EEL4768: Computer Architecture

Project One:

L1 Cache Simulator

Ryan Koons 11/01/2020

**Compilation Instructions:**

//"gcc cache\_sim.c -o SIM -lm"

//"./SIM <Cache\_Size> <Associativity> < Replacement Policy> <Writing Policy> <TraceFile>"

//NOTE: -lm IS NECESSARY SINCE math.h IS USED

**Note: Part A-C Were analyzed using LRU Policy. Part D Compares/contrasts with FIFO Policy.**

**Part A) The effect of Cache size on the Miss Ratio**

|  |
| --- |
| **Part A) How does miss ratio change with Cache size (XSBench and MiniFE)** |
| N=4, Write-back policy, LRU Policy |
| ./SIM <CACHE\_SIZE> <4> <0> <1> <TRACE\_FILE> |
| Vary Cache size from 8KB to 128KB (multiples of 2) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Cache Size(KB)** | **Cache Size (Bytes)** | **XS Bench Miss Ratio** | **MiniFE Miss Ratio** |
| 8 | 8192 | 0.136261 | 0.080703 |
| 16 | 16384 | 0.119363 | 0.074231 |
| 32 | 32768 | 0.112539 | 0.065595 |
| 64 | 65536 | 0.108198 | 0.059669 |
| 128 | 131072 | 0.104894 | 0.055829 |
|  |  |  |  |
|  |  | **XS Bench Miss %** | **MiniFE Miss %** |
|  |  | 13.6% | 8.1% |
|  |  | 11.9% | 7.4% |
|  |  | 11.3% | 6.6% |
|  |  | 10.8% | 6.0% |
|  |  | 10.5% | 5.6% |

**Analysis: I**ncreasing the cache size caused the miss rate to decrease. This result was evident for both tracefiles. This logically makes sense since a large cache will ensure that there is more overhead to store data; therefore, less evictions will take place and most data will remain on the cache (resulting in fewer cache misses). The XSBench tracefile proved to have a higher miss ratio than the MiniFE tracefile or all Cache Sizes. Both files saw a sharp decrease in miss rate initially that gradually flattened out over time.

**Part B) Comparison of Writing Policy**



**Analysis:**  For both files WB policy seemed to be the better choice as far as writing policy goes. This also makes sense since the WB policy only writes to memory when the corresponding block is dirty (updated). This saves a lot of instructions in comparison to the WT policy which writes to memory for every single write. Both have their advantages-- such as the WT ensuring the memory is always up to date, and the WB policy saving on performance. With WT Policy, the number of writes never changed (regardless of cache size). This is certainly a disadvantage since it is affecting the performance of the code in a negative manner. However, the number of WT Reads did decrease with cache size (which makes sense since the writing policy does not have an effect on the reads). This also explains why the number of reads is identical (for each corresponding cache size) regardless of writing policy. In WB policy, the number of writes did decrease as cache size increases. This makes sense in a similar manner to part A—a larger cache ensures less evictions, and there for less writes back to memory. Again, the MiniFE tracefile proved to have better performance results than the XSBench tracefile. In fact, it would execute in about 1/10th of the time that the XSBench file would take. Since WB policy had a significantly smaller number of writes to memory, it is the preferable writing policy.

**Part C) Varying the Associativity**

|  |  |  |
| --- | --- | --- |
| **Vary the Associativity 1 to 64 (in multiples of 2)** | | |
| Cache Size--32KB, LRU Replacement policy, **(ASSUME WB POLICY, not stated in project descr.)** | | |
| ./SIM <C32768> <x> <0> <1> <TRACE\_FILE> | | |
| **assuming project description indicates comparing miss ratio for various associativity values** | | |
|  |  |  |
|  |  |  |
|  |  |  |
| **Associativity** | **XS Bench Miss Ratio** | **MiniFE Miss Ratio** |
| 1 | 0.122697 | 0.075406 |
| 2 | 0.114457 | 0.066151 |
| 4 | 0.112539 | 0.065595 |
| 8 | 0.112008 | 0.065292 |
| 16 | 0.111906 | 0.06543 |
| 32 | 0.111784 | 0.065583 |
| 64 | 0.111753 | 0.065666 |
|  |  |  |
|  | **XS Bench Miss %** | **MiniFE Miss %** |
|  | 12.27% | 7.54% |
|  | 11.45% | 6.62% |
|  | 11.25% | 6.56% |
|  | 11.20% | 6.53% |
|  | 11.19% | 6.54% |
|  | 11.18% | 6.56% |
|  | 11.175% | 6.57% |

**Analysis:** Test runs of the two tracefiles clearly indicated that the increasing the associativity mostly decreases the miss rate. However, it was seen, in MiniFE specifically, that a threshold can be reached in which increasing the associativity further will cause the miss rate to increase very slightly. This is certainly important-- the goal is to increase the associativity to achieve the absolute minimum value of the miss rate; it would be counteractive to involuntarily increase the miss rate during this process. Testing showed that the most apparent diminishing of the miss rate occurs at the very beginning of increase the associativity. This rate then flattens out, as the associativity continues to increase. Again, the MiniFE had a much lower miss rate than the XSBench file.

**Part D) Analyzing the effect of replacement policy**

**Section 1: FIFO and Write back policy**

|  |  |  |  |
| --- | --- | --- | --- |
| **Part D (Section 1): Part B , FIFO Policy, Write-back** | | |  |
| **memory reads and writes** | |  |  |
| Vary Cache size from 8KB to 128KB (multiples of 2) | | |  |
| N=4, FIFO Policy, 1--write back | |  |  |
| ./SIM <CACHE\_SIZE> <4> <1> <1> <TRACE\_FILE> | | |  |
| program prints writes first, followed by reads | | |  |
|  |  |  |  |
| **XSBENCH** | | | |
|  |  | Write Back | |
| Cache Size(KB) | Cache Size (Bytes) | WB Writes | WB Reads |
| 8 | 8192 | 152689 | 3150161 |
| 16 | 16384 | 66019 | 2772187 |
| 32 | 32768 | 32613 | 2555217 |
| 64 | 65536 | 15806 | 2408529 |
| 128 | 131072 | 7707 | 2303098 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| **MINIFE** | | | |
|  |  | Write Back | |
| Cache Size(KB) | Cache Size (Bytes) | WB Writes | WB Reads |
| 8 | 8192 | 89078 | 415633 |
| 16 | 16384 | 80380 | 368833 |
| 32 | 32768 | 73880 | 330181 |
| 64 | 65536 | 68497 | 301067 |
| 128 | 131072 | 62284 | 277784 |

**Analysis:** With FIFO replacement policy, the number of write back writes and reads decreased as the cache size increased. For XSBench, the number of WB writes decreased exponentially as cache size increased. With FIFO policy, the MiniFE Tracefile had a smaller number of writes and reads than the XSBench tracefile. It makes sense that XSBench was much worse since it has a lot more addresses. In FIFO policy, the number of misses will increase a lot and so will the number of reads and writes to memory. LRU policy definitely is a better choice since it is more up to date with the instructions and therefore has a lower number of memory reads and writes.

**Section 2: FIFO Associativity**

|  |  |  |
| --- | --- | --- |
| **Part D Section 2) Vary the Associativity 1 to 64 (in multiples of 2)** | | |
| Cache Size--32KB, FIFO Replacement policy, **(ASSUME WB POLICY, not stated in project desc.)** | | |
| ./SIM <C32768> <x> <1> <1> <TRACE\_FILE> | | |
| **assuming project description indicates comparing miss ratio for various associativity values** | | |
|  |  |  |
|  |  |  |
|  |  |  |
| **Associativity** | **XS Bench Miss Ratio** | **MiniFE Miss Ratio** |
| 1 | 0.122697 | 0.075406 |
| 2 | 0.12061 | 0.068023 |
| 4 | 0.120672 | 0.067743 |
| 8 | 0.120891 | 0.067609 |
| 16 | 0.121221 | 0.06807 |
| 32 | 0.121125 | 0.068227 |
| 64 | 0.121212 | 0.068275 |
|  |  |  |
|  | **XS Bench Miss %** | **MiniFE Miss %** |
|  | 12.27% | 7.54% |
|  | 12.06% | 6.80% |
|  | 12.07% | 6.77% |
|  | 12.09% | 6.76% |
|  | 12.12% | 6.81% |
|  | 12.11% | 6.82% |
|  | 12.121% | 6.83% |

**Analysis:** FIFO Policy showed once more that increasing the associativity mostly decreases the miss rate. However, both tracefiles reached a threshold in which the associativity started to slightly increase. Once more it is important to achieve the lowest miss rate possible, to improve the performance of the cache simulator. The XSBench had a higher miss rate than the MiniFE which follows previous findings.

**Section Three: Contrasting Replacement Policies and Associativity**

**Analysis:** For both tracefiles, LRU yielded the lower miss rate and would therefore be a better choice for a cache. This also makes perfect sense—the LRU updates a block on every single concurrent access. Whereas, the FIFO only updates a block on its very first iteration. So, the LRU is more up to date and will therefore keep memory that was more recently used (hence the name). However, the FIFO will evict the first block that enters (like a queue). Once more, the MiniFE had a lower miss rate than the XSBench. Also, the miss rate decreased very quickly as associativity initially increased.