Gary Quilligan

Cache Analysis

1. Introduction

This simulation creates a cache with a selectable byte size, number of lines, number of lines in a set, and replacement strategy. If the number of lines in a set is equal to the number of lines, then this a fully associative cache where a line could be placed anywhere in the cache. If the number of lines in a set is 1, then this is a direct mapped cache and each block in main memory has only one valid line number in the cache to be placed. The last option is set-associative which would have n sets of m lines, and blocks from main memory can be placed into their respective sets. The two replacement strategies in this simulation are LRU and FIFO. LRU keeps track of whether a line in the cache has been recently accessed, so if there is a tag match, replacement, or first initialization you increase the global counter and set that equal to the count for that line. FIFO only keeps track of what order each line has been placed into the cache, so accessing a line and making a hit does not increase the count for that line. The goal of this simulation is to see the hit rates of each respective cache design.

2. Description of Tests

Test 1: How does performance of a cache change with associativity (keep everything constant besides associativity)

- Use LRU
- Use gcc.txt
- Use 1024 byte size and 64 byte line size, and 512 byte size 32 byte line size
- Compare NumLinesInSet: DM = 1, FA = 16, 2-way = 2, 4-way = 4

Test 2: How does the performance of a cache change with cache size

(keep everything constant besides byte size)

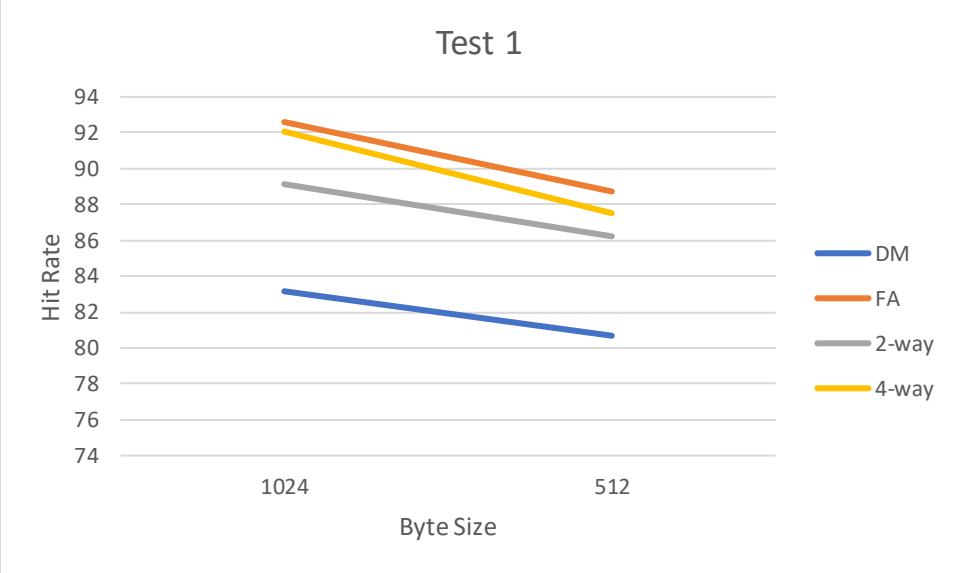
- Use LRU
- Use gcc.txt
- Use NumLinesInSet = 1 or Direct Mapping
- Compare Byte Size = 1024, 512, 256, 64 all using byte line size = (Byte Size/16)

Test 3: How does the performance of a cache change with replacement policy

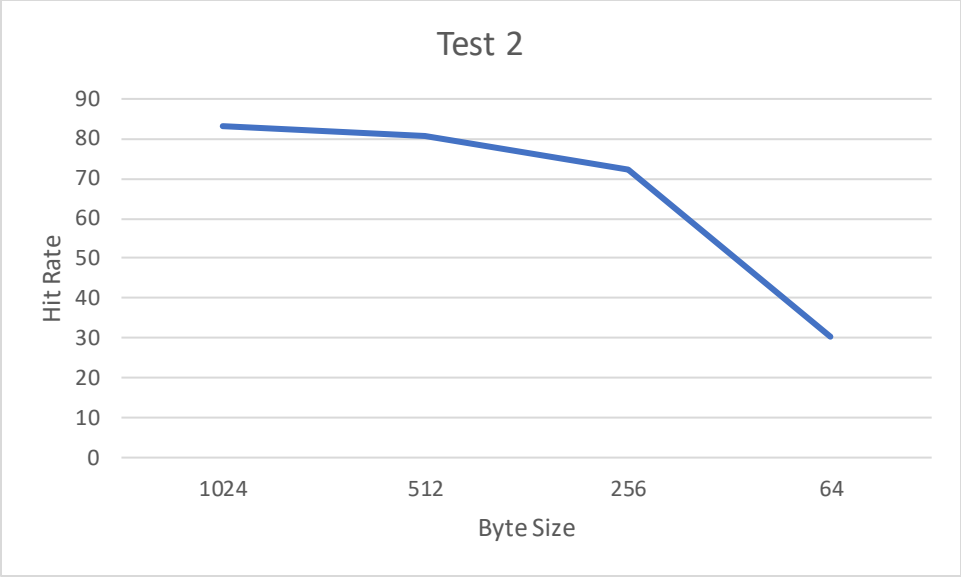
(keep everything constant besides replacement policy)

- Use gcc.txt
- Use 1024 byte size and 64 byte line size, and 512 byte size and 32 byte line size
- NumLinesInSet = 16 or fully associative
- Compare LRU and FIFO

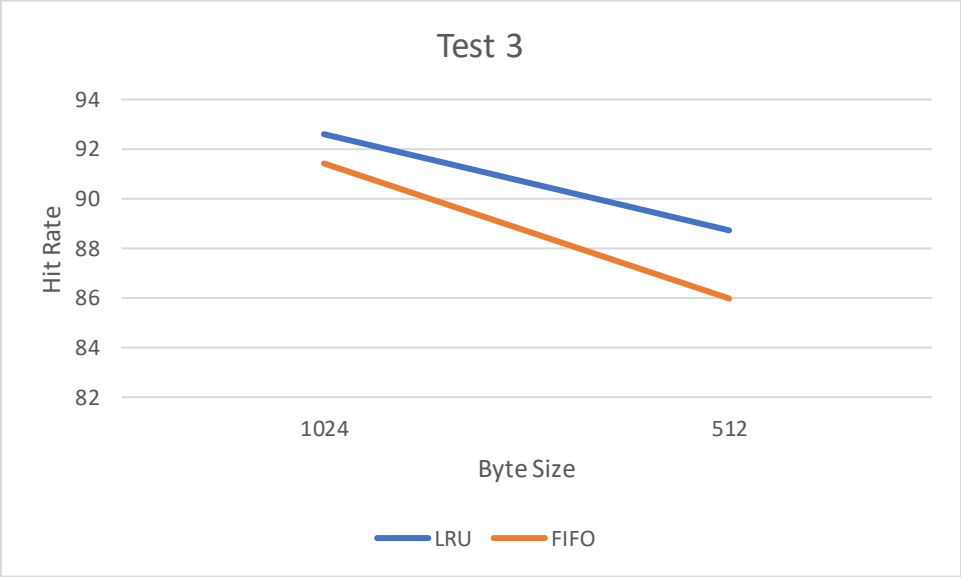
3. Results



	1024 bytes	512 bytes
DM	83.163	80.684
FA	92.6	88.725
2-way	89.134	86.225
4-way	92.068	87.514



1024 bytes	83.163 percent
512 bytes	80.684 percent
256 bytes	72.43 percent
64 bytes	30.332 percent



	1024 bytes	512 bytes
LRU	92.6 percent	88.725 percent
FIFO	91.422 percent	85.978 percent

4. Conclusions

Test 1: This test tells us that hit rate performance is ranked best to worst: fully associative, 4-way set associative, 2-way set associative, and direct mapped. This makes sense because if it is possible to check whole cache space for a desired tag there is a higher change of making a hit (fully associative). As the amount of locations that a tag can be checked decreases its chances of being found decreases.

Test 2: As the size of the cache decreases the hit rate performance decreases. This makes sense because if there are less lines in a cache it will get filled up more quickly, and less data can be stored, so more misses will occur. If there was more space more things could be stored in the cache, and when you check the cache if some data is there, there is a higher chance it didn't get replaced.

Test 3: LRU policy has a better hit rate performance than FIFO. Lets say the first thing that was stored in the cache at some point is accessed again and again, then we store something that moves this highly-accessed data out of the cache because we are using FIFO. Now when this important data is attempted to be found again it is gone and there is a miss. If LRU had been used, then data that is accessed commonly will be kept in the cache improving hit rate performance.