Cache Simulation Analysis

Katherine Teng

Introduction

This program simulates three different cache designs - fully associative, set associative, and direct mapped - and the two replacement policies, first in first out (FIFO) and least recently used (LRU). The goal of this simulation is to compare and analyze the hit rates of each cache design and replacement strategy.

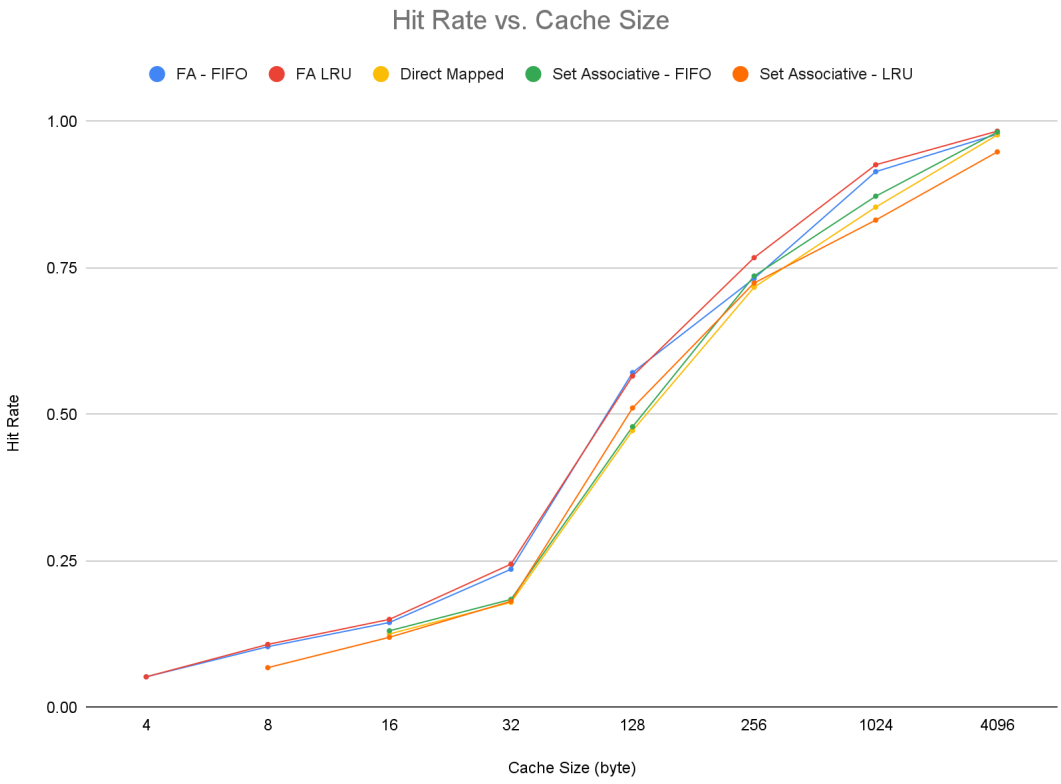
The program is written in C++, and uses a vector to simulate the cache, where each index is a line. Once the program opens, you are prompted to enter the trace file you will use for testing, and then the number of sets, blocks, and bytes for the cache. Once the simulation for this cache is finished and outputs the hit rate, you can enter the parameters for the next cache you would like to simulate. For every program run, you can simulate a total of 16 caches.

Description of Tests

For each test, I used the trace file gcc.trace. The parameters passed in are the number of sets, number of blocks per set, and number of bytes per block. I wanted to get a large range of cache sizes so I passed in parameters for the smallest cache size possible for each cache design up to a cache size of 4096 bytes. The number of bytes per block I used were 4, 8, 16, and 64. For fully associative, the blocks per set I used were 1, 2, 4, 8, 16, and 64. For set associative, I simulated 2-way, 4-way, and 8-way set associative caches. And the number of sets I used for set associative were 2, 4, 8, and 16. For direct mapped, I used the same numbers as set associative for the number of sets, as well as 64. For bytes per block I used 4, 8, 16, and 64. I used different combinations of these numbers for each cache design to get the cache size I wanted.

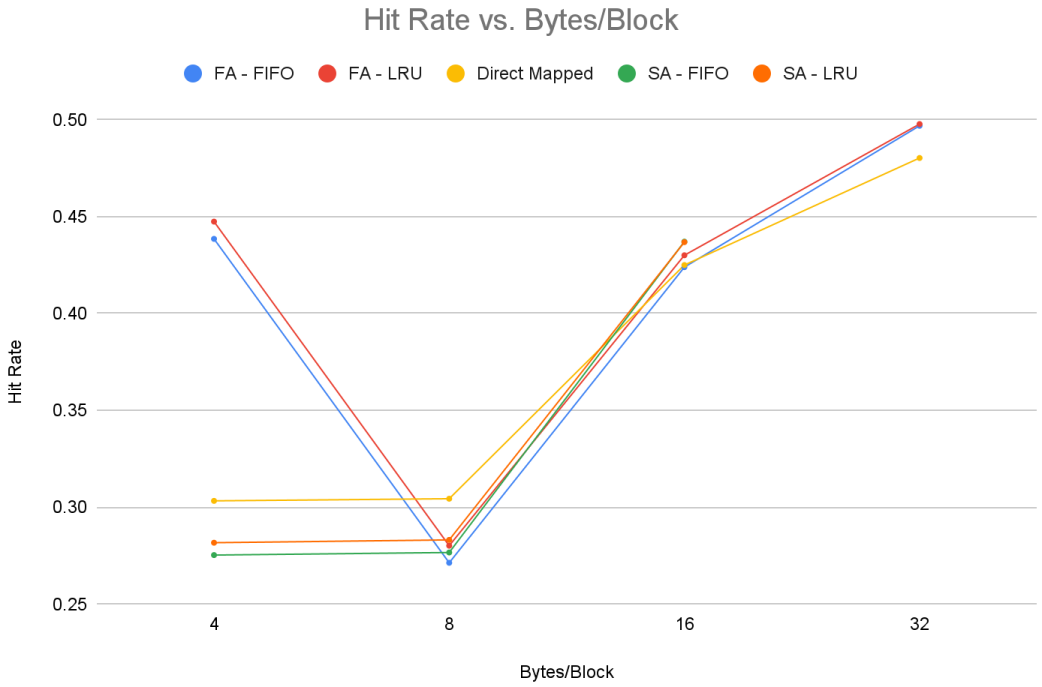
Results

The graph below shows the hit rates of each cache size for each cache design and replacement policy.



The graph shows that fully associative has a better hit rate than the others.

The graph below shows the hit rates for each cache design and replacement strategy for a fixed cache size of 64 bytes. The x-axis is the number of bytes per block. The table below shows the numbers used for the parameters for each cache design for a cache size of 64 bytes.



The graph shows that fully associative has the highest hit rate for the smallest and largest number of bytes per block. It also shows that for direct mapped and set associative, the hit rate increases as the number of bytes increases.

gcc.trace					
Fully Associative					
sets	**blocks/set**	**bytes/block**	**replacement**	**hit rate**	**cache size**
1	16	4	fifo	0.438512	64
1	8	8	fifo	0.271361	64
1	4	16	fifo	0.423906	64
1	2	32	fifo	0.496867	64
1	16	4	lru	0.44742	64
1	8	8	lru	0.280182	64
1	4	16	lru	0.430041	64
1	2	32	lru	0.497753	64
Set Associative					
sets	**blocks/set**	**bytes/block**	**replacement**	**hit rate**	**cache size**
4	4	4	fifo	0.275344	64
2	4	8	fifo	0.276678	64
2	2	16	fifo	0.437067	64
4	4	4	lru	0.281725	64
2	4	8	lru	0.28318	64
2	2	16	lru	0.436832	64
Direct Mapped					
sets	**blocks/set**	**bytes/block**	**replacement**	**hit rate**	**cache size**
16	1	4		0.303322	64
8	1	8		0.30442	64
4	1	16		0.4249988	64
2	1	32		0.480241	64

Conclusion

Based on the results, we can say that generally, fully associative has the highest hit rate, and direct mapped has the lowest. For replacement policies, the hit rate for LRU is always higher than FIFO for each cache design. However, if the cache size is very small, the hit rate for LRU would equal the hit rate for FIFO in some cases. Lastly, as cache size increases, hit rate also increases for every cache design and replacement policy. For a fixed cache size, the hit rate generally increases as the bytes per block increases. However, for fully associative, there is a dip in the hit rate when the bytes per block is equal to the blocks per set. When the cache size is fixed, it is not clear which cache design has the highest hit rate, it varies as the number of bytes increases.