Victim cache implementation and performance analysis using gem5

Madhav Rathi
mrathi3@wisc.edu

Jason Zhou
zzhou292@wisc.edu

I. Topic Description

We focus on victim cache as an optimization to reduce cache latency. We propose to implement victim cache at multiple levels of cache hierarchy to analyze it's performance impact. Victim cache will be implemented for both simple in-order TimingSimpleCPU and MinorCPU, before moving on to complex out-of-order O3CPU. Validation will be done on in-order CPUs through simple microbenchmarks. We compare the performance impact of adding victim cache at different levels of cache hierarchy, and analyze it's performance compared to CPUs without victim cache. Victim cache inclusion in simulation will be configurable, similar to cache inclusion in gem5.

II. Topic Relevance

Jouppi in [1] showed that victim cache can eliminate conflict misses in L1 D-cache as much as 40-100%. Victim cache has a small access latency due to being a small cache, thereby improving performance compared to lower level cache accesses. Gem5 being the state-of-art architecture simulation tool does not implement a victim cache, thereby missing key micro-architectural performance modeling information. Adding a victim cache to gem5 will help it mimic the micro-architecture further.

III. Evaluation Methods

We list the performance metrics (stats) compared and the benchmarks to be run for analysis.
Performance metrics compared:

1) Percentage of misses due to conflicts.
2) Percentage of conflict misses removed by victim cache.
3) Percentage of all misses removed due to victim cache.
4) Number of L2 accesses with and without victim cache at L1.
5) Number of DRAM accesses with and without victim cache at L2.

Benchmarks to be run for performance analysis:

1) MC, a microbenchmark which exhibits many conflict misses [5].
2) MCS, a microbenchmark which exhibits many conflict misses with stores [5].
3) Spec2006 {464.h264ref, 401.bzip2} [6].

IV. Project Plan

The project is partitioned into following tasks:

1) Implementation of victim cache in TimingSimpleCPU and MinorCPU: The implementation will extend the BaseCache class and create a fully associative victim cache. First implemented in TimingSimpleCPU and MinorCPU. Victim cache implemented in data path, between L1 D-Cache & L2, L2 & L3.
November 15, 2022
2) Validation of victim cache in TimingSimpleCPU and MinorCPU: Validation will be performed on TimingSimpleCPU and MinorCPU with victim cache implemented between L1 D-Cache & L2. Once, the correctness of victim cache is established, we can move on to performance analysis. If any incorrectness has been discovered, the previous step will need to be modified and replayed.
November 20, 2022
3) Performance analysis on TimingSimpleCPU and MinorCPU: A benchmark comparison will be performed with victim cache implemented between (1) L1 D-Cache & L2; (2) L2 & DRAM; (3) Both L1 D-Cache & L2, L2 & DRAM. Performance analysis will be performed using several different benchmarks mimicking the performance analysis conducted in [1]
November 25, 2022
4) Implementation of victim cache in O3CPU: Once analysis completed for TimingSimpleCPU and MinorCPU, we implement victim cache in O3 CPU. This brings additional complexity since call to the cache is from LSQ, as well as victim cache must be non-blocking. Here, victim cache is implemented only between L1 D-Cache & L2.
December 5, 2022
5) Validation of victim cache in O3CPU: Validation will be performed on O3CPU with victim cache implemented between L1 D-Cache & L2. Once, the correctness of victim cache is established, we can move on to performance analysis. If any incorrectness has been discovered, the previous step will need to be modified and replayed.
December 10, 2022
6) Performance analysis on O3CPU: A benchmark comparison will be performed with victim cache implemented between L1 D-Cache & L2. Performance analysis will be performed using several different benchmarks mimicking the performance analysis conducted in [1].

December 17, 2022

V. Implementation details

Table 1 lists tentative microarchitecture details of victim cache architecture. Table 2 lists tentative microarchitecture details of base L1 D-Cache and L2 cache. We simulate our design only on single core architectures, which eliminates coherence checks of the victim cache.

TABLE I
VICTIM CACHE MICROARCHITECTURE[a]

Victim Cache	Microarchitecture Details			
	Size	***Associativity***	***Replacement***	***Line Size***
L1	4KB	Fully-associative	LRU	64B
L2	16KB	Fully-associative	LRU	64B

[a]Subject to change.

TABLE II
BASE CACHE MICROARCHITECTURE[b]

Base Cache	Microarchitecture Details			
	Size	***Associativity***	***Replacement***	***Line Size***
L1	64KB	2-way set associative	LRU	64B
L2	512KB	4-way set associative	LRU	64B

[b]Subject to change.

VI. Reference Review and Description

Jouppi in [1] proposed the concept of victim cache and provided benchmark results showing that victim cache can significantly improve the cache performance. This paper tries to emulate this victim cache design in gem5. We present some basic conceptual implementation details which serve as a baseline for implementation and validation on TimingSimpleCPU and MinorCPU.

References [2], [3], and [4] optimize victim cache performance. Both [2] and [4] propose a modified selection and prediction policy identifying the content filled in the victim cache. In [2], Kim et al. propose an adaptive block management scheme for victim cache, wherein blocks filled in the victim cache also consider L1 cache history information. In [4], victim cache is also used as a miss cache, wherein an incoming block fill to L1 cache is selectively placed in L1 or victim cache based on the history of that block. We do not consider prediction or history information while filling the victim cache, as similar to [1], every eviction from the cache is filled in to the victim cache. [3] implements and benchmarks a combination of prefetching and victim caching, although this might not be directly related to our topic, it sheds lights on some benchmark results by implementing victim cache only. Our scope for this paper is only on victim caching, and we do not consider prefetch impact.

[5] provides many microbenchmark algorithms in five categories - Control, Data Parallel, Execution, Memory, and Store Intense - which we can take advantage of to understand how the implemented victim cache performs for various applications. For example, MIM exhibits stream accesses, MC which generates a lot of conflict misses, while CCa is not memory heavy. [6] provides CPU benchmark standards released on August 24, 2006 by Standard Performance Evaluation Corporation (SPEC), which we can utilize as a reference during our performance analysis.

[7] will serve as a general guide as we are navigating through the gem5 simulator.

References

[1] Norman P. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers” ACM SIGARCH Computer Architecture News Volume 18, Issue 2SI, June 1990.

[2] Cheol Hong Kim, Jong Wook Kwak, Seong Tae Jhang, Chu Shik Jhon, “Adaptive Block Management for Victim Cache by Exploiting L1 Cache History Information” International Conference on Embedded and Ubiquitous Computing, EUC 2004: Embedded and Ubiquitous Computing pp 1-11, 2004.

[3] Walter Schilling, Mansoor Alam , “The Impact of Prefetching and Victim Caching on Computer Systems Performance.” Conference: Proceedings of the ISCA 15th International Conference Computers and Their Applications, March 29-31, 2000, New Orleans, Louisiana, USA, January 2000.

[4] D. Stiliadis and A. Varma, ”Selective victim caching: a method to improve the performance of direct-mapped caches,” in IEEE Transactions on Computers, vol. 46, no. 5, pp. 603-610, May 1997.

[5] UC Davis Computer Architecture Research Group (July 12, 2019). Extremely Simple Microbenchmarks [Github].

[6] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. SIGARCH Comput. Archit. News 34, 4 (September 2006), 1–17.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. SIGARCH Comput. Archit. News 39, 2 (May 2011), 1–7.