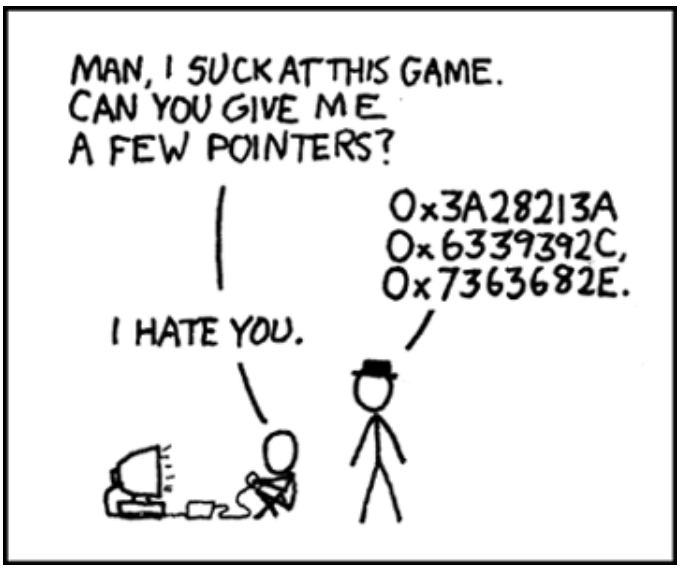
Aaron Mertz and William Haines

Cache Simulator



ECEN 4593

May 5th, 2014

1 Introduction

Modern processors are capable of running at very high clock rates. However, this doesn't do any good if it spends most of its time waiting for slow accesses to main memory. For this reason, a CPU typically has a small amount of memory built in that runs as fast as it does, called a cache. The diminutive size of the cache means that you have to be very careful about how you design it.

Generally, the total cache capacity is on the order of 1,000 to 10,000 times smaller than main memory. Despite this, it can still have a high hit rate due to spatial and temporal locality of memory references in code. A small drop in hit rate can cause a significant increase in the execution time of a program, so it's crucial for a cache to store the right data.

2 Cache Performance

2.1 Overview

The performance of a cache mainly depends on its size and associativity. An increase in either generally improves the performance of a cache. For both, the law of diminishing returns is true. You have to strike a balance between cost and performance. In order to get a general sense of the performance of the cache, the execution time normalized to the ideal execution time is presented below. The reason we normalize by ideal execution time is to get a better sense of how close a particular configuration is to achieving its ideal execution time. The results for each configuration average the results for the five production traces.

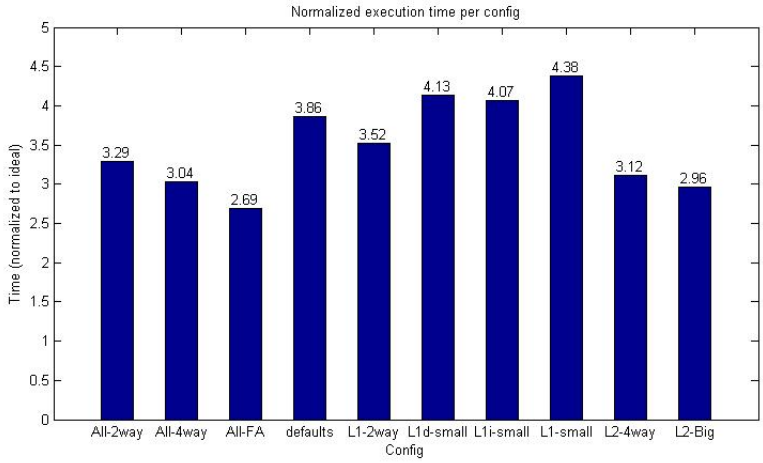


Figure 1: Execution Time Normalized to Ideal

In Figure 1, All-FA is the clearly the closest to matching the ideal performance time, being only a few times larger. L2-Big and L2-4way also come close to ideal, but not as close as All-FA. We will present later why larger associativity is correlated with better performance. The worst performing configuration by far is L1-small, which makes sense. Small L1 caches mean that there are more requests to L2, and in turn more main memory accesses when there is an L2 miss.

2.2 Performance vs Associativity

In a direct mapped cache, you have a problem when part of a program repeatedly references multiple blocks in memory that map to the same cache block. It will constantly have to kick one out to load the other, and actually make things slightly slower than they would be without a cache at all.

This situation can be avoided by allowing each cache block to keep track of two blocks that have the same index. Because the index is made from the least significant bits, consecutive locations will not

share an index. Adding ways solves a problem that does not occur very frequently, so you only see a marginal improvement.

A fully associative cache always kicks out the least recently used block in the entire cache when it is full and needs to add a new one. This gives it the highest hit rate of all the configurations that we tested. Unfortunately, it is so expensive that you would almost always get better performance per dollar by simply making the cache larger.

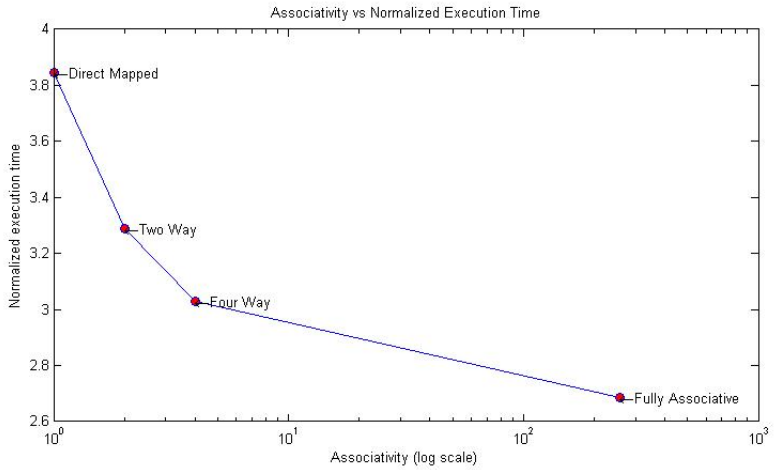


Figure 2: Associativity vs Normalized Execution Time

2.3 Performance vs Size

The effects of size are more direct than associativity. A bigger cache holds more total blocks, so it can have a higher hit rate. A cache that is too small will not be able to hold all of the references in a loop, which will give you horrible performance. On the other hand, once your cache is large enough to have a high hit rate ($> 90\%$), making it bigger doesn't offer any noticeable improvement in execution time. You can see this effect in Figure 3, where the big L2 cache is much faster than the default, which is much faster than the small L1.

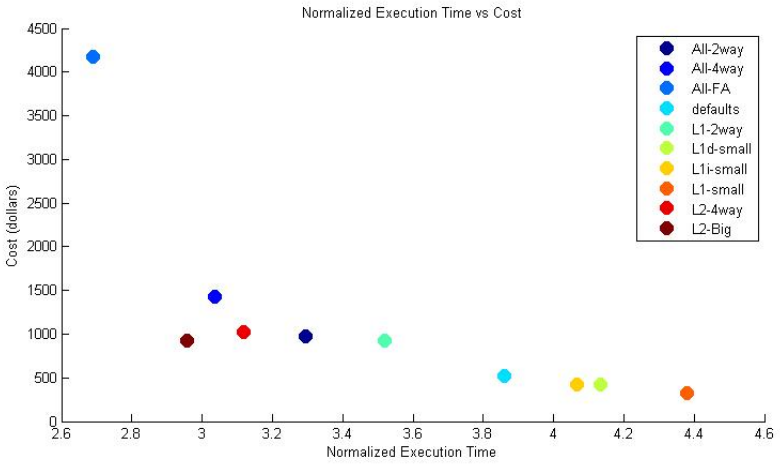


Figure 3: Normalized Execution Time vs Cost

3 Execution Time vs Cost

Another metric to determine cache effectiveness is analyzing the relationship between time and cost. Figure 3 shows this relationship. While All-FA performs the best, it also costs 4 times more than the next best option. In terms of performance for cost, L2-Big is the best configuration. A larger L2 cache is much cheaper than larger L1 caches, and gives slightly better performance. The next best option for minimizing cost and maximizing performance would be the default configuration, which gives the best execution time for a low cost.

3.1 Generally Optimal Cache

Chips built for different applications have different needs from their caches, such as low cost and power consumption for a mobile chip, or extremely high hit rate for a server chip. Even with these differences, the general strategy for maximizing performance within your budget is the same. You want to select parameters past the steep part of the graph, where you get a significant improvement for your money, but not too far down the flatter part. A budget cache might be 2-way set associative, and you might use 4-way for high performance, but it would never be practical to use fully associative or direct mapped. In general larger caches provide better performance. However, in real life the cache size cannot grow too large, as larger caches have higher access times. This is not reflected in our data, as access times are not scaled with size.

4 Changing Memory Chunk Size

For this simulation, memory chunk size influences the speed at which L2 can grab something from main memory. With a higher chunk size, an L2 miss takes less time. We ran omnetpp with all configurations using memory chunk sizes of 16, 32 and 64. These results are presented in the appendix. The plot for comparing each configuration's memory option were identical. The price of the cache increases, and the execution time decreases. In Figure 4, the default configuration is shown using mem chunk sizes of 16, 32 and 64. A larger chunk size costs more, but also has a very potent affect on performance. A memory chunk size of 64 is a 26% improvement in performance for a 17% increase in cost.

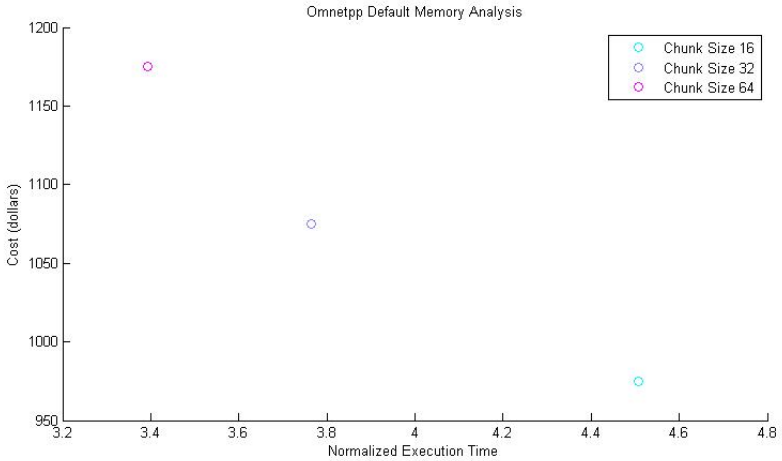


Figure 4: Changing Mem Chunksize Analysis

5 Hit Percentage

The figures below show the hit percentage of each cache compared to the ideal time divided by the total time. This shows us how strong hit percentage is correlated to ideal execution speed. For Figure 5 and 6 the results are straightforward. A higher L1 hit percentage means the time to complete that cache

configuration will be less and thus the configuration will be closer to ideal. However, for Figure 7 a higher L2 hit percentage does not necessarily mean the configuration will reach a high percentage of ideal time. The two worst performing configurations have the highest L2 hit percentage. This makes sense because a smaller L1 cache (either both L1, or just data) will cause more L2 requests and more hits, but will still take longer.

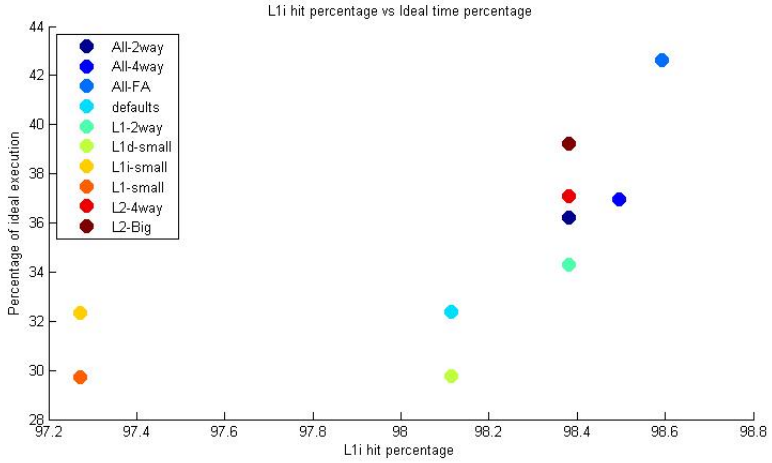


Figure 5: Percentage of L1i hits compared to the ratio of total time to ideal time per configuration

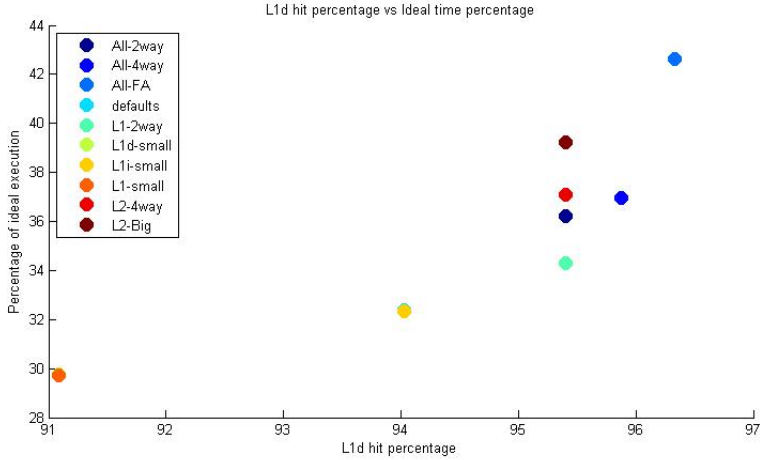


Figure 6: Percentage of L1d hits compared to the ratio of total time to ideal time per configuration

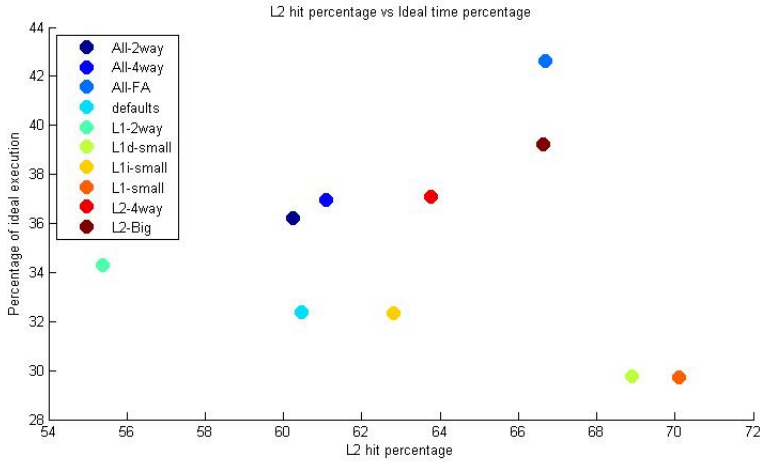


Figure 7: Percentage of L2 hits compared to the ratio of total time to ideal time per configuration

6 Cycles per Activity

In Figure 8 the cycles for each type of activity and each configuration are shown. These results by and large reflect all we have shown so far. In general All-FA takes the least amount of cycles for any kind of activity. Although each trace had a different instruction, read and write count, All-FA followed by L2-Big and L2-4Way performed the best.

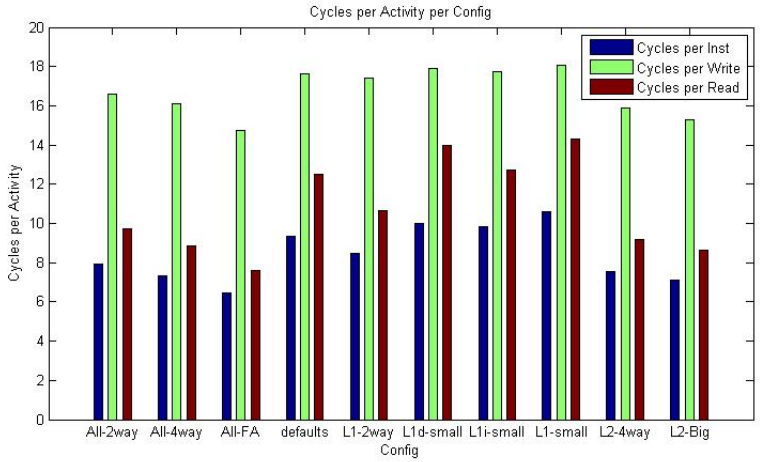


Figure 8: Overall Cycles Per Activity for each Configuration

7 Conclusion

Cache performance is a crucial part of system performance. Amdahl's law tells us that the benefit of speeding up part of a process is proportional to the time spent doing it. Processors generally spend a lot of time reading and writing data, so speeding that up can bring huge improvements.

A cache has many attributes that contribute to its performance. In order to understand how fast any configuration would be, you have to run it in a simulator. This way, you can examine many different options and choose the best one before you actually try to design a circuit in your favorite HDL.