Introduction

The efficient management of memory access is a critical aspect of modern computing systems, as it directly impacts program execution time and overall system performance. Caches, as a form of high-speed memory located close to the processor, play a key role in bridging the speed gap between the CPU and main memory

This project focuses on developing a basic cache simulator using the C++ programming language. The simulator models cache behavior based on configurable parameters, such as cache size, block size, associativity, replacement policy, and write policy. It processes memory access patterns provided in a trace file to compute key performance metrics, including hit rates and memory access latencies. By simulating different cache configurations and analyzing their impact on performance, this project aims to provide a deeper understanding of cache mechanisms and their significance in computer architecture.

The simulator was used to evaluate the performance of various cache setups across multiple trace files, generating insights that have been presented in this comprehensive report. This analysis is complemented by graphs comparing metrics like total hit rate and average memory access latency.

Important Note: In our implementation, for our Write-Allocation Policy, we implemented the 0 -> No Write-Allocation and 1 -> Write-Allocation.

Procedure

1. Cache Simulator Design and Implementation

- The project began with the creation of a cache simulator in C++. The simulator was designed to model cache memory behavior based on user-defined parameters.
- Parameters included:

    1. A configuration file which included all the required details for setting up the cache and calculating the output statistics:
       Block Size - The cache in bytes.
       Associativity - Specifies the associativity of the cache.
       Data size - Specifies the total size of the data in the cache.
       Replacement Policy - Specifies the replacement policy to be used.
       Miss penalty - Specifies the number of cycles penalized on a cache miss.
       Write allocate - Specifies the cache policy on store misses.
    2. A trace file used for calculating the output statistics. The trace file contained details of all data memory accesses made by the sample program. Each line in

the trace file represents a memory reference and included the following three fields:
Access Type - A single character indicating whether the access is a load ('l') or a store ('s').
Address - A 32-bit integer specifying the memory address that is being accessed.
Instructions since last memory access - Specifies the number of instructions of any type executed since the previous memory access.

- The simulator processed memory traces - sequences of memory addresses accessed by a program - provided in trace files.

2. Simulating Cache Configurations

- Configurations Tested:
    - **Direct-mapped Caches:** Large, medium, small, and mega configurations.
    - **Set-associative Caches:** 2-way and 4-way associativity with varying replacement policies (LRU and random).
    - **Write Policies:** Both Write-Allocate (WA) and No Write-Allocate (NWA) strategies were tested for 2-way caches.
- A total of eight cache configurations were simulated across all workloads.

3. Workload Selection

- To evaluate the simulator's performance, standard workloads were chosen from common benchmarks. The workloads used were:
    - **gcc**: A compiler workload.
    - **gzip**: A compression workload.
    - **mcf**: A memory-intensive workload.
    - **swim**: A floating-point computational workload.
    - **twolf**: A CAD application workload.
- These workloads were chosen to represent a variety of access patterns and working set sizes, ensuring a comprehensive evaluation.

4. Performance Metrics

- The simulator recorded the following key performance metrics for each configuration and workload:
    - **Total Hit Rate**: The percentage of memory accesses found in the cache.
    - **Load Hit Rate**: Hit rate specifically for load instructions.
    - **Store Hit Rate**: Hit rate specifically for store instructions.
    - **Total Run Time (in CPU cycles)**: The time taken to execute the workload.
    - **Average Memory Access Latency (in CPU cycles)**: The average time required to access memory.

5. Testing Procedure

- **Step 1:** Define the cache configuration parameters (size, associativity, replacement/write policy).
- **Step 2:** Load the trace file corresponding to the workload.
- **Step 3:** Simulate the cache behavior by processing the trace file through the simulator.
- **Step 4:** Record detailed results for the defined performance metrics.
- **Step 5:** Repeat the process for all combinations of configurations and workloads.

6. Data Analysis

- The recorded results were analyzed to understand the impact of different cache configurations on performance metrics.
- Specific trends, such as how associativity reduces conflict misses or how cache size affects capacity misses, were identified.
- Comparative analysis was conducted to highlight the trade-offs between configurations (ex. 2-way vs. 4-way associativity, direct-mapped vs. set-associative).

7. Visualization

- Results were visualized using graphs to make trends and comparisons clearer. Two key graphs were generated:

- **Total Hit Rate by Configuration and Trace**: Showed how hit rates varied across workloads and configurations.
- **Average Memory Access Latency by Configuration and Trace**: Illustrated the impact of cache configurations on memory access latency.

- These visualizations provided a concise and informative summary of the results.

Results

The following tables show the results of our cache simulation across different configurations and workloads. Each table includes performance metrics such as total hit rate, load hit rate, store hit rate, total run time (in cycles), and average memory access latency (in cycles) for the specified configuration. These results provide insights into how different cache designs influence system performance.

2way-nwa Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
2way-nwa	gcc	93.8323	98.7577	85.8962	3766584	5.3174
2way-nwa	gzip	66.7132	50.2414	99.5785	12292034	24.3007
2way-nwa	mcf	1.0530	95.6129	0.2701	51383752	70.2629
2way-nwa	swim	92.6073	99.0773	75.3069	2745182	6.1749
2way-nwa	twolf	98.8602	99.8002	96.3469	1836237	1.7978

2way-wa Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
2way-wa	gcc	98.7843	99.4679	97.6829	1978994	1.8510
2way-wa	gzip	66.8253	50.2576	99.8817	12254304	24.2223
2way-wa	mcf	75.2377	96.1487	75.0645	13619238	18.3336
2way-wa	swim	97.8736	99.7122	92.9573	1627492	2.4885

2way-wa	twolf	99.6570	99.8665	99.0968	1566947	1.2401

4way-lru Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
4way-lru	gcc	98.7636	99.4362	97.6798	1858964	1.6182
4way-lru	gzip	66.8253	50.2576	99.8817	9062604	17.5874
4way-lru	mcf	75.2378	96.1654	75.0645	10017608	13.3811
4way-lru	swim	97.8618	99.6982	92.9512	1500352	2.0691
4way-lru	twolf	99.6578	99.9677	99.0968	1533627	1.1711

4way-rand Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
4way-rand	gcc	98.6823	99.3375	97.6267	1879914	1.6588
4way-rand	gzip	66.8242	50.2561	99.8817	9062854	17.5879
4way-rand	mcf	75.2353	95.9812	75.0636	10018508	13.3823
4way-rand	swim	97.7661	99.5971	92.8700	1514852	2.1169
4way-rand	twolf	99.6349	99.8413	99.0816	1539177	1.1826

small-dm Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
small-dm	gcc	93.8516	96.4035	89.7400	3125464	4.0742
small-dm	gzip	66.6974	50.2186	99.5766	9093354	17.6513
small-dm	mcf	1.0202	91.6443	0.2698	37004264	50.4899
small-dm	swim	91.6143	97.3635	76.2411	2447452	5.1929

small-dm	twolf	98.2685	99.0544	96.1673	1869027	1.8657

medium-dm Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
medium-dm	gcc	95.3579	98.1687	90.8338	2736614	3.3201
medium-dm	gzip	66.7043	50.2277	99.5791	9091704	17.6479
medium-dm	mcf	1.0369	93.1849	0.2740	36998160	50.4815
medium-dm	swim	92.9237	98.8258	77.1415	2248952	4.5382
medium-dm	twolf	98.7235	99.5464	96.5234	1759177	1.6382

large-dm Configuration

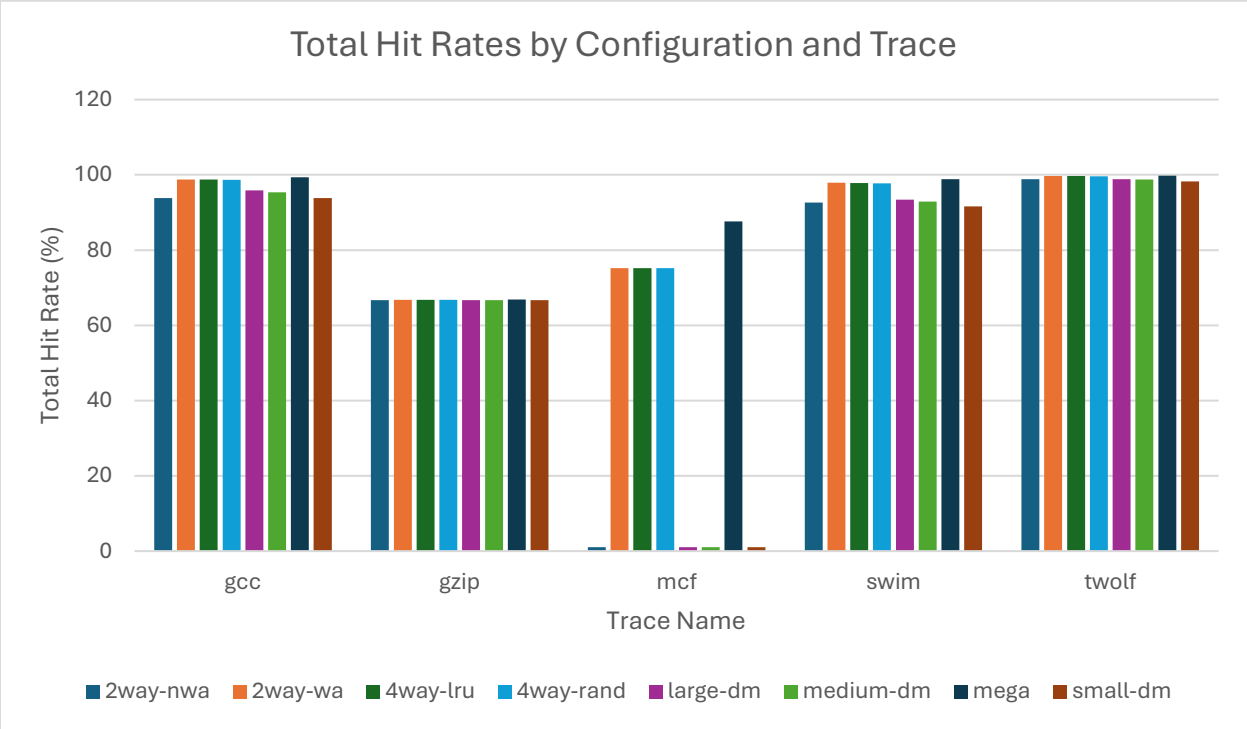
Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
large-dm	gcc	95.8346	98.7215	91.1832	2614164	3.0827
large-dm	gzip	66.7072	50.2320	99.5791	9091004	17.6464
large-dm	mcf	1.0379	93.2351	0.2745	36997808	50.4810
large-dm	swim	93.4319	99.3651	77.5668	2171902	4.2840
large-dm	twolf	98.8443	99.6855	96.5949	1730027	1.5779

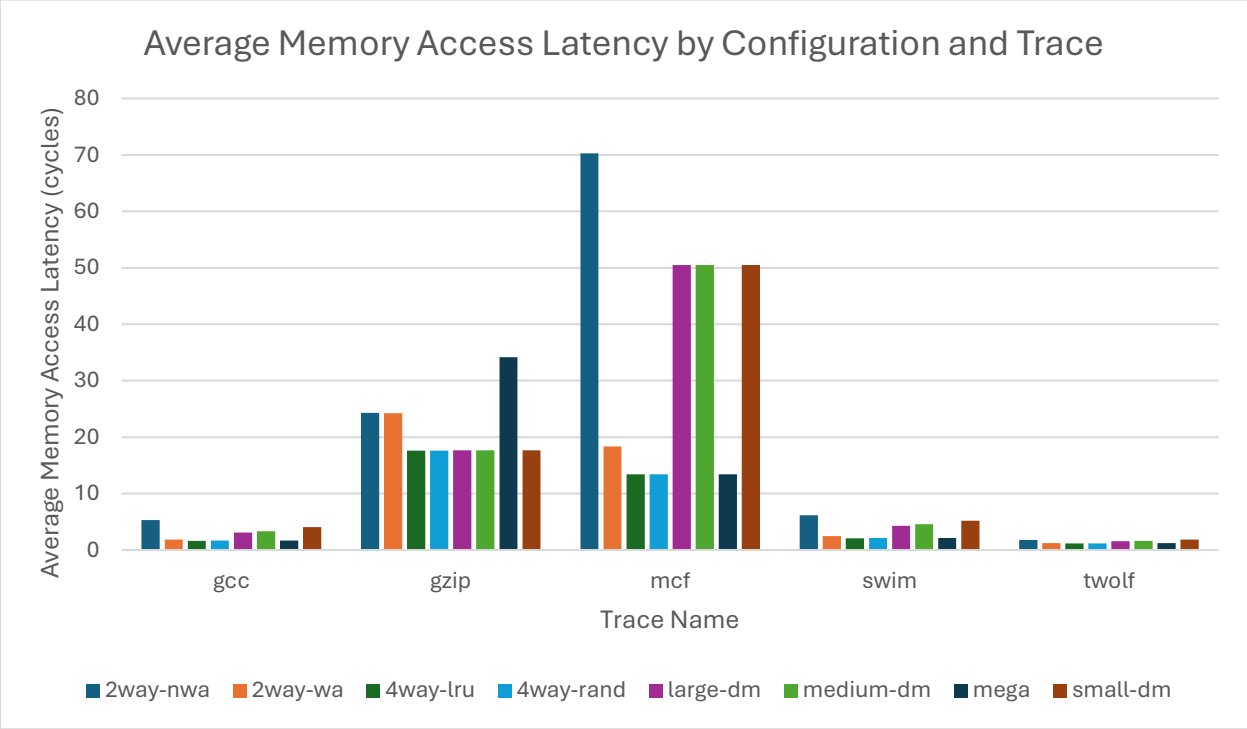
mega Configuration

Config	Trace	Total Hit Rate	Load Hit Rate	Store Hit Rate	Total Run Time (cycles)	Average Memory Access Latency
mega	gcc	99.3463	99.6873	98.7969	1877264	1.6537
mega	gzip	66.8461	50.2632	99.9328	17031854	34.1539
mega	mcf	87.6135	97.6055	87.5308	10021508	13.3865
mega	swim	98.8611	99.8219	96.2920	1521502	2.1389

mega	twolf	99.8026	99.9055	99.5275	1546327	1.1974

Bar Charts





Discussion

Total Hit Rate Bar Chart

This bar chart compares the total hit rates across different cache configurations (2way-nwa, 2way-wa, 4way-lru, 4way-rand, large-dm, medium-dm, mega, and small-dm) for five workloads (gcc, gzip, mcf, swim, and twolf). Below are the key observations and interpretations based on the updated results:

1. **High Hit Rates for gcc, swim, and twolf:**
    - For workloads like gcc, swim, and twolf, most configurations exhibit hit rates above 90%.
    - The mega, 4-way associative (lru and rand), and 2-way associative (wa) configurations achieve the highest hit rates, often exceeding 98%.
    - These results suggest strong spatial and temporal locality in these workloads, which aligns well with associative cache designs and larger cache sizes.
2. **Low Hit Rate for mcf with Direct-Mapped Caches:**
    - The workload mcf shows very low hit rates in direct-mapped caches (ex. small-dm: 1.02%, medium-dm: 1.04%). This indicates significant conflict misses due to mcf's access patterns that frequently reuse addresses mapping to the same cache index.
    - Associative caches (e.g., 4way-lru: 75.23%) significantly mitigate conflict misses, improving hit rates.
3. **Moderate Hit Rate for gzip:**
    - The gzip workload has moderate hit rates (~66%) across all configurations. This suggests gzip has less pronounced locality but benefits slightly from associativity and cache size.
    - Despite the consistent hit rates, configurations like 4way-lru and mega provide marginal improvements, indicating gzip’s access patterns are less sensitive to associativity or size.
4. **Impact of Cache Size and Associativity:**
    - Larger caches such as mega provide higher hit rates across all workloads, especially mcf (87.61%) and swim (98.86%), due to reduced capacity and conflict misses.

- 4-way associative caches outperform 2-way caches for workloads like gcc and twolf, although the performance gain is marginal, indicating diminishing returns beyond 2-way associativity.

5. **Consistency Across Replacement and Write Policies:**
    - Hit rates for configurations like 2way-nwa and 2way-wa, as well as 4way-lru and 4way-rand, are nearly identical. This suggests that replacement and write allocation policies have minimal impact on total hit rates for these workloads.

General Trends and Insights:

- **High locality workloads** (ex. gcc, swim) perform well across most configurations, while irregular access patterns (e.g., mcf) benefit more from higher associativity and larger cache sizes.
- **Direct-mapped caches** suffer from significant conflict misses, highlighting the importance of associativity in systems running diverse applications.
- These findings emphasize tailoring cache designs to workload characteristics for optimal performance.

Average Memory Access Latency Bar Chart

This bar chart depicts the average memory access latency (in cycles) across various cache configurations and workloads. Below are the key observations and interpretations based on the revised results:

1. **High Latency for mcf in Direct-Mapped Caches:**
    - The mcf workload exhibits the highest latencies with direct-mapped caches (ex. small-dm: 50.4899 cycles, medium-dm: 50.4815 cycles). This is due to frequent main memory accesses caused by poor hit rates.
    - Associative caches, such as 4way-lru (13.38 cycles), significantly reduce latency by mitigating conflict misses.
2. **Low Latency for gcc, swim, and twolf:**
    - Workloads like gcc, swim, and twolf maintain low latencies across all configurations due to high hit rates. For example, gcc with 2way-wa achieves 1.85 cycles, and twolf with mega achieves 1.19 cycles.
3. **Moderate Latency for gzip:**

- The gzip workload shows moderate latencies (~17-34 cycles) across configurations, with direct-mapped caches exhibiting slightly higher latencies. This reflects a balance between cache hits and misses.

4. **Impact of Cache Size:**
    - Larger caches, such as mega, consistently achieve lower latencies. For instance, mega reduces mcf's latency to 13.38 cycles compared to small-dm at 50.48 cycles.
    - This trend demonstrates the effectiveness of larger caches in handling workloads with larger working sets.
5. **Effect of Associativity:**
    - 4-way associative caches (lru and rand) generally achieve lower latencies than 2-way caches for workloads like mcf and gzip. For instance, gzip with 4way-lru achieves 17.58 cycles compared to 2way-nwa at 24.22 cycles.
    - For workloads with high locality, such as gcc and swim, the latency differences between 2-way and 4-way configurations are negligible.
6. **Minimal Effect of Replacement and Write Policies:**
    - Replacement and write allocation policies have limited impact on latency for most workloads. For example, 4way-lru and 4way-rand achieve nearly identical latencies for all workloads, reinforcing the dominance of cache size and associativity in performance outcomes.

General Trends and Insights:

- Workloads with high locality (ex. gcc, swim, twolf) achieve low latencies across all configurations due to fewer cache misses.
- Irregular workloads (ex. mcf) benefit significantly from higher associativity and larger cache sizes, reducing memory access latency.
- The relationship between hit rate and latency is evident, with configurations achieving higher hit rates consistently showing lower latencies, emphasizing the role of cache design in optimizing system performance.