Optimizing Cache Performance: Evaluating SRRIP Against Traditional Replacement Policies

Vaishnav Ganapathiraju
Electrical & Computer Engineering
Texas A&M University

***Abstract*—Cache replacement policies typically work by predicting the likelihood of a block being re-referred. The LRU policy works under the assumption that the most recent accesses will be most likely to be re-referred sooner, and as a result can struggle with workloads that have non-temporal or scan-heavy data. LFU keeps track of how many times each block is referenced and replaces the block with the lowest re-reference frequency, but it takes a hit in workloads with a high degree of temporal locality. SRRIP contrasts LRU by assigning a long re-reference interval on a cache miss, and setting the block to a near-immediate re-reference on the next hit, which enables it to resist scans and improve cache retention for frequently accessed data. Using an architectural simulator, I evaluated SRRIP's performance relative to LRU & LFU in terms of cycle counts, instructions per cycle (IPC), and miss rate per thousand instructions (MPKI).**

I. Introduction

LRU predicts that the most recently accessed blocks will be reused soon, favoring workloads with high temporal locality. However, for applications with distant re-referneces, LRU's near-immediate re-reference assumption results in bad performance. LFU is scan-resistant but doesn't work well for workloads with temporal locality. The RRIP approach predicts a block's re-reference interval using an m-bit register per block to store a prediction, aiming to approximate an optimal policy by focusing on both scan and thrash resistance. The paper [1] proposes two variants of RRIP, Static RRIP, which is scan-resistant, and Dynamic RRIP (DRRIP), which adds thrash resistance by adjusting its predictions dynamically. SRRIP has the benefit of using minimal hardware overhead (2 bits per block) and being easy to implement in an LRU-based system, providing better utilization for mixed access patterns.

II. SRRIP Technique

A. Short description of SRRIP technique

Static Re-reference Interval Prediction (SRRIP) provides robustness for different access patterns by inserting new blocks with a long re-reference prediction, instead of a near-immediate or distant re-reference prediction. Not using a near-immediate re-reference for new blocks prioritizes retention of frequently used data over rarely re-accessed blocks. This prevents scan-type data from displacing critical working-set data, improving overall cache performance.

B. Implementation details

SRRIP uses an M-bit register per block, which stores a Re-reference Prediction Value (RRPV). Upon insertion, a block is assigned a RRPV = $2^M - 2$, indicating that it will not be immediately reused. On a hit, SRRIP adjusts the RRPV to reflect a likely near-immediate reuse by setting it to zero. Victim selected for replacement is the leftmost block with an RRPV value equal to $2^M - 1$, ensuring that blocks predicted to be reused in the distant future are evicted first. In case no block has the maximum RRPV, the prediction values for all blocks are incremented by 1 and checking again, continuing until a victim emerges. In this report, SRRIP was implemented using M = 2 and RRPVMax = 3.

III. Methodology

Zsim [2], an x86_64 simulator was used to simulate 12 programs from the SPECCPU2006 suite - *bzip2, gcc, mcf, hmmer, sjeng, libquantum, xalan, cactusADM, namd, soplex, calculix, lbm* and 8 programs form the PARSEC [4] suite - *blackscholes, bodytrack, canneal, dedup, fluidanimate, streamcluster, swaptions, x264* using the *simlarge* input sets. The system configurations used for the two suites are listed in Table I.

	SPECCPU2006	PARSEC
Core(s)	1 OOO Westmere	8 OOO Westmere
L3 size	2MB	8MB
L3 associativity	16-way	16-way
Number of banks in L3	-	8

TABLE I: System configurations

IV. Evaluation

Total cycles (Figure 1), IPC (Figure 2), and Misses per Kilo Instructions (MPKI) of the L3 cache (Figure 3), were used to compare the performance of each of these policies. The figures are split into SPEC integer benchmarks, PARSEC floating point benchmarks and PARSEC benchmarks.

SRRIP consistently outperformed LRU and LFU across most benchmarks, although with close margins. However, a notable exception was the *streamcluster* workload, where SRRIP displayed substantial improvement. Table II and Table III show the average improvement across all other workloads and just *streamcluster* separately. The streamcluster workload's unique behavior is maybe due to a scan-heavy access pattern, which SRRIP would work well for.

	LRU	LFU
Total Cycles	0.88%	0.17%
IPC	0.83%	0.18%
MPKIy	4.43%	0.9%

TABLE II: SRRIP improvement over LRU & LFU, excluding *streamcluster*

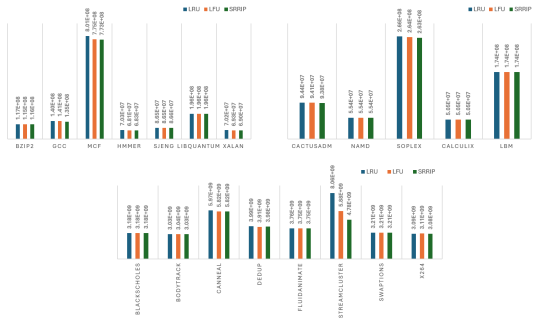

Fig. 1: Total Cycles

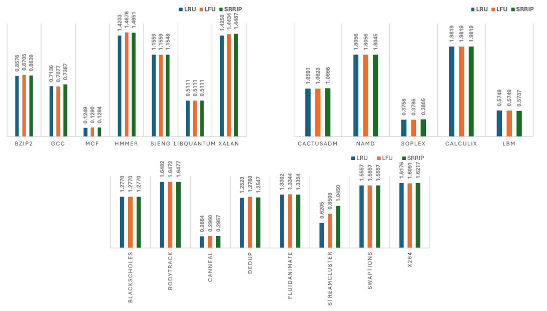

Fig. 2: Instructions per Clock (IPC)

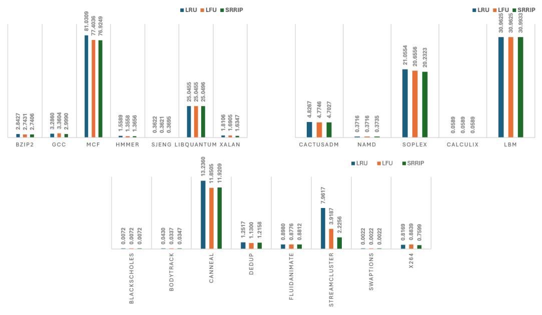

Fig. 3: Misses per Kilo Instructions (MPKI)

	LRU	LFU
Total Cycles	40.62%	18.58%
IPC	68.43%	22.81%
MPKIy	72.05%	43.20%

TABLE III: SRRIP improvement over LRU & LFU for *streamcluster*

Acknowledgment

I would like to thank the Computer Science Department for providing access to computing resources to run these simulations.

Conclusions

SRRIP yielded a very modest improvement over LRU and LFU in most workloads, while demonstrating significant performance gains in the *streamcluster* workload.

References

[1] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). In Proceedings of the 37th annual international symposium on Computer architecture (ISCA '10). Association for Computing Machinery, New York, NY, USA, 60–71. https://doi.org/10.1145/1815961.1815971

[2] ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems", Sanchez and Kozyrakis, ISCA-40, June 2013

[3] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. SIGARCH Comput. Archit. News 34, 4 (September 2006), 1–17. https://doi.org/10.1145/1186736.1186737

[4] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: characterization and architectural implications. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08). Association for Computing Machinery, New York, NY, USA, 72–81. https://doi.org/10.1145/1454115.1454128