Lab3 Design Document

Mei Yixuan 2019011041 Yao92

May 15, 2021

1 Advanced Cache Replacement Policies

1.1 LRU-LIP

In LRU-LIP, we set counter of the newly inserted block as the number of valid ways in the corresponding set minus one. This ensures that newly added blocks are in the least important position. Also, the counters are continuous (i.e. if we have 3 valids ways, their counters have value 0, 1 and 2). This nice property makes eviction and reversion much easier: we can simply use the same function as in LRU. The hardware control overhead of LRU-LIP is one counter each way, which is identical to LRU. Exact hardware cost is in the following figure.

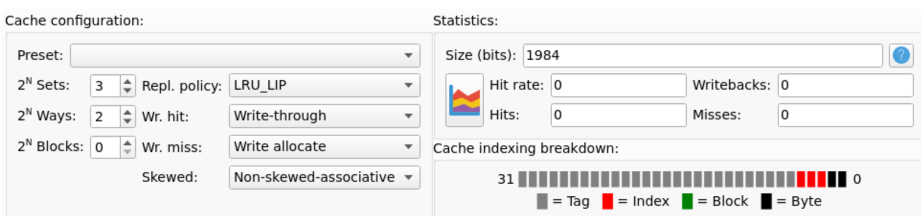

Figure 1: LRU-LIP hardware cost

1.2 DIP

In DIP, we use the first set (SET0) as MIP sample and the second set (SET1) as LIP sample. In each memory access on non-dueling sets (i.e. sets other than SET0 and SET1), we update cache control fields according to current better replacement policy. We reset all counters every 100000 memory accesses to avoid potential risk of overflow. Thanks to the good property of LRU-LIP, eviction and reversion of DIP is also identical to LRU. Besides the counter in each way, DIP also needs five counters for data recording. Exact hardware cost is in the following figure.

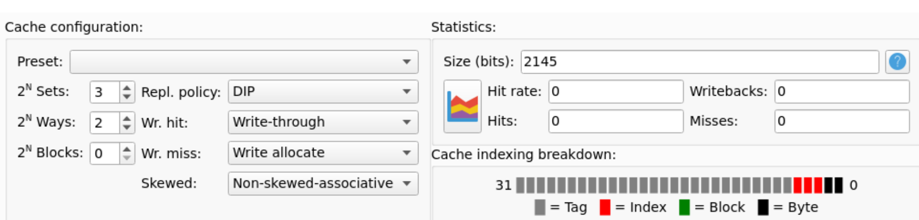

Figure 2: DIP hardware cost

1.3 RRIP

In RRIP, we use counter field of each way to store its RRI. It has field width of 3 bits. Upon hit, we set RRI of corresponding entry as 0. Upon insertion, we set RRI of corresponding entry as long RRI (i.e. 6). When choosing a block for eviction when all ways are occupied, we choose the block with largest RRI and normalize all values to distant RRI (this is identical to adding one repeatedly). Exact hardware cost is in the following figure.

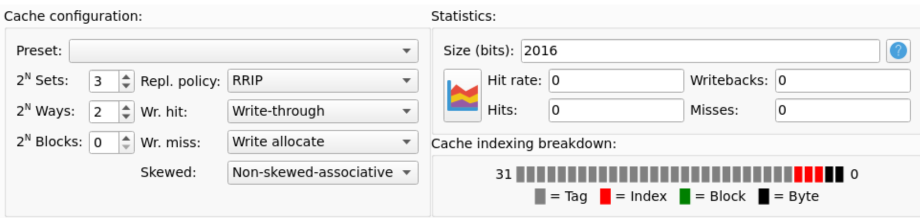

Figure 3: RRIP hardware cost

1.4 Results

Results of different replacement policies:

Design	Benchmark 1 Miss Rate	Benchmark 2 Miss Rate	Benchmark 1 Total Cycles	Benchmark 2 Total Cycles
No Cache	100	100	2083191	23508468
Random Replacement	16.47	2.44	1230758	9275269
LRU Replacement	13.86	1.99	1204202	9209763
LRU-LIP Replacement	14.77	2.76	1213470	9322490
DIP-Replacement	14.66	2.27	1212268	9250427
RRIP-Replacement	13.71	2.24	1202611	9246016

Results of different cache capacities and mapping schemes:

Capacity and Mapping	Benchmark 1 Miss Rate	Benchmark 2 Miss Rate	Benchmark 1 Total Cycles	Benchmark 2 Total Cycles
16-entry direct-mapped				
16-entry 4-way associative				
16-entry 8-way associative				
16-entry fully-associative				
32-entry direct-mapped				
32-entry 4-way associative				
32-entry 8-way associative				
32-entry fully-associative				
64-entry direct-mapped				
64-entry 4-way associative				
64-entry 8-way associative				
64-entry fully-associative				

2 Skewed-Associative Cache

2.1 Implementation

In skewed-associative cache, we need separate hash functions for each way. Since we want blocks to be independently and uniformly mapped in each way, we choose a group of universal hash functions for mapping. Specifically, we choose $h_{ab}(k) = (a * k + b) \mod m$, in which k is the block address, m is the number of sets. In order for this group of hash functions to be universal, we need to choose a and b uniformly random for each way. Considering actual performance, we choose $a = 1$ for all ways, and b be a random number for each way. Note that the tag function used for non-skewed cache is not suitable for skewed cache, since multiple blocks with the same tag may be mapped into the same way of the same set. Therefore, we use the whole address field as tag instead.

2.2 Results

3 Benchmarks

3.1 Replacement Benchmarks

3.2 Writehit Benchmarks

3.3 Writemiss Benchmarks

4 Ripes Bug Report

5 Question Answering