Implementation and Evaluation of SRRIP and SF-LRU Cache Replacement Policies

Team 9: Manoj Bontalakoti, Balasainath Yelampalle, Nikhilesh Devarasetty, Bhavana Marpadaga, Gnanesh Naknamoni

ABSTRACT

In this project, we implemented two cache replacement policies namely, Static Re-Reference Interval Prediction (SRRIP) and Second Chance-Frequency - Least Recently Used (SF-LRU). There are certain applications where the working set is larger than the cache or there might be frequent references to the temporal data, such applications usually underperform under LRU replacement policy. The SF-LRU and SRRIP not only improves the performance but also reduces the number of cache when compared to LRU replacement policy. The SRRIP policy easily integrates with LRU by using 2 bits per cache block. Studies show that the SRRIP policies outperform the LRU on an average of 4%. On the other hand, the SF-LRU policy compares the frequency of the block with the block next to it in the set and provides a second chance to the block that might be deleted in the LRU policy. We also showed the comparison among the three replacement policies: LRU, SF-LRU and SRRIP with respect to Instruction per count and various benchmarks, associativity, block sizes and sets.

1. INTRODUCTION

Memory hierarchy is one of the most important elements for a high-performance processor or system and fetching data from these would require more power. Among all the memories, cache plays a critical role in improving the performance of a high-performance microprocessor. These caches prevent in reducing the execution time and the power consumption by providing the access to the frequently referenced data by the processor. Cache memory is the simplest and cost-effective way to improve the performance of the processor. There are three types of cache organization: direct mapped, set associative and fully associative. Generally, a set associative cache has a good balance between hit and miss ratios but along with this the cache replacement policy also plays a crucial role. The most common function of a replacement policy is to reduce the miss rate and the most popular among all is the least frequently used (LRU) but using this algorithm as reference there are many other sophisticated replacement policies proposed like SFLRU and SRRIP. The relative performance of these policies primarily depends on the history reviewed. It is crucial to observe the common access patterns of the cache block to make a replacement decision.

In RRIP the replacement of the block is done by reviewing the access patterns and then it will replace the block that will be referenced in the late future. The RRIP chain represents the order in which the blocks are predicted to be re-referenced; the block at the head of the RRIP chain is predicted to have a near immediate re-reference interval while the block at the tail of the RRIP chain is predicted to have a distant re-reference interval. This implies that the cache block will be re-referenced in the distant future. RRIP performs well when the re-references only occur in the distant future unlike the LRU which performs well only when the workloads have high data locality. So, when a burst references a non-temporal data or when the working set is larger than the cache LRU replacement policy underperforms using the cache inefficiently. The implementation of RRIP is done by using a M-bit register per cache block to store the re-reference prediction value (RRPV).

The rest of the report is organized as follows, Section 2 dwells into the algorithm and implementation details of the Static Re-Reference Interval Prediction (SRRIP) and section 3 on the details of Second

Chance-Frequency - Least Recently Used (SFLRU) policy. Section 4 presents the results and analysis of the three policies with respect to IPC on various parameters like associativity, block side and sets. Finally, Section 5 summarizes the report and our work.

2. STATIC RE-REFERENCE INTERVAL PREDICTION REPLACEMENT POLICY:

As the references made by RRIP are statically determined on cache hits and misses, this replacement policy got “static” in its name. This replacement policy is the modification to the existing LRU (least recently used policy). When the cache access patterns are mixed (i.e., few of the references have near-immediate re-reference interval while others have distant re-reference interval), LRU replacement policy cannot differentiate between the blocks with near-immediate re-reference interval from blocks with distant re-reference interval. When there are frequent references to non-temporal data (scans), they do not get hits after their initial reference and when the data referenced after the scans belongs to the working set prior to the scan, the distant re-reference interval replacement policy has to be applied to the cache blocks belonging to the scans and near-immediate re-reference interval to the working sets.

RRIP prevents cache blocks with a distant re-reference interval from evicting the blocks that have a near-immediate re-reference interval. Static RRIP is only scan resistant. It requires M-bits per cache block to store 2^M re-reference prediction values (RRPVs). RRIP learns re-reference information on-flight as the cache access pattern passes by. RRPV value of zero means that cache block is predicted to be re-referenced in the near-immediate future while 2^M-1 value means the block is predicted to be re-referenced in the farthest future. RRIP always inserts the new blocks with a long re-reference interval (2^M-2). Doing this, not only prevents the cache blocks with distant re-references from polluting the cache but also it gives adequate time for the predictor to learn the re-reference prediction of the cache block.
If a new block has a near-immediate re-reference interval than the previous block, then the new block’s RRPV is set to be shorter than the previous block.

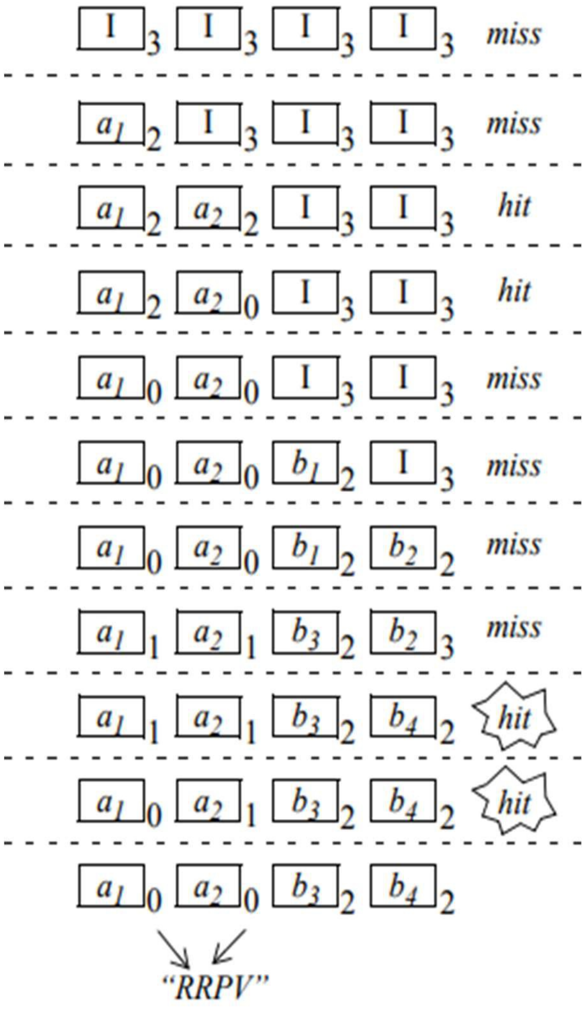


Fig 2.1: The above figure represents the behavior of 2-Bit SRRIP with Hit promotion update policy.

2.1 Algorithm for the above SRRIP behavior:

```
if (Cache Hit):
    i) Set RRPV of the block to '0'
else if (Cache Miss):
    i) Search for the first '3' from left to right.
    ii) If '3' found, go to step (v)
    iii) Increment all RRPVs
    iv) Go to step (i)
    v) Replace block and set RRPV to '2'
```

The predictor updates the predictions by incrementing the RRPV's if a distant re-reference interval is not found. When there is a hit to the block the predictor has a chance to change the re-reference prediction of a block. This is called hit promotion policy. There are two policies to update the re-reference prediction, Hit Priority (HP) and Frequency Priority (FP). HP policy assigns a block with near-immediate future RRPV on a hit. This policy will degrade the cache performance when a cache block is re-referenced only once after insertion. This will unnecessarily accommodate a cache block which has no immediate re-reference. In such situations, using the frequency metric of the cache blocks, the policy updates the re-reference prediction. Instead of updating the re-reference prediction to be near-immediate on a hit, it updates the re-reference interval to be shorter than the previous reference interval each time a block receives a hit. Thus, SRRIP is efficient for the mixed cache access patterns.

3. SECOND CHANCE FREQUENCY-LRU REPLACEMENT POLICY (SF-LRU)

This replacement policy is a combination of both LRU and LFU. It uses frequency and recency to evict the blocks and insert the blocks. The policy gives the second chance for the block to present in the cache which gets evicted in the LRU policy. The basic structure is to modify the LRU with a counter for estimating the frequency for each block and giving the second chance for the block with the highest counter value. This combined number is called recency frequency control value (RFCV).

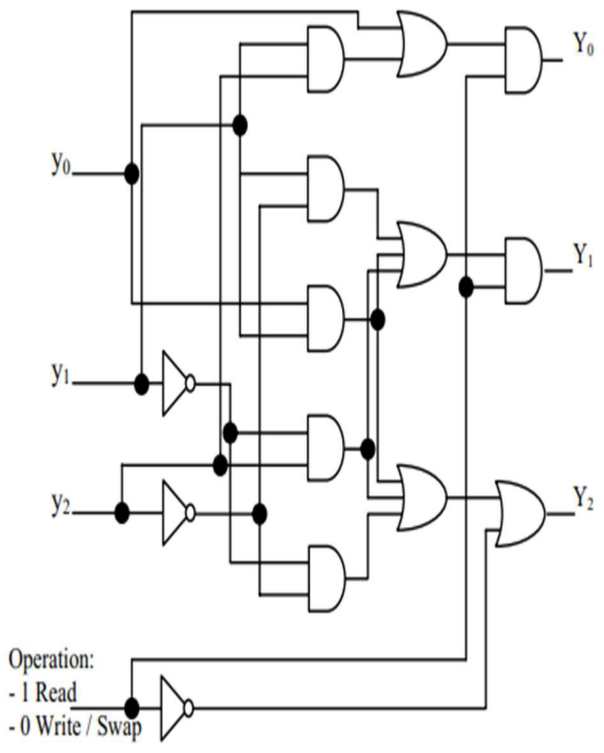


Fig 3.1: The figure shows Hardware representation of the RFCV. Y= (y+1)^operation

3.1 Algorithm for the SF-LRU policy:

```
if(Cache Hit)
    Execute LRU;
    Update RFCV;
else
    Call Miss Handling Procedure
    Compare RFCV's for last two blocks
        if(last >previous)
            Update RFCV of last;
            Swap two blocks (previous and last)
            Execute LRU;
        else
            Execute LRU
```

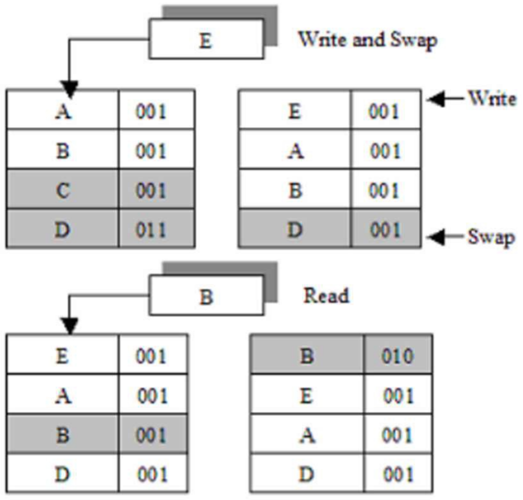
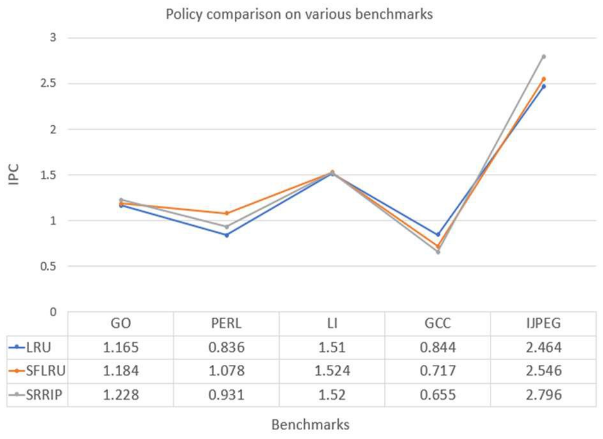


Fig 3.2: Example demonstrating SF-LRU replacement policy.

For the write case in the above figure, the RFCV values are compared for the last two blocks. As the last block's value is greater than the previous block's value, SF-LRU gives a second chance for the last block and evicts the previous block. For the read case, if it is a hit then RFCV values are updated and LRU is executed.

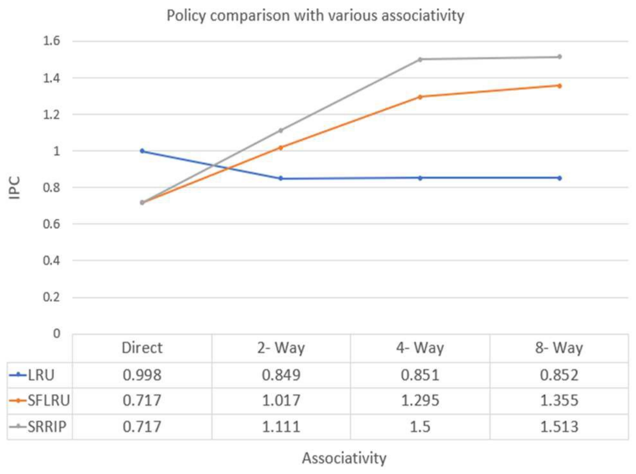
4. RESULTS AND ANALYSIS

The following are the comparisons among three replacement policies: LRU, SF-LRU, SRRIP.



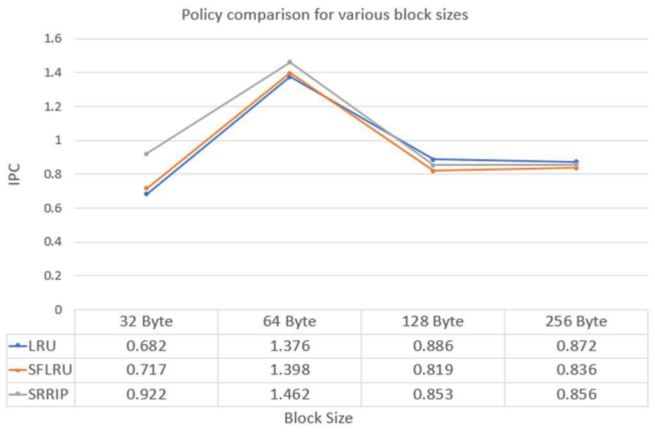
	GO	PERL	LI	GCC	IJPEG
LRU	1.165	0.836	1.51	0.844	2.464
SFLRU	1.184	1.078	1.524	0.717	2.546
SRRIP	1.228	0.931	1.52	0.655	2.796

Fig 4.1: Comparing LRU, SF-LRU and SRRIP w.r.t IPC and various benchmarks



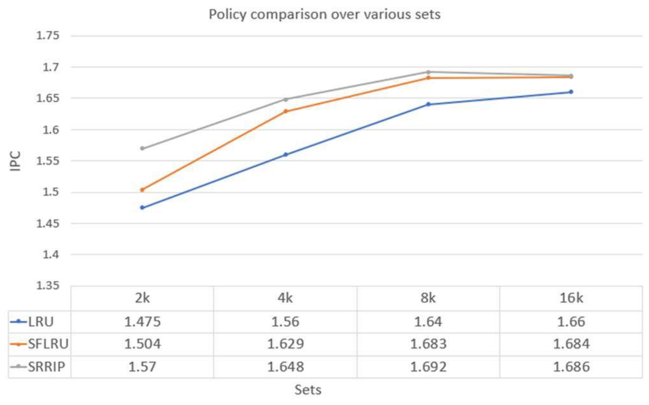
	Direct	2- Way	4- Way	8- Way
LRU	0.998	0.849	0.851	0.852
SFLRU	0.717	1.017	1.295	1.355
SRRIP	0.717	1.111	1.5	1.513

Fig 4.2: Comparing LRU, SF-LRU and SRRIP w.r.t IPC and various associativity



	32 Byte	64 Byte	128 Byte	256 Byte
LRU	0.682	1.376	0.886	0.872
SFLRU	0.717	1.398	0.819	0.836
SRRIP	0.922	1.462	0.853	0.856

Fig 4.3: Comparing LRU, SF-LRU and SRRIP w.r.t IPC and various block sizes



	2k	4k	8k	16k
LRU	1.475	1.56	1.64	1.66
SFLRU	1.504	1.629	1.683	1.684
SRRIP	1.57	1.648	1.692	1.686

Fig 4.4: Comparing LRU, SF-LRU and SRRIP w.r.t IPC and various sets

To evaluate the performances of SRRIP and SF-LRU replacement policies and to compare them with LRU performance metrics, we have considered IPC and number of cache misses (miss rate) as the metrics to analyze them. We ran on the simple super scalar simulator. The first fifty million instructions are skipped and then evaluation starts taking 10 million instructions at a time. The values are obtained for different SPEC95 benchmarks like GCC, GO, PERL, LI, IJPEG and various cache metrics like different associativity, cache size and the number of sets. From each of the graphs above, we can come to various conclusions such as,

i) SRRIP has a better IPC rate than SF-LRU and LRU in general. (SRRIP > SF-LRU > LRU)

ii) There is no uniform rise or fall trend for any of the policies with the varying associativity, block sizes and number of sets.

iii) As the associativity increases, the number of misses decreases (miss rate goes low), so the replacement policy has little effect at higher associativity. Hence the IPC and miss rate are constant for all the policies at 8 way set associativity.

iv) As the block size increases, the effect of better replacement policy first increases then it goes down and stays constant at larger cache line size (256 Byte). This is primarily due to the reason that as the cache block size is large, the active working set fits entirely at once in the block, which means that LRU, SRRIP and SF-LRU yield the same results.

v) The number of cache misses for SRRIP are less when compared to SF-LRU for all the access patterns.

vi) On a comparison with LRU policy, SRRIP yields 5.7 % and SFLRU yields 4.3 % improvement in handling the misses.

5. CONCLUSION

The conventional LRU policy works best for the non- temporal data references that are near immediate re-reference intervals. But the SRRIP policy predicts the re-referenced intervals of all the missing cache blocks that are in between the near immediate re-reference interval and a distant re-reference interval. And the implementation of this policy only requires 2 bits per cache block. On the other hand, the SFLRU policy not only decreases the miss rate, but also reduces the power consumption. This policy compares the frequency of the block with the block next to it in the set, and then provides a second chance to the block that might be deleted according to the LRU. Comparisons among all the three policies have been discussed in the report. Finally, we can conclude that not a single policy can handle any type of access pattern, but the policies need to be chosen based on the access pattern. One good policy for a particular pattern might not be efficient for the other pattern.

6. REFERENCES

[1] Handy, J., The Cache Memory Book, Second Edition, Academic Press, 1998.

[2] R. Subramanian, Y. Smaragdakis, G. Loh. “Adaptive caches: Effective shaping of cache behavior to workloads.” In MICRO-39, 2006

[3] H.Al-Zoubi, A. Milenkovic, M. Milenkovic. “Performance evaluation of cache replacement policies for the SPEC CPU2000 benchmark suite.” In ACMSE, 2004

[4] Patterson, D.A. and Hennessy, J.L., Computer Organization & Design, The Hardware/Software Interface, Second Edition, Morgan Kaufmann Publishers, 1998.

[5] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., Joel Emer. “High Performance Cache Replacement Using Re-Reference Interval Prediction (RRIP)

[6] Jaafar Alghazo, Adil Akaaboune, Nazeih Botros. “SF-LRU Cache Replacement Algorithm”.

[7] Jamil, T, Stacpoole, RA, “Cache Memories”, IEEE Potentials, vol. 19, no.2, pp. 24 - 29 (2000).