Cache memory simulator

Simulation results analysis

Aleksa Majkić

August 2021

Contents

Chapter 1

Results analysis

1.1 *Directly mapped* vs. *N-way associative* cache

For these simulations I have used the following parameters:

Cache type	L1-D only
Cache memory size	32,768 B
Cache line size	32 B
Write-hit policy	write-back
Write-back policy	write allocate
Type of CPU	single core
Replacement algorithm	LRU
Trace file size	medium (1,000 commands)
RAM size	1 GB

Table 1.1: Simulation parameters

Set-associative cache is a trade-off between direct-mapped cache and fully associative cache. Cache is divided in to n sets where every set contains m cache lines meaning that it can be represented as a $n \times m$ matrix. Based on the choice of the associative level a slower cache could be created due to the need for more searches while on the other hand you could get a cache memory that has fewer misses which will result in CPU not wasting time reading from a memory which is slower compared to the cache memory. For example, if we have 10 location on which a memory location has been mapped, then we would have to search all 10 locations for our data, meaning that this approach will probably require more time. A base rule for cache would be: doubling in associativity, from directly mapped to two-way or from two-way to four-way associative cache, has approx. the same effect on hit/miss ration as doubling the size of the cache. I would be remiss if I didn't mention that increasing the associativity more than four-way associative cache wouldn't yield a much

better hit/miss ratio, which is why it's rarely done if ever. Directly mapped cache has a good best-case time, but it is unpredictable in the worst-case scenario. Fully associative cache makes sense using for a cache with a small amount of cache lines.

Number of accesses	970
Number of hits	812
Hit ratio	0.837
Number of misses	158
Miss ratio	0.163
Number of cache evictions	64
Number of memory writes	38
Number of memory reads	79

Table 1.2: *Directly mapped* cache simulation results

For the next simulation I've used a *two-way* cache, which yield in the hit count increased by 27 (3.22%), meaning that the number of misses, cache evictions and number of read/write operations to and from RAM has decreased.

Number of accesses	970
Number of hits	839
Hit ratio	0.865
Number of misses	131
Miss ratio	0.135
Number of cache evictions	35
Number of memory writes	21
Number of memory reads	64

Table 1.3: *Two-way* cache simulation results

For a *four-way* cache number of hits increased by 3.56% in comparison to the *two-way* cache.

Number of accesses	970
Number of hits	870
Hit ratio	0.897
Number of misses	100
Miss ratio	0.103
Number of cache evictions	0
Number of memory writes	0
Number of memory reads	46

Table 1.4: *Four-way* cache simulation results

1.2 *Least-recently* vs. *Belady* algorithm

For these simulations I have used the following parameters:

Cache type	L1-D only
Cache memory size	16,438 B
Cache line size	32 B
Cache associativity	2-way
Write-hit policy	write-back
Write-back policy	write allocate
Type of CPU	single core
Replacement algorithm	LRU
Trace file size	medium (1,000 commands)
RAM size	1 GB

Table 1.5: Simulation parameters

Second simulation or the simulation done with a *Bélády* algorithm yielded better results compared to the simulation using LRU algorithm. The results showed clearly a better hit/miss ration, which makes sense because this is the most efficient caching algorithm that can theoretically produce an optimal cache placement based on the beforehand known instructions. The results can be seen in the tables below.

Number of accesses	970
Number of hits	829
Hit ratio	0.855
Number of misses	141
Miss ratio	0.145
Number of cache evictions	46
Number of memory writes	30
Number of memory reads	67

Table 1.6: LRU results

Number of accesses	970
Number of hits	842
Hit ratio	0.868
Number of misses	128
Miss ratio	0.132
Number of cache evictions	33
Number of memory writes	23
Number of memory reads	61

Table 1.7: Belady results