Introduction

The cache simulation was designed and implemented using different data structural designs in C++ to gather data and compare the performances of three different cache designs; direct-mapped, n-way associative, and fully associative. Additionally, the performance will be tested based on the changes of the size of the cache and the replacement policies for n-way associative and fully associative. The simulation reads in trace files in order to obtain the hexadecimal addresses. After converting the addresses from hexadecimal to binary, different algorithms are used to simulate the three different cache designs. Depending on the design, these algorithms will isolate the tag, set, and offset, and use these found values to calculate the hit rate for each design.

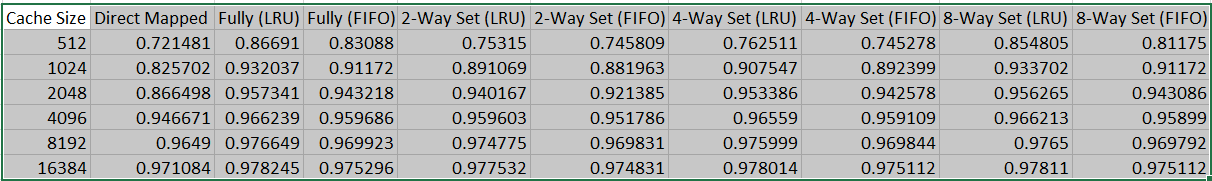
Description of Tests

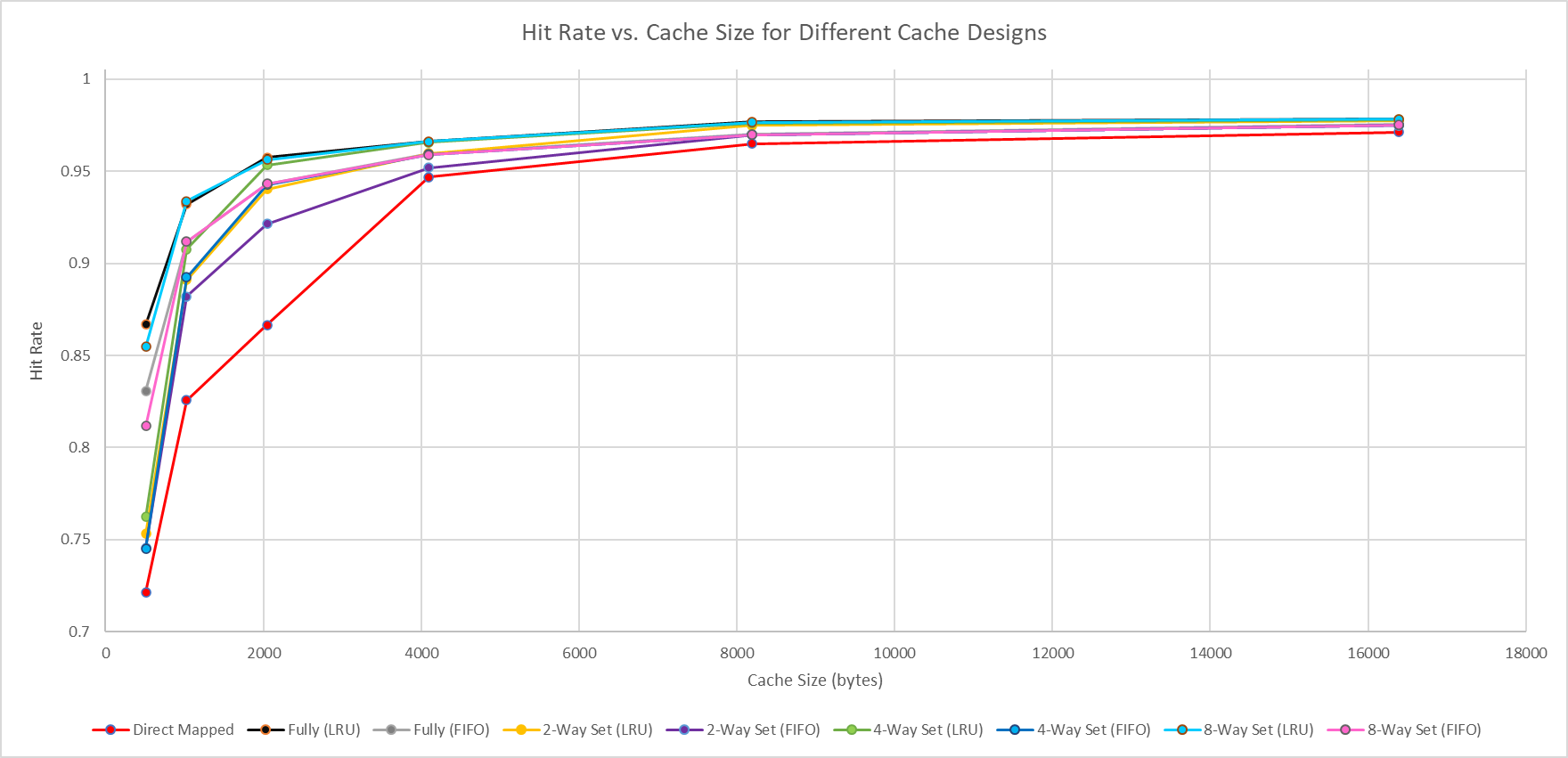
The parameters for the Direct Mapped design were the cache size and the bytes per block. The parameters for the Fully Associative design were the cache size and the bytes per block in addition to the replacement policy which affected the performance. Replacement policies are important for simulating cache designs as it is needed to see how the techniques to replaces blocks that are already used affect the overall performance. Finally, the parameters for the Set Associative design were the ones previously mentioned as well as the value of n for set way. The n values used were 2, 4, and 8. Since Set Associative is a balance between Fully Associative and Directly Mapped, it was important for these tests to have multiple markers to examine in between a one way (Directly Mapped) and B-way (Fully Associative.)

For the cache size, powers of 2 were chosen as data is stored in binary which makes it more efficient to use memory addresses, which can also be applied to bytes per block. I chose 6 different cache sizes, 2^9 to 2^14 because it shows a wide range of hit rates while also accurately showing the comparisons of each design as the cache size goes up. Since the increase in hit rates isn’t linear with increase in cache size, it was important to show a wide range of results.

Results

Here are the results of the hit rates for each different configuration. The left column shows the different cache sizes and the row shows the correlation between the cache size and the different design configurations.

 The graph below shows the hit rates for the different cache designs over the cache size. The x-axis of the graph is the cache size, and the experimented values range from 2^9 to 2^14. The y-axis shows the hit rate, ranging from the lowest found value of 0.721481 for a Directly Mapped design at 512 bytes per cache, to the highest found value of 0.978245 for a Fully Associative design with the Least Recently Used replacement policy at 16384 bytes per cache.



Conclusions

It’s very noticeable that Direct Mapped cache design has the lowest performance during these tests. The Direct Mapped design has the advantage of being the simplest to implement, but suffers from significantly reduced hit rates resulting from increased conflict misses. From the results Fully Associative has the highest hit rate consistently. This is due to its allowance of placing each memory block in any cache location. Additionally, the results from Set Associative cache design shows that higher n values yield to increased performance. 8-way Set Associative results show its superior hit rates than the reduced values of n. This shows that the choice of n should be considered when implementing a cache design, as higher values of n reduce conflicts between blocks.

The results today show that Least Recently Used replacement policy consistently performs better and outputs higher hit rates than First In First Out. This is due to the fact that LRU avoids blocks that are more likely to be accessed again, while FIFO does not take into account the how frequently a block has been accessed. LRU being aware of how often blocks are accessed allows it to have better memory management for cache designs.

Having an increased cache size is very useful in increasing the hit rate substantially, allowing for the cache to have an improved performance. However, there is a certain point where increasing cache size has diminishing returns. The results show that hit rates increase a lot from 512 to 4096 bytes per cache, but after that the hit rate increasing rate slows down considerably. Increasing a cache size means using up more resources to perform, so it’s important to recognize at what points are there better performances in cache size and optimize cache size properly in order to have a good balance of resource usage and performance.