Charles Richardson

Cache Simulation Analysis

Introduction

A cache is a low-maintenance tool, once built. It takes 4 pieces of input, and can efficiently handle memory transfer and access with just 4 inputs. These inputs describe what the cache type is, and have a dramatic impact on the performance of the system, and whatever system depends on it.

My simulation is designed to have plenty of feedback. The user is specified to input the parameters from the terminal, where it is expected to be run, the program displays the inputted values as confirmation and displays the hit rate as a percentage. This improves readability and ease of use.

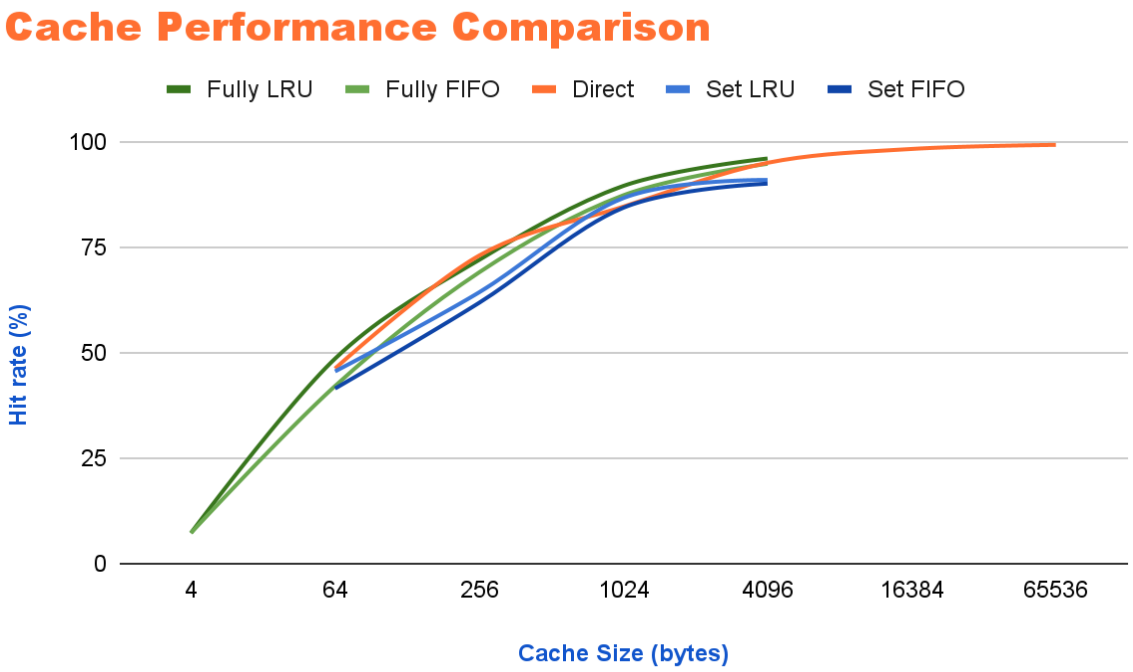
Description of tests

Testing the cache was simple. The inputs were quantized. They were required to be multiples of 2. The set and block inputs could start from 1, while the bytes per block had a minimum value of 4. For the set and tag, inputs choose parameters between the range of 1 and 256. I did this because they were restricted by 1 on the bottom end, and once the values got to 256, the hit rate was virtually 100%. On the bytes per block, I went all the way to 1024 to test the limits of the cache. Running these numbers with 64 sets and 256 blocks/set, the program took over a minute to complete execution. I decided not to go further than this for the sake of time.

Results

Expectedly, as the parameters of the cache increased, so did the hit rate. From a 7.3% hit rate with the minimum parameters to 99.6% with values more than 10-fold, the more a cache can hold, the more hits it will have. The lowest hit rates came from the fully associative cache, specifically using FIFO as a replacement policy. The direct-mapped (DM) cache is not significantly better than the fully associative, but it is easier to analyze given it does not have a replacement policy. What I observed from the DM cache is that the number of sets and the number of bytes/block has about the same weight on the hit rate. Increasing one by 4-fold will make the same difference as increasing the other by 4-fold.

Set associative reached the highest hit rate, but not by a lot. It took large parameters to reach the high levels of hits, as well as plenty of processing power and time. Testing this cache took quite a bit of time. In particular, the FIFO policy took longer. This would make one think it is better, but the results show that it is marginally worse than LRU. In fact, FIFO performs worse for both fully and set associative compared to LRU.



Conclusions

What the graph above shows me is that the different cache types make minute differences. All three cache types perform at about the same level when the cache size is kept equal. In terms of development, the direct mapping cache is the easiest to build in software, but I recall it being harder to develop in hardware. Therefore, there is no best cache type for a general user. Cases may arise where a specific cache is optimal for specific use cases, but that is outside of the scope of this analysis.

The replacement policies play a significant role when compared with each other. As mentioned earlier, the FIFO replacement policy consistently performed worse than its counterpart LRU. The difference was more dramatic at smaller cache sizes, and it usually converges at extremely large cache sizes. Cache size is the most significant determinant of the hit rate. The more space in the cache, the more slots it has available for blocks. Similar to the replacement values, as the size of the cache approaches infinity, the performances of the different cache types converge at 100%.