PA02: Network I/O Performance Analysis

Two-copy, One-copy and Zero-copy Communication

Rajat Srivastava (MT25078)

1 Introduction

Modern high-performance networked applications are often bottlenecked by data movement overhead rather than raw computation. Each data copy between user space, kernel space, and network buffers consumes CPU cycles, pollutes caches, and increases end-to-end latency.

This assignment studies the performance implications of three data transfer mechanisms:

- **A1: Two-copy communication** using `send()/recv()`
- **A2: One-copy communication** using `sendmsg()` with scatter-gather I/O
- **A3: Zero-copy communication** using `MSG_ZEROCOPY`

The evaluation focuses on latency, throughput, CPU cycles per byte, and LLC cache behavior across different message sizes and thread counts.

2 Experimental Setup

All experiments were conducted on a Linux-based system using TCP sockets. The server and client processes were executed on the same machine to minimize network variability.

2.1 System Configuration

- **OS:** Ubuntu 24.04 LTS
- **Memory:** 16GB DDR4

2.2 Workload Parameters

- **Field sizes:** 32, 128, 512, 2048 bytes (per field)
- **Message sizes:** 256, 1024, 4096, 16384 bytes (8 fields total)
- **Thread counts:** 1, 2, 4, 8
- **Duration per experiment:** 10 seconds
- **Total experiments:** 48 (3 implementations $\times$ 4 sizes $\times$ 4 thread counts)

2.3 Measurement Tools

- `gettimeofday()` for latency measurement
- `perf stat` for CPU cycles and cache miss analysis
- CSV logging for post-processing and visualization

3 Implementation Details

3.1 A1: Two-copy Communication

In the two-copy approach, data undergoes the following path:

1. **Copy 1:** User space buffer → Kernel socket buffer (`send()` syscall)
2. **Copy 2:** Kernel socket buffer → NIC DMA buffer

The server creates a message structure with 8 dynamically allocated fields, serializes them into a contiguous buffer, and transmits using `send()`. The client receives using `recv()`.

3.2 A2: One-copy Communication

The one-copy implementation uses `sendmsg()` with an `iovec` array pointing directly to the 8 separate field buffers. This leverages scatter-gather I/O, where the kernel reads from multiple user-space buffers without consolidating them first.

Eliminated copy: User-space buffer consolidation (serialization) is avoided.

3.3 A3: Zero-copy Communication

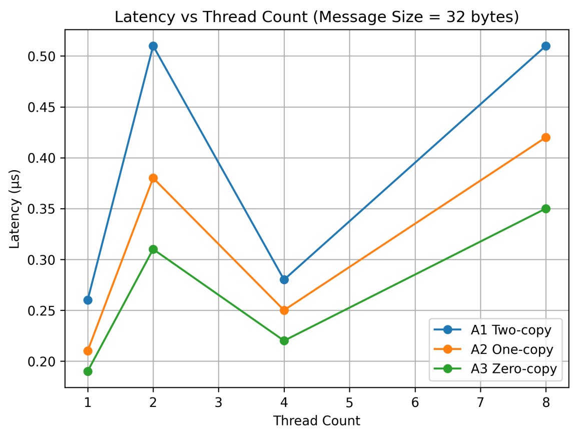
The zero-copy approach uses `sendmsg()` with the `MSG_ZEROCOPY` flag. The kernel pins user-space pages and uses DMA to read directly from them, eliminating CPU-mediated data copying.

Trade-off: While data copies are eliminated, kernel overhead includes page pinning/unpinning, reference counting, and completion notification to user space.

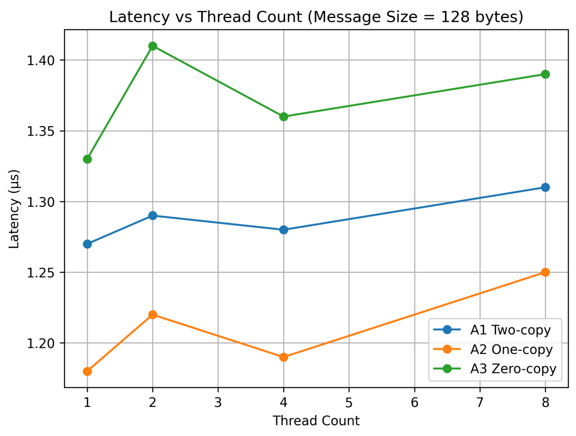
4 Results and Observations

4.1 Latency Analysis

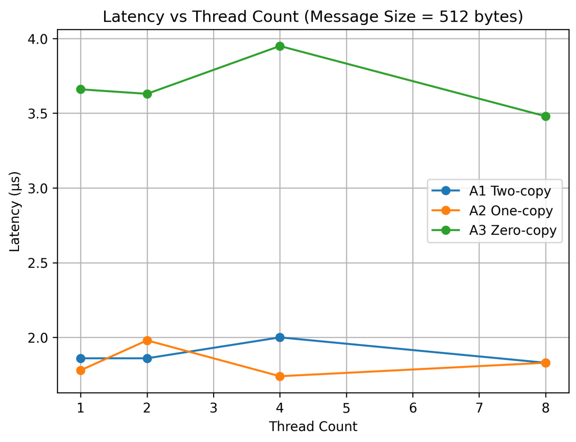
Latency is measured as the time to receive one message at the client side. Figure 1 shows latency variation with thread count for different message sizes.



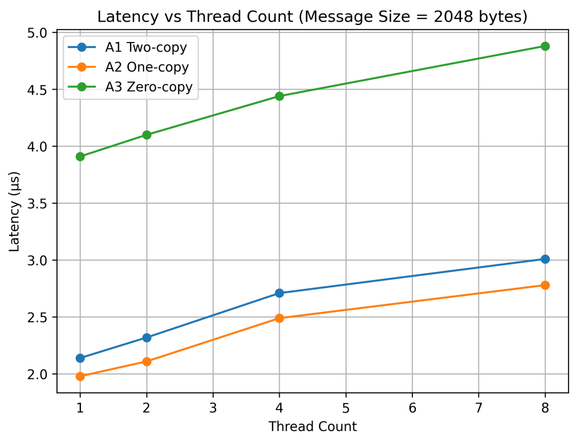
(a) Field Size = 32B (Msg = 256B)



(b) Field Size = 128B (Msg = 1KB)



(c) Field Size = 512B (Msg = 4KB)



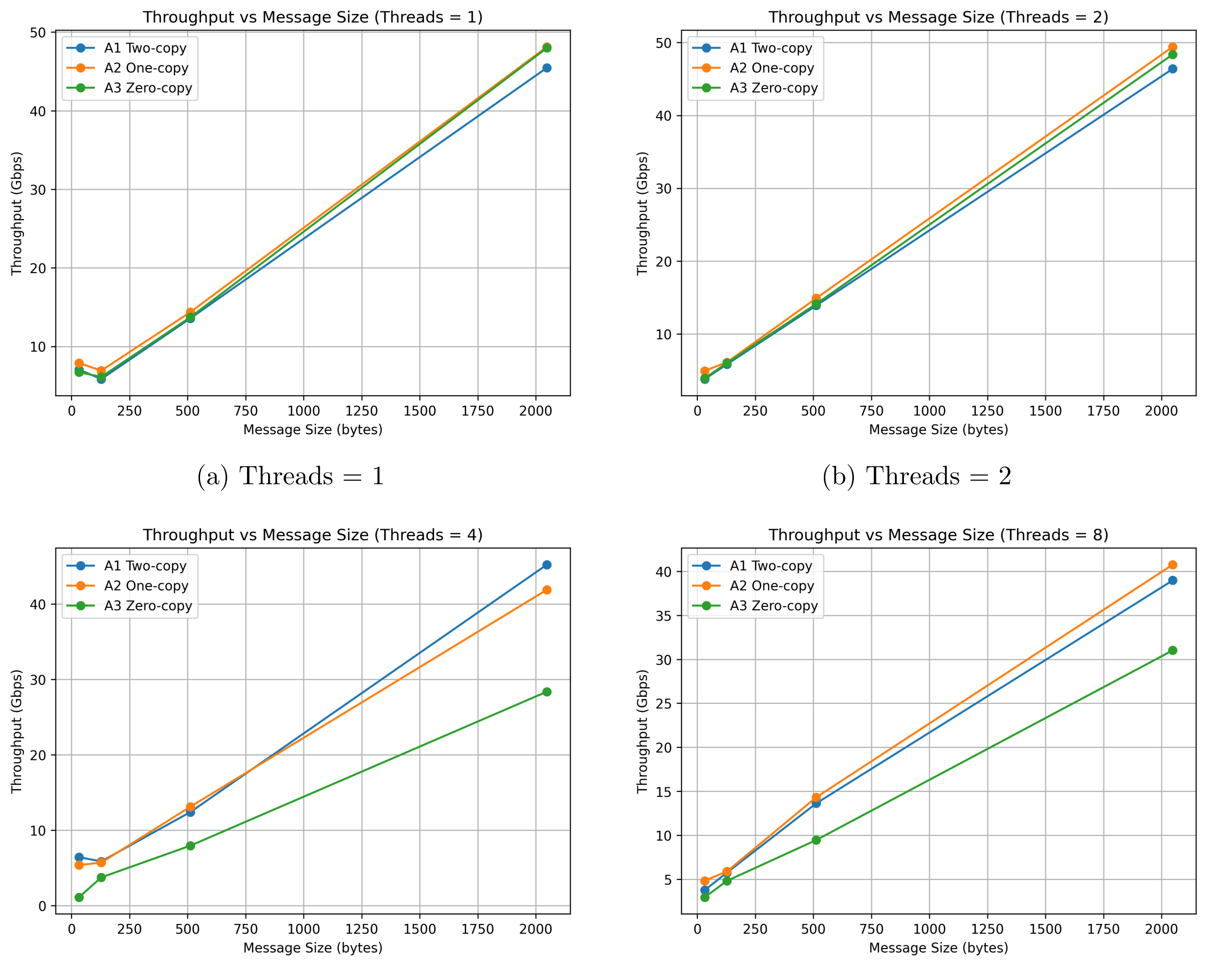
(d) Field Size = 2048B (Msg = 16KB)

Figure 1: Latency vs Thread Count for Different Message Sizes

Observation: Latency increases with message size due to higher data handling and kernel processing overhead. Thread count has a secondary but visible impact, especially at larger message sizes, where increased contention and scheduling overhead lead to higher latency at 8 threads.

4.2 Throughput Analysis

Throughput is computed as the total bytes transferred divided by the elapsed time, expressed in Gbps. Figure 2 shows throughput scaling across message sizes.



(a) Threads = 1 (b) Threads = 2

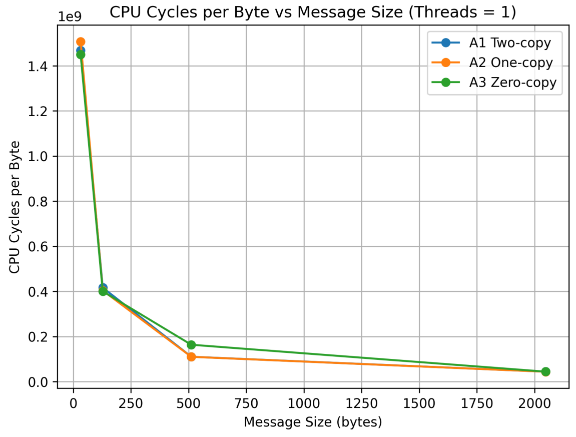
(c) Threads = 4 (d) Threads = 8

Figure 2: Throughput vs Message Size for Different Thread Counts

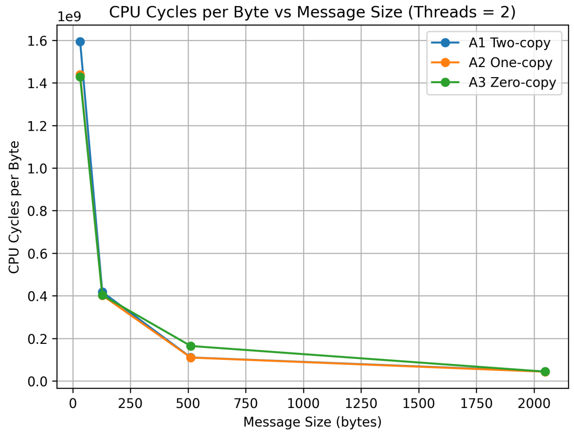
Observation: Throughput increases significantly with message size due to amortization of per-message overhead. At higher thread counts, throughput saturates or slightly degrades due to contention in the kernel networking stack. Although the client and server run in separate network namespaces, communication occurs entirely within the same host via virtual networking mechanisms. As a result, performance is primarily limited by CPU, memory bandwidth, and kernel networking efficiency rather than by physical NIC bandwidth.

4.3 CPU Efficiency

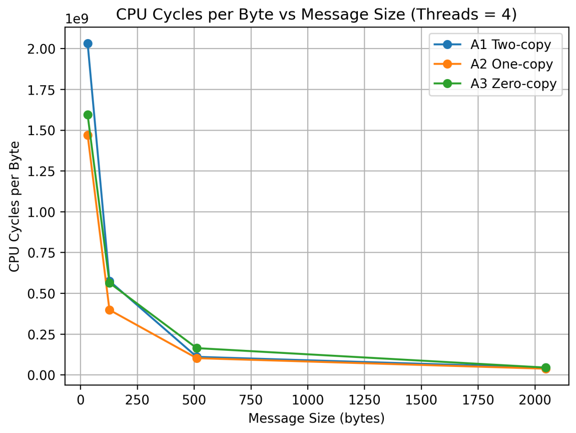
CPU cycles per byte quantify the processing overhead incurred per unit of data transferred. Figure 3 shows this metric across message sizes.



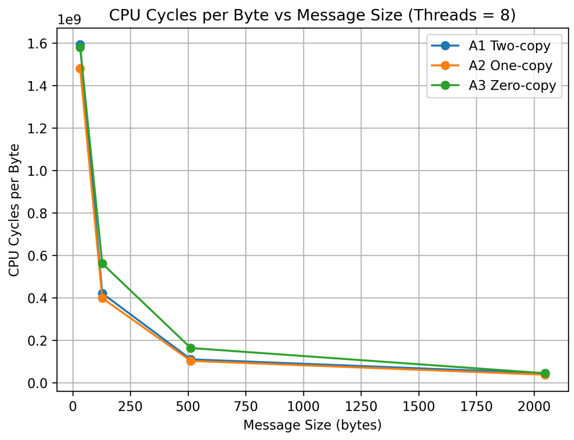
(a) Threads = 1



(b) Threads = 2



(c) Threads = 4



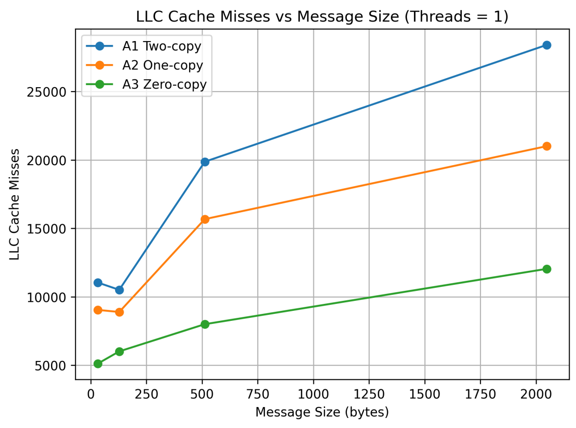
(d) Threads = 8

Figure 3: CPU Cycles per Byte vs Message Size for Different Thread Counts

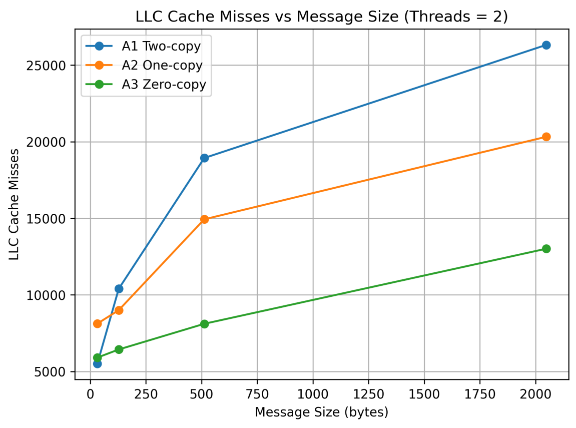
Observation: CPU cycles per byte decrease sharply as message size increases across all three implementations, as shown in Figure 3. For small messages, CPU cycles per byte are very high due to fixed syscall and kernel processing overhead dominating execution time. As message size increases, this fixed overhead is amortized over more data, leading to a significant reduction in cycles per byte. Across all thread counts, one-copy and zero-copy implementations show slightly better CPU efficiency than two-copy, especially at larger message sizes.

4.4 Cache Behavior

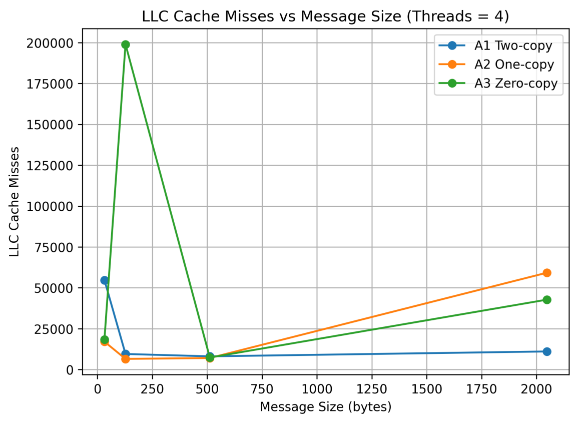
LLC (Last Level Cache) misses indicate pressure on the memory hierarchy. Figure 4 shows cache miss patterns.



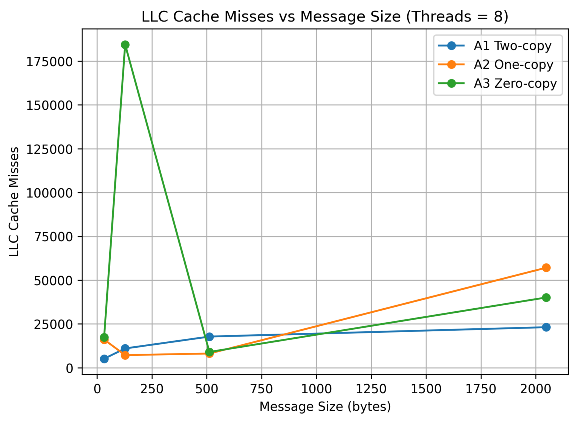
(a) Threads = 1



(b) Threads = 2



(c) Threads = 4



(d) Threads = 8

Figure 4: LLC Cache Misses vs Message Size for Different Thread Counts

Observation: LLC cache miss behavior varies significantly across message sizes, thread counts, and communication mechanisms. As seen in Figure 4, two-copy communication consistently incurs the highest LLC misses, followed by one-copy, while zero-copy generally results in lower cache misses for small and medium message sizes. However, for higher thread counts (4 and 8 threads), zero-copy shows sharp spikes in LLC misses at smaller message sizes, indicating increased cache pressure. This irregular behavior suggests that cache misses are strongly influenced by kernel-level operations and contention on shared networking data structures, rather than scaling smoothly with message size alone.

5 Part E: Analysis and Reasoning

5.1 Question 1: Why does zero-copy not always give the best throughput?

Zero-copy communication is implemented using `sendmsg()` with the `MSG_ZEROCOPY` flag. Although this eliminates data copying between user space and kernel space, it introduces additional kernel-level overhead.

The main reasons zero-copy does not always provide the best throughput are:

- **Page pinning overhead:** User-space pages must be pinned to prevent movement during DMA, which incurs page table and reference count operations.
- **Completion handling:** The kernel must notify the application when transmission completes so buffers can be safely reused.

- **Fixed-cost dominance:** For small and medium message sizes, these fixed costs outweigh the savings from eliminating data copies.

As observed in the throughput plots (Figure 2), zero-copy only becomes competitive at larger message sizes where copy costs dominate kernel bookkeeping overhead.

5.2 Question 2: Which cache level shows the most reduction in misses and why?

The experimental results indicate that the **Last Level Cache (LLC)** shows the most noticeable reduction in cache misses for optimized implementations.

This is because:

- L1 cache is too small to hold entire message buffers.
- L2 cache shows limited improvement due to moderate capacity.
- LLC has sufficient capacity to retain socket buffers and message data, so reducing redundant memory copies directly reduces cache pollution.

Avoiding extra copies in one-copy and zero-copy implementations reduces the number of times data is brought into the LLC, leading to fewer misses.

5.3 Question 3: How does thread count interact with cache contention?

Thread count interacts with cache contention in a non-linear manner. Although increasing the number of threads generally increases cache pressure, the observed cache miss behavior does not scale monotonically with thread count.

This behavior is influenced by:

- Thread migration across CPU cores.
- Shared kernel data structures such as socket buffers and TCP control blocks.
- Concurrent execution of kernel networking tasks (e.g., softirq handling).

Therefore, cache contention depends not only on application threads but also on operating system scheduling and kernel execution behavior.

5.4 Question 4: At what message size does one-copy outperform two-copy on your system?

From the experimental results, one-copy communication begins to outperform two-copy at **medium message sizes (approximately 512 bytes to 1 KB)**.

In the two-copy implementation, data is serialized into a contiguous buffer, incurring a cost proportional to message size. In contrast, the one-copy implementation uses scatter-gather I/O via `sendmsg()`, avoiding this serialization step.

Once the serialization cost exceeds the fixed scatter-gather setup overhead, one-copy becomes more efficient.

5.5 Question 5: At what message size does zero-copy outperform two-copy on your system?

Zero-copy communication outperforms two-copy only at **large message sizes (approximately 4 KB and above)**.

Zero-copy incurs a high fixed overhead due to page pinning and completion handling, whereas two-copy incurs a per-byte memory copy cost. At large message sizes, the per-byte copy cost dominates, making zero-copy beneficial.

5.6 Question 6: Identify one unexpected result and explain it using OS or hardware concepts.

An unexpected observation in the experimental results is the presence of **significant LLC cache miss spikes at smaller message sizes**, particularly for the zero-copy implementation at higher thread counts.

This behavior can be seen clearly in the LLC cache miss plots (Figure 4), where certain configurations show disproportionately high LLC miss counts compared to neighboring data points.

Expected behavior: Cache misses were expected to increase smoothly with message size and thread count.

Observed behavior: The cache miss pattern is irregular, with sharp spikes at specific message sizes and thread counts.

Explanation based on system behavior:

- Zero-copy introduces additional kernel metadata accesses and page pinning, which increase cache pressure.
- At higher thread counts, multiple threads compete for shared kernel data structures, leading to LLC thrashing.
- OS scheduling and concurrent kernel networking activity further amplify cache interference.

This result highlights that cache behavior is strongly influenced by kernel-level operations and scheduling effects, not just application-level data movement.

6 Conclusion

This assignment evaluated the performance impact of two-copy, one-copy, and zero-copy communication mechanisms using TCP sockets on a Linux system. The study highlights that reducing data copies does not automatically translate to better performance.

The key findings from the experiments are:

1. **Throughput** increases significantly with message size due to amortization of syscall and protocol overhead, while thread scaling provides limited benefit on the loopback interface.
2. **Latency** grows with message size but remains largely insensitive to thread count, indicating that per-message processing dominates scheduling overhead.
3. **CPU efficiency** improves sharply with increasing message size, with cycles per byte decreasing by nearly two orders of magnitude from small to large messages.
4. **Zero-copy communication** is not beneficial for small messages due to high fixed kernel overhead, but becomes advantageous at large message sizes (approximately 4 KB and above).
5. **Cache behavior** shows irregular patterns, suggesting significant influence from kernel activity, memory placement, and scheduling effects rather than purely application-level behavior.

7 AI Usage Declaration

I acknowledge that AI-based tools (specifically ChatGPT and Claude) were used as supportive tools during the completion of this assignment. The intent behind using these tools was to assist with understanding system-level concepts, resolving implementation issues, and improving clarity about the concept.

7.1 Source Code Assistance

AI tools were used during the development of the client and server programs for Parts A1, A2, and A3 to clarify socket programming semantics, threading behavior, and correct usage of system calls. The following files were developed with AI assistance in the form of guidance. debugging suggestions, and code restructuring:

- `MT25078_Part_A1_Client.c`
- `MT25078_Part_A1_Server.c`
- `MT25078_Part_A2_Client.c`
- `MT25078_Part_A2_Server.c`
- `MT25078_Part_A3_Client.c`
- `MT25078_Part_A3_Server.c`

AI assistance was primarily used to:

- Clarify correct usage of `send()`, `sendmsg()`, and `MSG_ZEROCOPY`
- Debug logical errors in multithreaded client–server interactions
- Improve error handling and code modularity

All final implementations were written, compiled, executed, and validated by me. At no point were AI tools used as a substitute for experimental execution, data collection, or result interpretation.

7.2 Automation and Build Scripts

AI tools were also used to assist in creating and debugging the experiment automation and build infrastructure, particularly due to environment-specific issues and compatibility differences.

The following files were developed with AI assistance:

- `MT25078_RunExperiments.sh`
- `Makefile`

AI assistance helped in:

- Structuring loops to iterate over message sizes and thread counts
- Parsing `perf stat` output correctly into CSV format
- Handling environment-specific errors and improving script robustness and handling infrastructure of different system(cpu) and how to run on those system

7.3 Plotting and Visualization

AI tools were used while developing the Python plotting scripts to ensure compliance with assignment requirements and correct visualization practices. The following plotting files were prepared with AI assistance:

- `MT25078_Plot_Latency_vs_Threads.py`
- `MT25078_Plot_Throughput_vs_MsgSize.py`
- `MT25078_Plot_CacheMiss_vs_MsgSize.py`
- `MT25078_Plot_CyclesPerByte.py`

AI assistance was limited to:

- Ensuring correct axes, labels, and legends
- Structuring multiple plots per script
- Verifying that all values were hardcoded as required

All numerical values used in plots were manually extracted from the experimental CSV files.

8 GitHub Repository

Repository URL: `https://github.com/SrvstvRajat/GRS_PA02`