EECE 7352: Computer Architecture

Assignment 4

Jiayun Xin

NUID: 001563582

College of Engineering

Northeastern University Boston, Massachusetts

Spring, 2022

1.

(A)

Table

Description automatically generated with medium confidence

Figure 1: 16 KB instruction cache, 32B block and direct mapped

Table

Description automatically generated with medium confidence

Figure 2: 16 KB data cache, 32B block and direct mapped

Table

Description automatically generated with medium confidence

Figure 3: 16 KB instruction cache, 64B block and direct mapped

Table

Description automatically generated

Figure 4: 16 KB data cache, 64B block and direct mapped

Table

Description automatically generated

Figure 5: 16 KB instruction cache, 128B block and direct mapped

Table

Description automatically generated

Figure 6: 16 KB data cache, 128B block and direct mapped

Table

Description automatically generated with medium confidence

Figure 7: 16 KB instruction cache, 32B block and 8-way associative mapped

Table

Description automatically generated

Figure 8: 16 KB data cache, 32B block and 8-way associative mapped

Table

Description automatically generated

Figure 9: 16 KB instruction cache, 64B block and 8-way associative mapped

Table

Description automatically generated with medium confidence

Figure 10: 16 KB data cache, 64B block and 8-way associative mapped

Table

Description automatically generated with low confidence

Figure 11: 16 KB instruction cache, 128B block and 8-way associative mapped

Table

Description automatically generated with medium confidence

Figure 12: 16 KB data cache, 128B block and 8-way associative mapped

|  |  |  |
| --- | --- | --- |
| Block Size (Byte) | Miss Rate (Directed) | Miss Rate (8-way) |
| 32 | 0.0475 | 0.0303 |
| 64 | 0.0333 | 0.0221 |
| 128 | 0.0252 | 0.0177 |

Table 1: summary of instruction caches with variable cache size and block size

|  |  |  |
| --- | --- | --- |
| Block Size (Byte) | Miss Rate (Directed) | Miss Rate (8-way) |
| 32 | 0.0698 | 0.0479 |
| 64 | 0.0691 | 0.0480 |
| 128 | 0.0708 | 0.0495 |

Table 2: summary of data caches with variable cache size and block size

Figure 13

Figure 14

The figure 13 and 14 show the trends of miss rate as block size increases for instruction caches and data caches respectively. The orange line represents direct mapped cache, and the blue line is 8-way associativity. As we can see, the miss rates of instruction caches decrease as block size increases because the larger block size have more instruction data. 8-way associative cache has smaller miss rates than direct mapped cache. For data cache, the miss rates keep stable as block size increases. The miss rate of 8-way associative cache is lower than direct mapped cache. Associative cache avoids much more both instructions and data being replaced. For those data with the same set index, they can be set into a new place in cache but replace the existing data.

(B)

1. Assume widths of address fields is 32 bits

#sets = 16KB / (2 \* 32) = 256 = 2^8

Blocksize = 16 = 2^4

Tag bits = 32 – 8 – 4 = 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address stream (hex) | Tag (hex) | Index (hex) | Block offset (hex) | comment |
| 0X00000 00 0 | 0X00000 | 0X00 | 0X0 | miss |
| 0X00001 00 0 | 00001 | 00 | 0 | miss |
| 0X00000 00 0 | 00000 | 00 | 0 | hit |
| 0X00000 00 0 | 00000 | 00 | 0 | hit |
| 0X00002 00 0 | 00001 | 00 | 0 | Miss/rep |

To touch every cache set exactly 5 times, three of them use totally same address and the rest two change the tag address. The last address will replace the existing data in cache by LRU policy. Do the same steps but only change index address for 256 times to ensure each set is touched exactly 5 times.

2. Assume widths of address fields is 32 bits and assoc is n

#sets = 16KB / (n \* 32)

Blocksize = 16 = 2^4

Tag bits = 32 – m – 4 = 20

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Address stream (hex) | Tag (hex) | Index (hex) | Block offset (hex) | comment |
| 0X00000 00 0 | 0X00000 | 0X00 | 0X0 | miss |
| 0X00001 00 0 | 00001 | 00 | 0 | miss |
| 0X00002 00 0 | 00002 | 00 | 0 | miss |
| 0X00003 00 0 | 00003 | 00 | 0 | miss |
| 0X00004 00 0 | 00004 | 00 | 0 | Miss/rep |
| 0X00001 00 1 | 00001 | 00 | 1 | miss |

Only change the tag part of address periodically. Use the same tag address but change block offset to see whether it hit. The period of changing tag address increases gradually until the address with used tag address misses. Through the period, the associativity of the cache can be calculated.

2.

Major findings:

1. Using causal profiling enable effectively improve performance tuning by 68% through observing the results of case studies about 2 real applications and six PARSEC benchmarks.
2. Based on the case studies - ferret and dedup, causal profiling (Coz’s predictions) is highly accurate.
3. Coz’s overhead is low enough to be used in practice depending on the result that Coz’s profiling overhead is 17.6% on average.

Performance improvement is an important index to assess code optimization. As a software casual profiler, Coz not only improves software performance, but also has high accurate. It means a lot for software developers and improves software running efficiency. It is common that a computer has multiple cores nowadays. The improvement of profiling can same much running time. Several case studies and benchmarks confirm the truth of findings.

3.

Victim caching is designed as loading a small full-associative cache which catch those entries replaced from cache level 1 to decrease conflict misses. When a data hits in the victim cache but direct-mapped cache, the cache line matched in the victim cache and a cache line in the direct-mapped cache swapped.

Victim cache improves the percentage of conflict misses removed, especially for the benchmarks have conflicting long sequential reference streams.

A direct-mapped cache with a 2-entry victim cache has a better performance than a 2-way set associative cache.

Victim cache has a better performance than miss cache in terms of the overall reduction in miss rate.

4.

a) The dynamic insertion policy is designed to reduce cache misses by choosing between LRU Insertion Policy and Bimodal Insertion Policy depending on which incurs fewer misses.

Through using the dynamic insertion policy, the advantages and disadvantages of LIP and BIP can be balanced. LIP has good performance for high-locality workloads but it is memory-intensive and not adapt to changes in working set. BIP responses to the changes in working set and keeps the thrashing protection of LIP.

b) Set dueling is a mechanism that arranges some cache sets to the LIP and BIP and chooses the policy performing better on the cache sets arranged.

Set dueling is used to ensure that DIP implements without significant hardware overhead.

5.

The key idea discussed in this paper is FLEXclusion, which can dynamically choose between exclusive and non-inclusive caches based on workload behavior.

Benefits: FLEXclusion effectively reduces LLC insertion traffic, power consumed and improve performance.

Downsides: The two options for FLEXclusion have similar coherence framework, inclusion and other modes can be considered in the future work.