Jared Barber

Dr. Resch

CDA 3101

16 April 2022

Cache Simulation Report

**Introduction**

The simulation is a C++ console application that depends on user input for the cache size, block size, and number of lines in a set. The trace file name is hard coded into the main function and has to be changed manually. The program will run simulations for a direct mapped, fully associative, and n-way set associative cache. The user will also be prompted to choose between an LRU and a FIFO replacement scheme.

Each of the three cache designs are handled by separate functions which create a cache data structure, parse the input file, simulate a cache, and return the hit rate. The result is then printed to the console and the program exits.

Finding an element in a requires O (1) time in a direct mapped cache, O (log n) time in a set-associative cache, and O (n) in a fully associative cache, where n is the size of the cache.

**Description of Tests**

The first test examines how the performance of a cache changes with its associativity – whether the cache is direct mapped, n-way associative, or fully associative. For this test, I compared direct mapped, 2-way associative, 4-way associative, and fully associative caches. I chose these parameters because are sufficient to show a pattern in the data. A direct mapped cache is the same as a 1-way set associative, and with the parameters given, an 8-way set associative models a fully associative cache. The cache size and block size were held constant at 512 and 64 bytes. Where applicable, a least-recently used replacement scheme was used. The only change was the number of blocks per set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | Direct mapped | 8 | 1 | 64 | 0.651529 |
| Swim | 2-way associative | 4 | 2 | 64 | 0.679267 |
| Swim | 4-way associative | 2 | 4 | 64 | 0.671289 |
| Swim | Fully associative | 1 | 8 | 64 | 0.850028 |
| Gcc | Direct mapped | 8 | 1 | 64 | 0.785403 |
| Gcc | 2-way associative | 4 | 2 | 64 | 0.828616 |
| Gcc | 4-way associative | 2 | 4 | 64 | 0.845578 |
| Gcc | Fully associative | 1 | 8 | 64 | 0.850598 |

The second test examines how the performance of a cache changes with cache size. For this test, I chose to test direct mapped, 2-way set associative, and fully associative caches of 512, 1024, and 2048 bytes each. Since this only needs to establish a pattern, there is no need to test additional associativities.

The cache size was the only parameter intentionally changed; for 2-way associative and direct mapped the number of sets in the cache changed as a result. For fully associative, the number of blocks in the set also changed as size increased. Where applicable, an LRU replacement scheme was used. For all tests, block size was constant at 64 bytes.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | Direct mapped | 8 | 1 | 64 | 0.65129 |
| Swim | Direct mapped | 16 | 1 | 64 | 0.761868 |
| Swim | Direct mapped | 32 | 1 | 64 | 0.818901 |
| Gcc | Direct mapped | 8 | 1 | 64 | 0.784503 |
| Gcc | Direct mapped | 16 | 1 | 64 | 0.831627 |
| Gcc | Direct mapped | 32 | 1 | 64 | 0.889568 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | 2-way associative | 4 | 2 | 64 | 0.679267 |
| Swim | 2-way associative | 8 | 2 | 64 | 0.843865 |
| Swim | 2-way associative | 16 | 2 | 64 | 0.926588 |
| Gcc | 2-way associative | 4 | 2 | 64 | 0.828616 |
| Gcc | 2-way associative | 8 | 2 | 64 | 0.891342 |
| Gcc | 2-way associative | 16 | 2 | 64 | 0.921929 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | Fully associative | 1 | 8 | 64 | 0.650028 |
| Swim | Fully associative | 1 | 16 | 64 | 0.91078 |
| Swim | Fully associative | 1 | 32 | 64 | 0.967183 |
| Gcc | Fully associative | 1 | 8 | 64 | 0.850998 |
| Gcc | Fully associative | 1 | 16 | 64 | 0.92603 |
| Gcc | Fully associative | 1 | 32 | 64 | 0.962729 |

The third test examines how the performance of a cache changes with replacement policy. Direct mapping can only have one replacement policy, so it was not considered. As there were only two options, least recently used and first in first out, fewer values needed to be considered.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Replacement | Hit Rate |
| Swim | 2-way associative | 4 | 2 | 64 | LRU | 0.679267 |
| Swim | 2-way associative | 4 | 2 | 64 | FIFO | 0.665939 |
| Gcc | 2-way associative | 4 | 2 | 64 | LRU | 0.828616 |
| Gcc | 2-way associative | 4 | 2 | 64 | FIFO | 0.819662 |
| Swim | Fully associative | 1 | 8 | 64 | LRU | 0.650028 |
| Swim | Fully associative | 1 | 8 | 64 | FIFO | 0.619345 |
| Gcc | Fully associative | 1 | 8 | 64 | LRU | 0.850598 |
| Gcc | Fully associative | 1 | 8 | 64 | FIFO | 0.820252 |

The fourth test examines how the performance of a cache changes with block size, or the number of bytes in a block. I compared the hit rates for the three types of caches using a 1024-byte cache with increasing block sizes of 64,128, and 256 bytes. I chose these values because after three tests, I felt confident that a pattern had been established.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | Direct mapped | 16 | 1 | 64 | 0.761868 |
| Swim | Direct mapped | 8 | 1 | 128 | 0.714766 |
| Swim | Direct mapped | 4 | 1 | 256 | 0.622722 |
| Gcc | Direct mapped | 16 | 1 | 64 | 0.831627 |
| Gcc | Direct mapped | 8 | 1 | 128 | 0.77877 |
| Gcc | Direct mapped | 4 | 1 | 256 | 0.738902 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | 2-way associative | 8 | 2 | 64 | 0.843865 |
| Swim | 2-way associative | 4 | 2 | 128 | 0.781733 |
| Swim | 2-way associative | 2 | 2 | 256 | 0.684907 |
| Gcc | 2-way associative | 8 | 2 | 64 | 0.891342 |
| Gcc | 2-way associative | 4 | 2 | 128 | 0.862165 |
| Gcc | 2-way associative | 2 | 2 | 256 | 0.799448 |

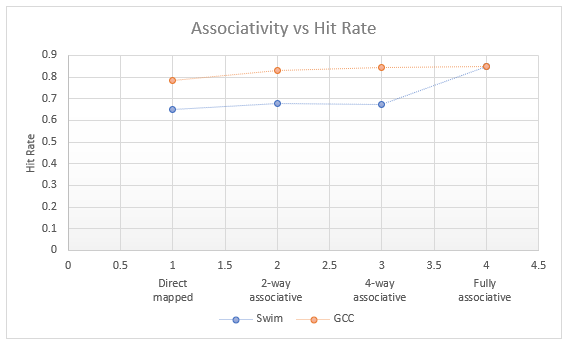
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Trace File | Mapping | Sets in Cache | Blocks in Set | Bytes in Block | Hit Rate |
| Swim | Fully associative | 1 | 16 | 64 | 0.760394 |
| Swim | Fully associative | 1 | 8 | 128 | 0.71078 |
| Swim | Fully associative | 1 | 4 | 256 | 0.680761 |
| Gcc | Fully associative | 1 | 16 | 64 | 0.92603 |
| Gcc | Fully associative | 1 | 8 | 128 | 0.892131 |
| Gcc | Fully associative | 1 | 4 | 256 | 0.812602 |

**Results**

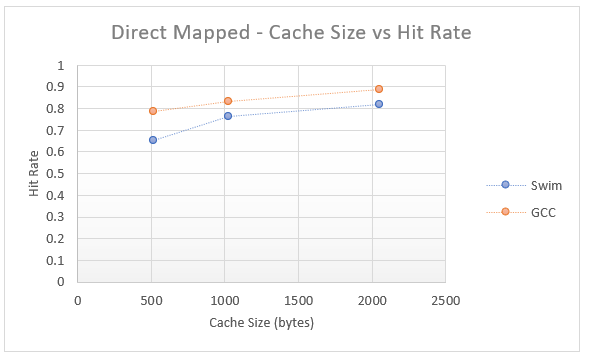
Evaluating my first test results shows that a direct mapped cache has the lowest hit rate out of my three choices. However, it is the fastest to access, since checking whether a value is in the cache happens in O (1) time.

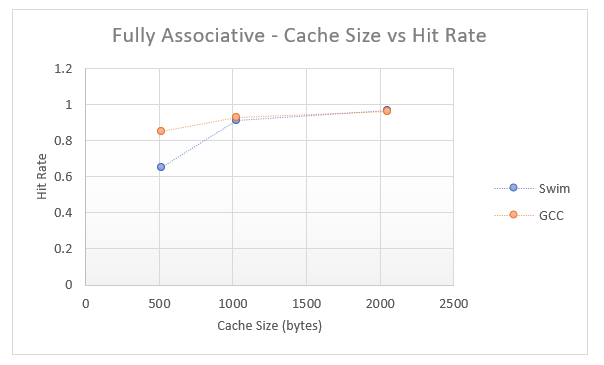
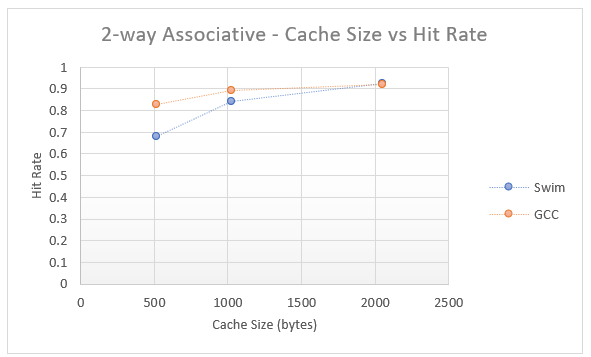
An n-way set associative cache has performs better than a direct mapped cache, since increasing the associativity decreases miss rate. Because there is more than one line per set, an n-way associative cache decreases the number of conflict misses, or conflicts caused by addresses competing for the same location. However, as associativity increases, so does the access time.

A fully associative cache has the highest hit rate of the three cache designs considered. A fully associative cache also has the longest access time, O (n).

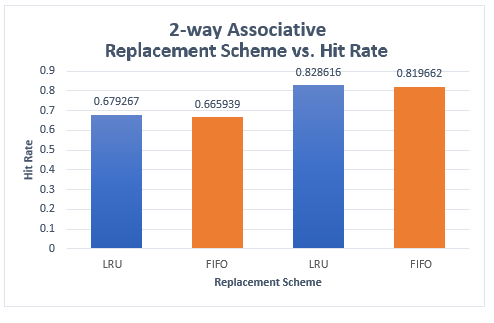


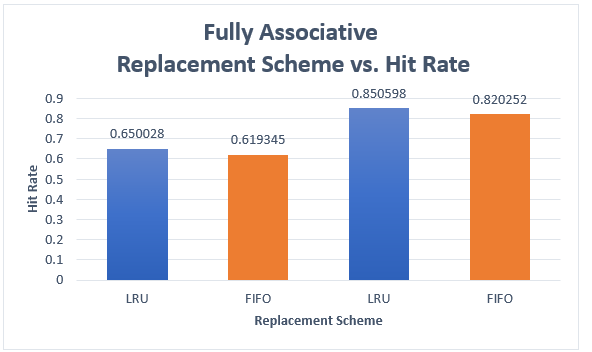
The second test’s results show that for a direct mapped cache, as size increases, hit rates improve. A larger cache tends to decrease the chances that two blocks will map to the same line, decreasing conflict misses.



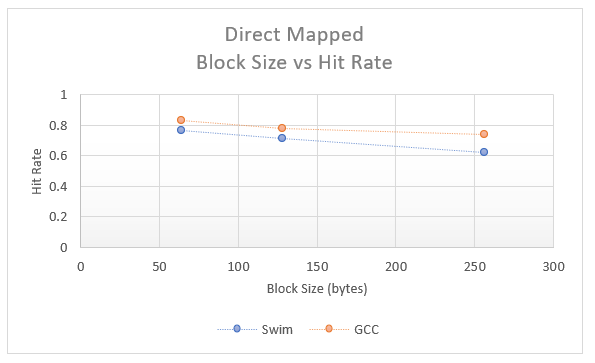


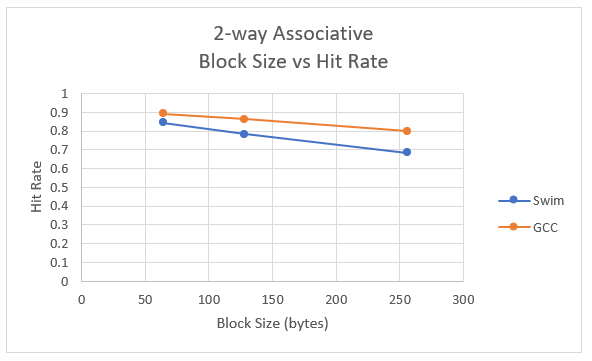
Test 3 shows that for both N-way set and fully associative caches, LRU is a better replacement scheme than FIFO. For both files, the hit rates fall when using FIFO rather than LRU.

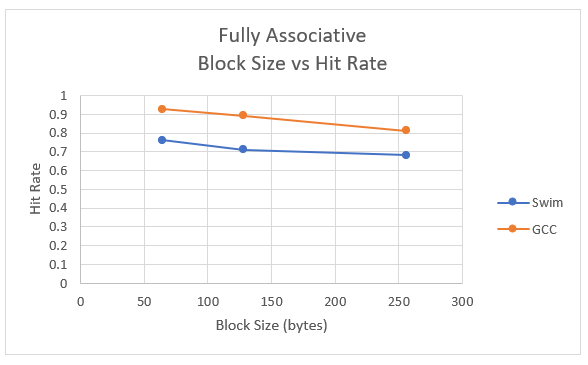




Test 4’s results show that in general, increasing the block size causes the hit rate to decrease. The amount by which the hit rates decrease varies between the cache mappings and trace files, but is consistently lower.

****

****

****

**Conclusions**

Direct mapped has the lowest hit rates and fully associative caches have the highest hit rates of the three caches considered. Increasing the degree of associativity increases the hit rate because it reduces the number of conflict misses caused by addresses competing for the same location.

In general, increasing cache size can reduce conflict and capacity misses, but it does not affect compulsory misses. A larger cache improves hit rates for all three cache designs, but for N-way and fully associative caches, an increase in size also decreases the access time.

For both N-way set and fully associative caches, least recently used replacement policies work better than first-in first-out. LRU replacement schemes take advantage of temporal locality, while FIFO does not. FIFO will discard a block there is no room for, even if it has been recently used. The difference between the hit rates for a FIFO and LRU varies, but FIFO is consistently lower.

Increasing the block size decreases the hit rate. Theoretically, increasing block size could reduce compulsory misses, since the cache can hold more information in fewer blocks, and fill in less time. However, increasing block size also increases conflict misses because more addresses map to the same set, which outweighs the benefits of a larger block.