Jeremy Sommerfeld

cs3421 - Spring 2015

Lab 8 – Cache Analysis

There are a number of factors that contribute to how a processor cache performs. The main parameters are total cache size, block size, associativity, the replacement policy and the write policy. Through variations in these parameters, we can alter the performance of the cache. For this lab, I decided to alter the total size of the cache and its set associativity.

For the first test, the only thing that was altered between tests was the overall size of the cache. All of the other parameters were kept constant. For these tests, I used a cache with a block size of 8, a set associativity of 2, least recently used replacement policy, and a write back write policy. I also assumed a cold start up would be used, as this would help to give better real world performance. I tested all cache sizes of all powers of two from 1K to 1M.

As we can see from the graph, increasing our cache size has a positive impact on the overall hit rate of the processes run. Some processes, due to the way they are inherently written, are able to show a much greater increase over others. For example, the cc1 process trace starts out much lower than the others at a lower cache size, but the increases in the size available drastically improve the hit rate. Others that have better initial performance still see increases in performance, but the performance gains become negligible at higher sizes. Past 256K, the only increases are on the order of tenths of a percent. This helps us to understand why modern processors don’t have giant L1 caches. The difference in the amount of cycles needed to read the cache and the hit rate becomes a loss past these values. This helps to explain why all of the modern Intel architectures all have an L1 cache size of 64K. This seems to be the sweet spot for hit rate to cycle time availability.

Total cache size is not the only factor in a cache’s hit rate. For the next tests, I kept the cache size at the Intel standard of 64K and altered the associativity of the caches instead. I used a block size of 8, a least recently used replacement policy, and both write though and write back caches. For the write though cache, I used an update write-hit policy, and an allocate write-miss policy. I again assumed a cold startup policy for these tests.

As we can see, increasing the associativity also helps to increase the hit rate in the cache. However, due to how higher associations are implemented, a higher associativity costs a *lot* more to implement over the hit rate performance benefit. In my tests, a 4 way set associative cache seems to be the sweet spot for the complexity to hit rate trade off. This trade off of efficiency can be seen in the real world, as Intel uses mainly 4 and 8 way associativity in all of their caches. On the other hand, AMD’s Athalon architecture uses a 2-way associative cache.

Optimizing the hit rate of a cache is a great way to increase the performance of a processor. Simple changes that increase the hit rate by even a few percent can help provide drastic performance gains. One way to increase the hit rate is to scale up the size of your cache. An increase in cache size slows down the access time, so it is vital to have a small, but incredibly fast Level 1 (L1) cache. Modern processors add in larger L2, L3 and even L4 caches in order to keep the cache hit rate as high as it can be. Cache performance is one of the key areas for modern processor performance optimization as we approach the frequency limits of silicon.