

# CSE 3031 Computer Architecture

## Lab Report 2

Cole Johnson

Spring 2024


Deadline: Midnight of Thursday, March 8

### Introduction

This lab purpose is demonstrate how caching in modern Computer Architecture works and how cache design decisions (cache size, block size, associativity, etc.) affect the performance and behavior of the cache. To further demonstrate the concepts of caching, we implement a trace-driven cache simulator—given both a configuration file that determines the design of the cache and a trace that determine the memory references used by the programs. After implementing the cache simulator, we can figure out several important statistics—hit rate, miss rate, total run time, and average memory access latency—to gauge the performance of several different types of cache designs.

### Methods

The cache simulator created for this purposes of this lab was written in the C language. The program takes in two files: a configuration file that determines the design of the cache and the instruction trace file that gives the memory references to simulator the running of the cache. The qualities of the cache that can be configured are: LINE/BLOCK SIZE, ASSOCIATIVITY, DATA/CACHE SIZE, REPLACEMENT POLICY, MISS PENALTY, and WRITE ALLOCATE POLICY. Each instruction trace has three different pieces of data: ACCESS TYPE (LOAD OR STORE), ADDRESS, and INSTRUCTIONS SINCE LAST MEMORY REFERENCE. Once the program is given a valid cache configuration file and instruction trace file, we can generate performance statistics of the cache using the following command:

A terminal window with a dark background. The prompt is '~ /CSE331/cache-sim main'. The command entered is './cache-sim sample.conf traces/gcc.trace'. The time '21:16:38' is shown in the top right corner.

```
~/CSE331/cache-sim main ..... 21:16:38  
./cache-sim sample.conf traces/gcc.trace
```

This will load the sample.conf cache configuration and run the gcc.trace, which will output performance statistics inside a new file named gcc.out in the current directory.

### Results and Discussion

Here's the results for each cache configuration and trace file supplied: All the data provided can be found inside the out directory.

<b>2way-nwa</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	93.20	60.56	32.64	3958546	5.76
gzip	66.68	33.47	33.21	12144469	24.33
mcf	1.32	0.78	0.53	50528799	70.07
swim	92.58	71.99	20.59	2729186	6.20
twolf	98.82	72.61	26.21	1843983	1.82

<b>2way-wa</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	98.22	61.11	37.11	2173309	2.25
gzip	66.79	33.48	33.31	12107761	24.25
mcf	74.96	0.79	74.17	13578540	18.53
swim	97.14	72.49	24.65	1775261	3.00
twolf	99.58	72.64	26.94	1590408	1.29

<b>4way-fifo</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	97.60	60.55	37.05	2146883	2.20
gzip	66.79	33.47	33.32	8911938	17.61
mcf	75.02	0.77	72.25	9913677	13.48
swim	96.68	71.93	24.76	1668800	2.66
twolf	98.98	72.30	26.68	1691618	1.51

<b>4way-rand</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	94.46	59.45	35.01	2940291	3.77
gzip	41.95	33.47	8.48	14766507	30.03
mcf	32.21	0.75	31.46	25170072	34.90
swim	93.07	70.95	22.13	2205056	4.46
twolf	98.35	71.91	26.44	1841264	1.82

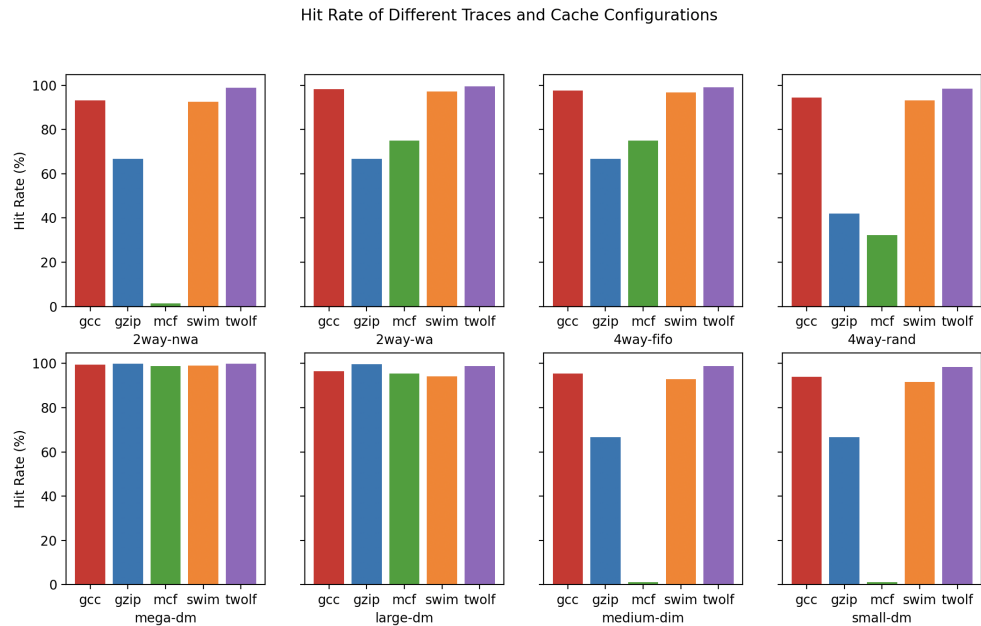
<b>mega-dm</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	99.42	61.53	37.89	1836472	1.58
gzip	99.76	66.39	33.37	1198789	1.24
mcf	98.87	0.80	98.07	1824717	2.13
swim	98.94	72.70	26.24	1494488	2.06
twolf	99.81	72.72	27.09	1542801	1.19

<b>large-dm</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	96.42	61.17	35.25	2444558	2.79
gzip	99.67	66.42	33.25	1161069	1.16
mcf	95.49	0.78	94.71	2619341	3.25
swim	94.19	72.46	21.73	2039436	3.91
twolf	98.87	72.57	26.30	1718078	1.56

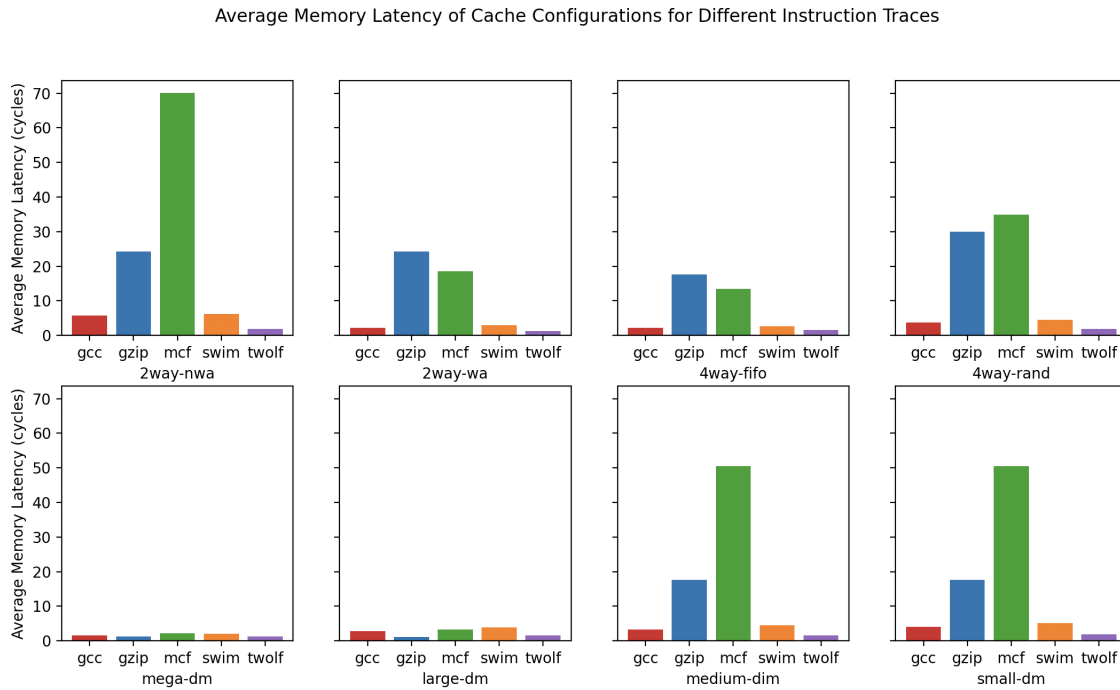
<b>medium-dm</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	95.38	60.59	34.79	2708619	3.31
gzip	66.71	33.46	33.25	8931244	17.65
mcf	1.04	0.77	0.27	36276412	50.48
swim	92.93	71.93	22.00	2226175	4.53
twolf	98.75	72.47	26.28	1747821	1.63

<b>small-dm</b>	Total Hit Rate (%):	Load Hit Rate (%):	Store Hit Rate (%):	Total Run Time (cycles):	Average Memory Access Latency (cycles):
gcc	93.86	59.49	34.37	3091407	4.07
gzip	66.70	33.45	33.25	8932861	17.65
mcf	1.02	0.76	0.27	36283174	50.49
swim	91.62	70.86	20.75	2421440	5.19
twolf	98.28	72.10	26.18	1858953	1.86

Here's a bar chart showing the total hit rate for each config running the different program traces:



Here's the average memory latency of all the trace files for each cache configuration:



One thing that becomes immediately noticeable is that the mcf.trace by far has the lowest hit rate and subsequently has the highest average memory latency, for most of the cache configurations. Only mega-dm and large-dm caches have a decent hit rate and a low average memory latency. Looking at the mcf SPEC CPU2006 description, mcf low hit rates could be potentially be from high amounts of conflict misses. Overall, the results obtained from the cache sim make sense: low hit rate caches have higher average memory latency and higher runtime, which are the results that we would expect. Overall, the data collected seem plausible and all the statistics measured are what we would expect, although I am a little suspicious about how high the average memory latency is for some of the traces—especially the results running the mcf.trace.

## Conclusion

Overall, the lab was preformed successfully and the data collected represents the behavior we would expect from a real cache. For example, greater size and associativity increase total hit rate because both decrease the likelihood of conflict misses—and this result is reflected in the data collected by the cache simulator. The lab demonstrates the basic concepts of caching and the design tradeoffs computer architects make when desinging a performant cache.