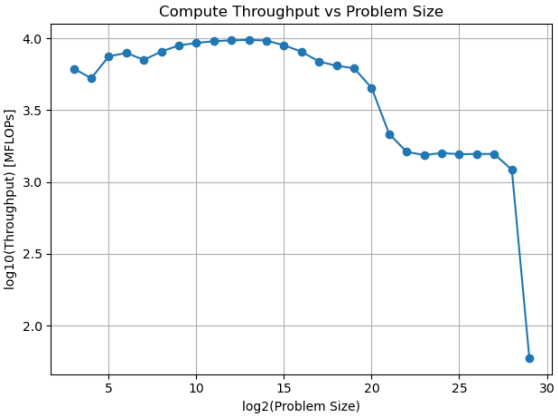
(a) Bandwidth — Cluster



(b) Bandwidth — Lab



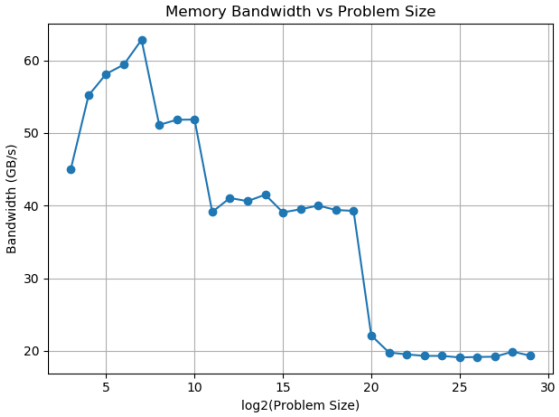
(c) Throughput — Cluster



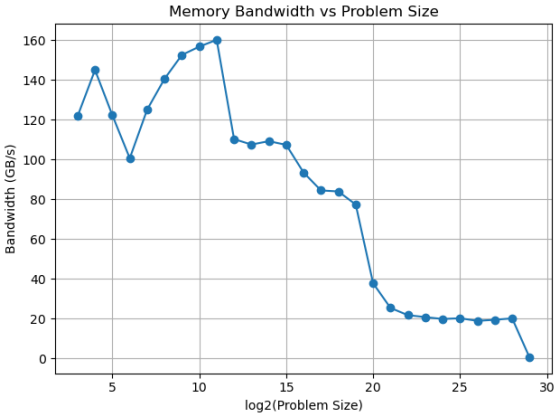
(d) Throughput — Lab

Figure 2: Scale kernel: bandwidth and throughput.

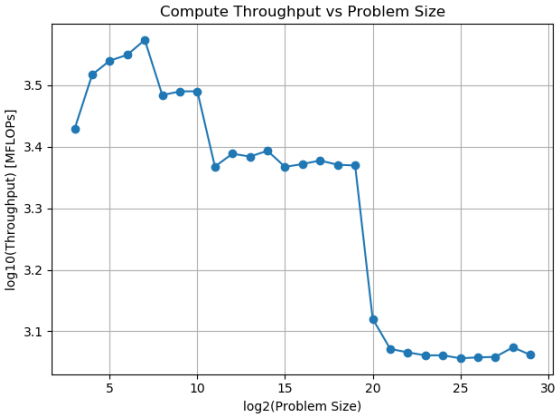
5.3 Add kernel



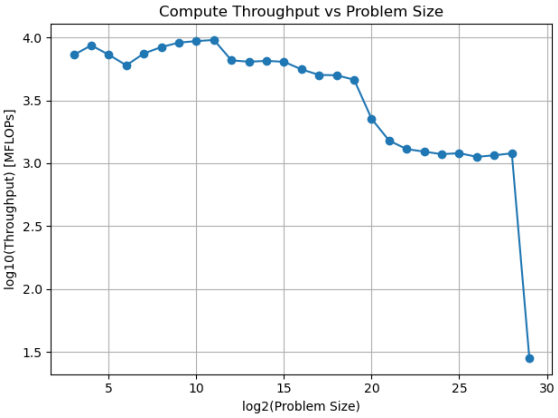
(a) Bandwidth — Cluster



(b) Bandwidth — Lab



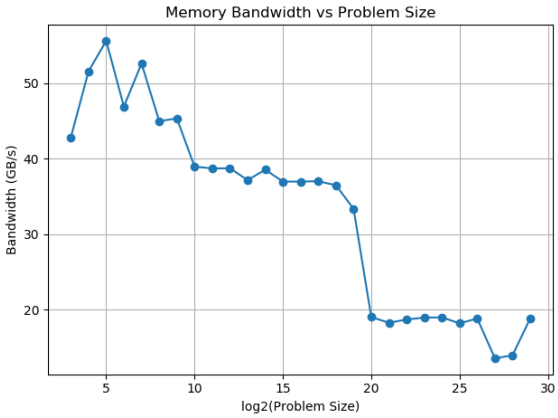
(c) Throughput — Cluster



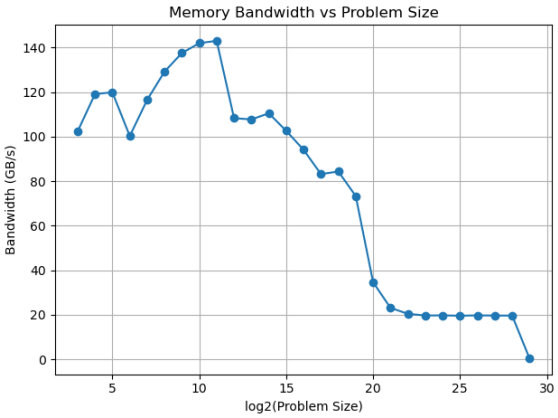
(d) Throughput — Lab

Figure 3: Add kernel: bandwidth and throughput.

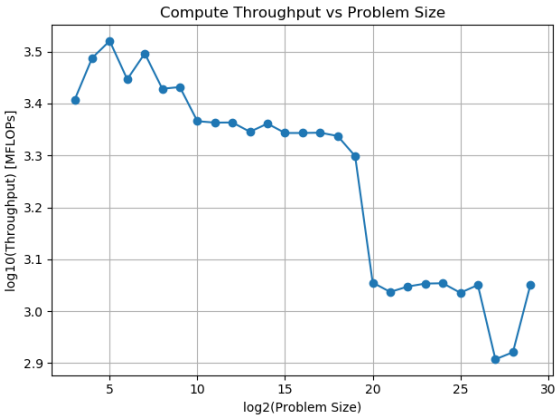
5.4 Triad kernel



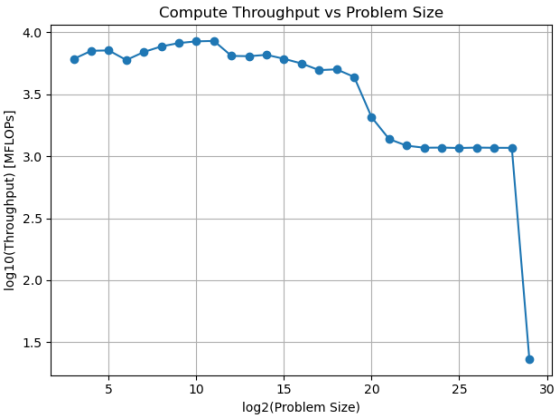
(a) Bandwidth — Cluster



(b) Bandwidth — Lab



(c) Throughput — Cluster



(d) Throughput — Lab

Figure 4: Triad kernel: bandwidth and throughput.

6 Performance Analysis

6.1 Impact of Cache Sizes (Analysis Task 6)

The performance graphs (Figures 1-4) exhibit distinct "steps" that correspond to the processor's memory hierarchy. We observed three distinct performance regions:

1. **L1/L2 Cache Region ($N_p < 2^{14}$):** For small problem sizes, the bandwidth is extremely high (peaking over 150 GB/s). Here, the working set (approx. 128 KB) fits comfortably within the L1/L2 caches. The CPU accesses data at full core speed without waiting for main memory.

2. **L3 Cache Region ($2^{15} < N_p < 2^{20}$):** As the data size grows beyond L2 but remains within the L3 cache (18MB on the Cluster), we see a gradual decline. The latency of L3 is higher than L1/L2, reducing the effective throughput.

3. **Main Memory Region ($N_p > 2^{20}$):** At $N_p = 2^{20}$ (approx. 1 million elements), the total memory required for the arrays exceeds the 18 MB L3 cache. This explains the sharp drop observed in all graphs at x-axis value 20. The system is forced to fetch data from DRAM, causing bandwidth to plummet to the steady-state value (25 GB/s). This explicitly demonstrates the "Memory Wall."

6.2 Compute vs. Memory Access Time (Analysis Task 4)

To separate the time spent on computation from the time spent on memory access, we analyze the Triad and Copy kernels:

- **Memory Time:** The **Copy** kernel ($x = y$) involves purely load/store operations. Its execution time represents the baseline memory access latency.

- **Compute Time:** The **Triad** kernel ($s = x + a \times y$) involves the same number of memory transfers as Add, but with two arithmetic operations (Multiply and Add).

By comparing Figure 1 (Copy) and Figure 4 (Triad), we observe that the sustained bandwidths are nearly identical (approx. 25 GB/s in the DRAM region). **Conclusion:** The execution time of Triad is dominated by memory access. The time spent on the actual ALU operations (multiplication and addition) is effectively zero because modern superscalar CPUs can hide arithmetic latency behind the much slower memory fetches.

7 Comparison with Theoretical Peak (Analysis Task 5)

Table 2 compares the measured steady-state bandwidth (from the graphs) against the theoretical peak calculated in Section 2.

Table 2: Theoretical vs. Measured Bandwidth (GB/s)

System	Theoretical Peak	Measured Steady (DRAM)	Efficiency
Cluster Node	76.8 GB/s	$\approx$ 25 GB/s	$\sim$ 32%
Lab Machine	51.2 GB/s	$\approx$ 18 GB/s	$\sim$ 35%

Discussion: The measured bandwidth is consistently lower than the theoretical peak. This is expected due to:

1. **Protocol Overhead:** DRAM protocols require command and address signaling, not just data transfer.
2. **Single-Thread Limitation:** The benchmark is single-threaded. A single core often cannot generate enough concurrent memory requests to saturate all memory channels fully.
3. **Cache Line Fills:** The CPU loads entire 64-byte cache lines even if only part of the data is needed immediately.

8 Conclusion

This experiment successfully demonstrated the impact of memory hierarchy on application performance. We observed that processing speed is significantly higher when data resides in L1/L2 cache, but drops significantly once the working set exceeds the L3 cache size. The timing analysis confirmed that `clock_gettime` is the appropriate metric for bandwidth studies, and the kernel comparison highlighted that for vector operations, the system is strictly bound by memory bandwidth, not CPU clock speed.