Jonah Fidel

12/12/14

COSC 201

Lab 10

**Version 1:**

1. Direct, 256, 1
   1. memory access count: 1088
   2. cache hit count: 896
   3. cache miss count: 192
   4. hit rate: 82%

Performance: A direct mapped cache with one word per line and 256 blocks has pretty mediocre performance. This is because there are so many individual blocks that must be allocated. More specifically, a word can go in exactly one location and there is a separate tag for every word. Performance improves when words per line increase and block decrease.

1. Direct, 64, 4
   1. memory access count: 1088
   2. cache hit count: 1039
   3. cache miss count: 49
   4. hit rate: 95%

Performance: much better than a direct mapped cache with only one word per line. With a hit rate of 95%, this cache is extremely efficient. Having four words per line allows a greater margin for error in writing to the cache.

**Version 2:**

1. Direct, 64, 4
   1. memory access count: 1088
   2. cache hit count: 788
   3. cache miss count: 300
   4. hit rate: 72%

Performance: The reason performance is so poor with this cache is because it uses so few blocks overall. With only 64 blocks to fit 4-block words into, there is a very small margin for error and misses occur very frequently.

1. SA, 4, 64, 4
   1. memory access count: 1088
   2. cache hit count: 1037
   3. cache miss count: 51
   4. hit rate: 95%

Performance: A set associative cache dramatically improves performance here. This is because unlike the direct mapped cache, a set-associative cache will allocate by sets of words instead of each word individually. The fact that there is a size of 64 bits here conveniently complements the 4-block word and set size. Since words and sets are the same size, blocks are very easy to allocate.

**Version 3:**

1. SA 4, 64, 4
   1. memory access count: 8448
   2. cache hit count: 7323
   3. cache miss count: 1125
   4. hit rate: 87%

Performance: This cache implements the exact same specifications as version 2 part 2 but for a much larger matrix. The larger size of the matrix means there is less precision when placing sets and words in the cache. With so much more area to the matrix, it is much easier to miss cache hits.

**Version 3a:**

1. 4x4
   1. memory access count: 8448
   2. cache hit count: 7105
   3. cache miss count: 1343
   4. hit rate: 84%

Performance: the implementation of the matrix as 4x4 means each placement of blocks must proceed in conjunction with the multiplication algorithm. This is not ideal for allocating blocks as it allows for much greater chance of error. Considering the unique implementation, however, there is not much of a dropoff from the previous set-associative cache in terms of performance.

How would more words per line impact performance?

* More words per line likely enhances performance, especially in conjunction with set-associative caches because set sizes can match line sizes which vastly reduces chance of error.

Set associative caches in general have better performance than direct mapped caches simply because more data can be allocated at once meaning there is less chance for error overall. A two way set associative cache would likely not work as well for the programs described here because it means fewer blocks are allocated at once. Additionally, most of the implementations here use 4 block lines, which is more convenient for a four way set associative cache.

Reorganizing data could help improve performance. Namely, the closer the data is arranged to its eventual layout in the cache, the faster performance will be. In larger matrices this technique becomes less effective because the time it takes to organize the data is exponentially higher. That said, with very large matrices, data reorganization is likely unhelpful.

A parallel implementation of caches running on 4 processors would likely improve performance tremendously. Multiple blocks could be allocated at once, making the process much more efficient. Of course the actual implementation of parallel processors for matrix multiplication would be extremely complicated mathematically, but theoretically performance could still be enhanced. One arrangement that might work for such a setup is to implement a large matrix across four processors. For example with an 8x8 matrix you could implement 4 4x4 matrices, one for each processor.