Aroon Sharma

ENEE 646

11/3/13

Project 3 Performance Evaluation

\*Simulation results and graphs are shown in asharma4.xlsx

*Working Set Characterization*

1. This experiment is testing the hit rate for the instruction and data caches as we vary the cache size, keeping all other cache parameters constant. We take away conflict misses by making all caches fully associative. Because of temporal locality, we expect that the hit rate will increase as we increase the cache size. Increasing the cache size allows more memory reference history to be in the cache at a given time. Therefore, if we reference an address that we have accessed before, it is more likely to be in the cache. Looking at the plots for each trace, we see similar trends in that the hit rate increases as cache size increases, but after a certain cache size there is no more increase. This is because applications only have so much temporal locality to exploit. Our caches in this experiment are still subject to compulsory misses.

2. We calculate the working set size of the instruction cache and data cache by increasing the cache sizes of each cache until the number of misses for each cache stops decreasing. By doing so, the only misses in the cache will be compulsory misses (misses that fill up the cache). The working set is then the number of compulsory misses at this point times the block size of the cache. The table below summarizes the working sets for each trace:

|  |  |  |
| --- | --- | --- |
| Trace | I-cache working set (bytes) | D-cache working set (bytes) |
| spice.trace | 35856 | 16900 |
| cc.trace | 124780 | 37092 |
| tex.trace | 636 | 38088 |

*Impact of Block Size*

1. For each trace, we see that the hit rate increases as you start to increase the block size, but after the block size becomes too big, the hit rate actually decreases. This is the case for both instruction and data caches. Increasing the block size for takes advantage of spatial locality. Every time we bring an entry into the cache, we also bring in the next few blocks as well. However, if the cache block size is too big, conflict misses become a bigger problem and effect hit rate. This is because there is a greater chance of evicting a useful block from the cache.

2. The optimal block size corresponds to the block size that has the highest hit rate for each trace. These values are summarized in the table below and can be confirmed by the data taken in Performance Evaluation.xlsx.

Optimal block sizes for instruction and data caches

|  |  |  |
| --- | --- | --- |
| Trace | Instruction cache | Data cache |
| spice.trace | 1 KB | 32 bytes |
| cc.trace | 2 KB | 32 bytes |
| tex.trace | 2 KB | 128 bytes |

3. Clearly the optimal block sizes for instruction and data caches are different. Instruction caches have much larger optimal block sizes than data caches. This suggests that instruction references exhibit much more spatial locality than data references.

*Impact of Associativity*

1. The hit rate vs. associativity plots generally show gradual increase in hit rate as associativity increases. However, after a certain point, hit rate stops increasing and levels out. This happens because increasing associativity is taking advantage of conflict misses in the cache. Programs generally have only so many conflict misses that after a certain point, increasing associativity will do nothing.

2. There is a difference between the plots for data and instruction references. Increasing associativity has a greater effect on data references than instruction references. That is, the hit rate on data references increases faster than instruction references as you increase associativity. This is because instruction references generally refer to a continuous segment of memory, whereas data references can happen from multiple segments of memory all over the place.

*Memory Bandwidth – Write back vs. Write through*

1. Write back caches have the smaller memory traffic for all simulations. If we only vary write-back vs. write through, demand fetches will be the same, but copies back will always be smaller for write back caches. This is because write back caches only write to main memory on an eviction, whereas write through caches write to main memory and the caches on a cache hit. Write back caches are banking on the fact that if you write to a block in memory you are likely to write to the same block again, so it is advantageous to aggregate the writes to main memory.

2. There would never be a case where write through cache has less memory traffic than the write back cache. Assume the block sizes for both caches are 4 bytes (the size of a word). No matter what program you consider (as long as it has one write) the write through cache will write once to main memory when the write occurs and once when the cache is flushed. The write back cache will only write back to main memory once. You can expect double the memory traffic (for copies back) for a write through cache compared to a write back cache in this case.

*Memory Bandwidth – Write allocate vs. No write allocate*

1. According to the simulation results, one policy is not better than another. Sometimes write allocate performs better in terms of smaller memory traffic and sometimes no write allocate performs better. This occurs because this policy is dependent on what program you are running. If you are running a program with one or few writes on each cache block, no write allocate will perform better. If you are running a program with many writes on each cache block, write allocate will perform better.

2. Yes, my simulation results show that for simulation 1, write no allocate has the smaller memory traffic, but for simulation 3, write allocate has the smaller memory traffic. See Performance Evaluation.xlsx and the bash scripts memory\_bandwidth\_3.sh and memory\_bandwidth\_4.sh for details on each simulation configuration and their memory traffic results.