## Computer-Architecture (Memory Design)

## Assignment-report

**Group: Thearchitects -Bharath( IMT2023504) ,Rakshith(IMT2023544),Lesin(IMT2023565)**

### 

### **Design and Analysis of a 4-Way Set Associative Cache**

**The goal is to:**

* Design a 4-way set associative cache of size 1024KB with a block size of 4 bytes.
* Report hit/miss rates using the given memory traces.
* Vary cache size from 128KB to 4096KB and analyse miss rates.
* Vary block size and associativity for additional experiments.

**About the code**:

Python code simulates a set-associative cache and analyses its performance under different configurations. It defines the SA-Cache class for managing cache operations such as checking hits and handling cache replacement using LRU (Least Recently Used) policy. The code evaluates trace files to measure miss rates while varying cache size, block size, and associativity, and then plots the results using matplotlib.

**Experiments and Results**

1. Cache Size: 1024 KB

Block Size: 4 bytes

Address Size: 32-bit address space

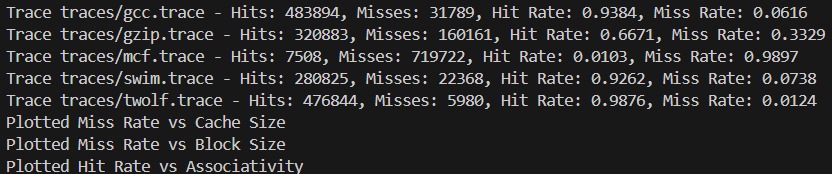
Associativity: 4-way set associative

The number of cache lines is determined by the total cache size divided by the block size:

Number of cache lines = Cache Size / Block Size

=(2^20 bytes / 4\*4 bytes)=2^16.

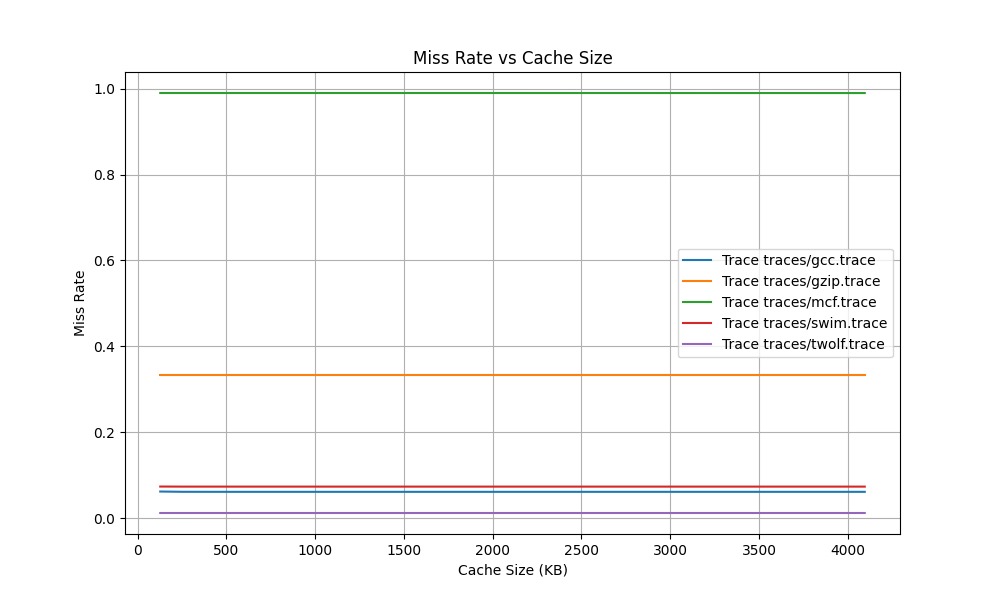
Terminal output:



**b) Cache Size Variation**

The cache size was varied from 128KB to 4096KB to observe the change in miss rates. The following observations were made:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Trace traces/gcc.trace | Trace traces/gzip.trace | Trace traces/mcf.trace | Trace traces/swim.trace | Trace traces/twolf.trace |
| 128 | 6.198381564 | 33.29445955 | 98.96758935 | 7.380117615 | 1.238753666 |
| 256 | 6.168906092 | 33.29445955 | 98.96758935 | 7.377479032 | 1.238546551 |
| 512 | 6.164639905 | 33.29445955 | 98.96758935 | 7.377479032 | 1.238546551 |
| 1024 | 6.164445987 | 33.29445955 | 98.96758935 | 7.377479032 | 1.238546551 |
| 2048 | 6.164445987 | 33.29445955 | 98.96758935 | 7.377479032 | 1.238546551 |
| 4096 | 6.164445987 | 33.29445955 | 98.95452608 | 7.377479032 | 1.238546551 |

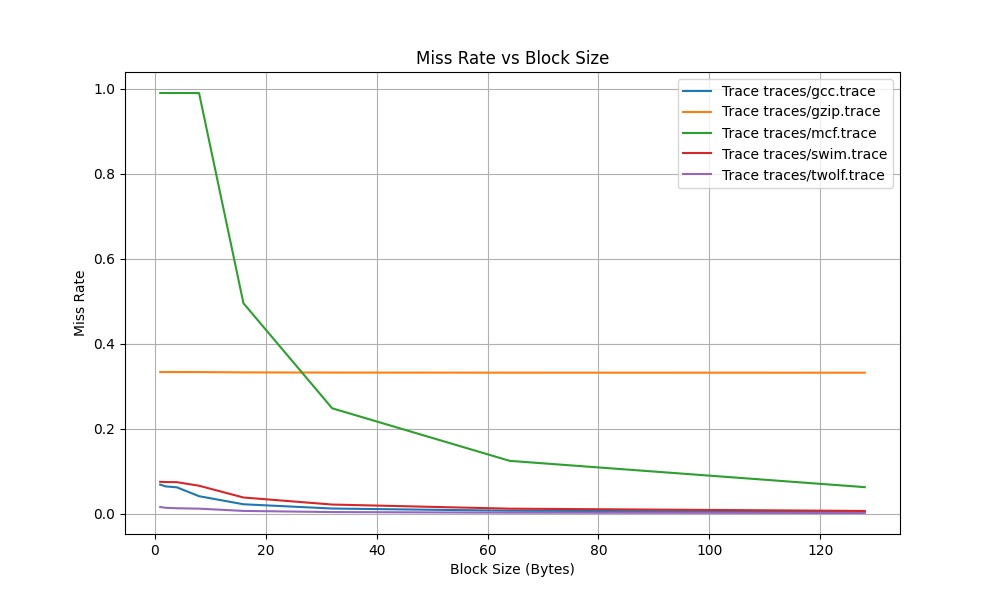
**Graph: Miss Rate vs Cache Size** 

* **Observations**: As the cache size increases, the miss rate decreases due to more data fitting into the cache. However, beyond a certain point, increasing the cache size has diminishing returns.
* Not all traces behave the same way when varying the cache size. Some workloads (like **gcc, swim, twolf**) benefit up to a point, while others (like **gzip** and **mcf**) do not see significant improvements, even with a larger cache. This highlights the importance of understanding the specific memory access patterns of workloads when designing cache systems

**c) Block Size Variation**

The block size was varied from 1 byte to 128 bytes while keeping the cache size constant at 1024KB. The change in miss rates was noted and graphed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Trace traces/gcc.trace | Trace traces/gzip.trace | Trace traces/mcf.trace | Trace traces/swim.trace | Trace traces/twolf.trace |
| 1 | 6.801077406 | 33.2961226 | 98.97542731 | 7.455647063 | 1.523122297 |
| 2 | 6.375234398 | 33.29591472 | 98.97130206 | 7.406503448 | 1.339204348 |
| 4 | 6.164445987 | 33.29445955 | 98.96758935 | 7.377479032 | 1.238546551 |
| 8 | 4.07343271 | 33.2927965 | 98.9616765 | 6.535770945 | 1.140167017 |
| 16 | 2.17497959 | 33.21442529 | 49.49699545 | 3.767567193 | 0.612024257 |
| 32 | 1.171068273 | 33.17471998 | 24.76217978 | 2.109547384 | 0.340082515 |
| 64 | 0.654084001 | 33.15393186 | 12.39195303 | 1.138878536 | 0.197587527 |
| 128 | 0.379108871 | 33.14353781 | 6.204502014 | 0.602256648 | 0.119091015 |

**Graph: Miss Rate vs Block Size**

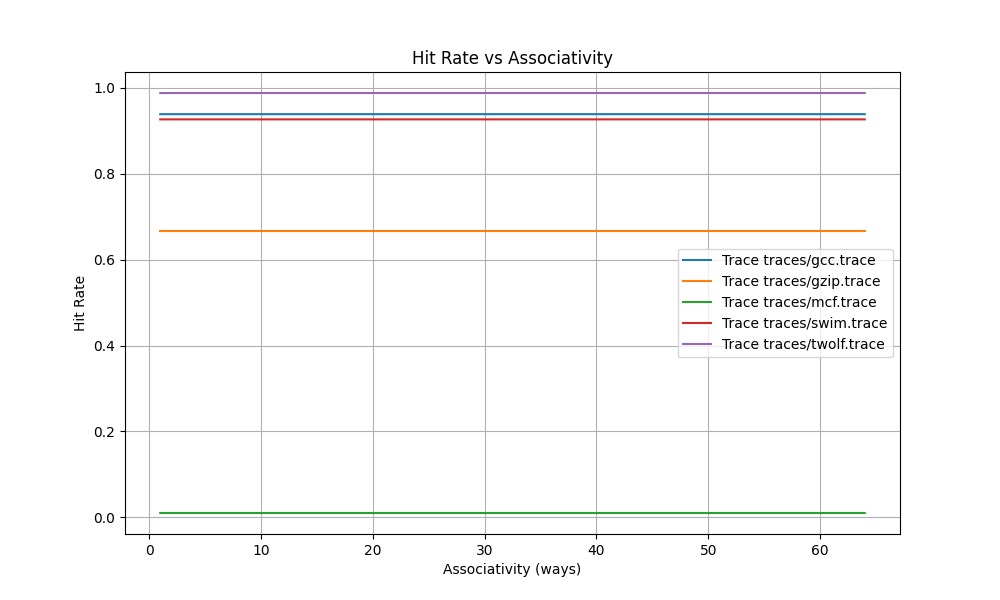
* **Observations**: Larger block sizes reduced the number of cache lines, which in turn impacted the miss rate. Depending on the trace, the optimal block size differed, suggesting varying access patterns in the programs.
* Different traces behave differently when varying the block size. Applications with good spatial locality (like gcc, swim, twolf) see significant improvements with larger blocks, whereas gzip shows little sensitivity to block size. mcf, with poor spatial locality for small blocks, benefits the most from increasing block size. Therefore, the choice of block size should be carefully considered based on the specific workload and its access patterns.

**d) Rate vs Associativity**

* The associativity was varied from 1-way (direct mapped) to 64-way, with a constant cache size of 1024KB

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameter | Trace traces/gcc.trace | Trace traces/gzip.trace | Trace traces/mcf.trace | Trace traces/swim.trace | Trace traces/twolf.trace |
| 1 | 93.83051216 | 66.70554045 | 1.03199813 | 92.62054203 | 98.74633407 |
| 2 | 93.83477834 | 66.70554045 | 1.032273146 | 92.62252097 | 98.7608321 |
| 4 | 93.83555401 | 66.70554045 | 1.032410654 | 92.62252097 | 98.76145345 |
| 8 | 93.83555401 | 66.70554045 | 1.032410654 | 92.62252097 | 98.76145345 |
| 16 | 93.83574793 | 66.70554045 | 1.032410654 | 92.62252097 | 98.76145345 |
| 32 | 93.83594185 | 66.70554045 | 1.032410654 | 92.62252097 | 98.76145345 |
| 64 | 93.83594185 | 66.70554045 | 1.032410654 | 92.62252097 | 98.76145345 |

* **Graph: Miss Rate vs Associativity**



* **Observations**: Higher associativity typically improves the hit rate, as there are more slots available for each set, reducing conflicts. However, beyond a certain level of associativity, the improvement was minimal.
* Some traces behave differently because of their memory access patterns, locality, and working set sizes, leading to varying benefits from increasing cache associativity.

**Inferences from all traces :**

**1.gcc.trace**

* *In part b)* cache capacity increases, the miss rate gradually decreases.
* *In part c)* On varying the block size from 1byte to 128 bytes, we can see gradual decrease in the miss rate of the instructions. This shows that there is more temporal caching in the trace.
* *In part d)* On increasing the associativity from 1 to 64, the hit rate almost remains the same which indicates that there is not much spatial locality in the cache. Thus, adjacent addresses are not present in the trace.

**2. gzip.trace**

* From the table, we observe that miss rate remains same in all the 3 cases (*part b,c and d*). This shows that similar instructions are more frequent in the cache.

**3.mcf.trace**

* The miss rate for this trace is notably high, which suggests a lower cache hit ratio. This indicates a greater diversity in the instruction set within the trace.
* *In part c)* As the block size increases from 1 kB to 128 kB, we observe a significant reduction in the cache miss rate. This occurs because larger block sizes allow for more instructions from the same memory addresses to be fetched. As the byte offset increases with the block size, it enhances the likelihood of a cache hit by allowing more relevant data to be present in the cache.

**4. swim.trace**

* Adjusting the associativity(*in part d*) and cache size (*in part a*) leads to a slight reduction in the miss rate. However, when the block size (*in part c*) is increased significantly from 1 to 128, the miss rate drops to nearly zero

**5.twolf.trace**

* As in the previous traces, the miss rate remains consistent despite changes in cache size (*in part a*) and associativity(*in part d*). However, the hit rate significantly decreases, approaching zero, as the block size(*in part c*) increases from 1 KB to 128 KB.

**5. Conclusion**

The experiments highlight the trade-offs involved in cache design. Increasing cache size and associativity generally improves hit rates, but with diminishing returns. Block size also plays a crucial role, with optimal configurations depending on the specific memory access patterns of the program.