High Performance Cache Replacement Using Static Re-Reference Interval Prediction (SRRIP)

**Abstract:**

The cache system of a computer can have multiple levels that use different forms of memory management. There is a constant need to improve the memory access and cache management of computers to increase performance and speed. The only way for that to happen is to decrease the miss rate of the cache access. There are multiple ways of doing this though the way we plan to implement is through determining which blocks of memory needs to be replaced. What we use is to test this simple scalar which is a simulator that helps determine efficiency of a cache system. Though what simplescalar implements is LRU, FIFO, and random memory access. Those three implementation will work but they aren’t truly effective. Based on a research paper which presents a SRRIP which uses bit values and through an algorithm it determines which least recently used cache block that based on bit mapping values will determine the longest expected block and replaces it. We plan to implement this in simple scalar and executing it on the Bzip2 benchmark. From there we are going to compare values and determine if the SRRIP will decrease miss rate and is similar to the data to the research paper proving similar implementation.

**Introduction:**

An optimal replacement policy is important for performance of a cache system. The replacement policy is to determine which block in the cache set that needs to be replaced when there is a miss. The basic replacement policies presented are least recently used (LRU), first in first out (FIFO), and random. The actions that random takes for cache miss is that it chooses a random block to replace in the cache set. If there is a hit Random does nothing. While for FIFO and LRU the miss actions are the same by choosing the last block of the set replacing it with the new missed block and moving it to the front of the set. For the hit FIFO is similar to random meaning it does nothing if hit. The LRU is different it moves the hit block to the front of the set block.

All of these replacement policies performances are good but technology is increasing and cache systems need a better framework to increase performance. What we present is a set of bits that determines which blocks to replace based on how often the block has be referenced. This is called a Static Re-Referenced Interval prediction [1]. The replacement policy will have a lower miss rate and replacement rate which will be presented using Simplescalar simulation through 2006 benchmarks as presented in the paper [1].

Static Re-Referenced interval prediction is a replacement policy is and expansion to the LRU policy. Though instead of replacing the most recently used block in set it uses a set of N bits per block that determines which block is the best to be replaced. These N bits are the re-reference values which based off the number of bits determines which block had the longest time when it was referenced. So when the cache is invalid and created it makes all the N bits = 1 which makes the total value of the re-reference value equal to 2^N - 1. So based off the value of the bits we can determine 3 different kinds of hits in the blocks. The first one is near-immediate which is when the block had a recent hit and is close to the head of the cache set. Next is distant which is that it has been hit but not as immediately as near immediate. Finally is long which is that the block in the set has been hit but long time ago. The reason why these 3 cases are important is to determine which block to replace. Though for an LRU it doesn't matter when one was hit it just removes the last block which is the longest referenced block. Though with a large number of blocks in a set it will be tough to determine which block to replace if there is a miss.

The papers implementation uses simple scalar cache system that is set up into sets with each set containing a linked list of blocks based on the associativity. The paper implementation searches through the link list starting from the head of the set and finds the first block re-reference value that equals (2^n)-1 and replaces it. What we implement is a block moving system that finds the least recent that is also equal to 2^n-1. If it is a hit it moves the block to the front and changes the re-reference value.

As for the results we will present 2 cases based on miss rate and IPC using a certain number of benchmarks. These 2 cases were defined by the paper which will give accurate results when doing a comparison. The first case is a finite set cache size, associativity, and level but there will be a change in the N value. Second case is a change in cache size but a finite set N, associativity, and level. With these 2 case experiments we can determine either our implementation is accurate or better than the paper.

**Motivation:**

Any perfect replacement policy will make exact predictions about re-reference interval of blocks in a cache. This is however not practically feasible as the exact information about when a block will be re-referenced again is not available. It is however possible to make an estimated prediction about when the block will be re-referenced. Many strategies are used like Least recently used (LRU), first in first out (FIFO), except that it tries to predict block re-reference interval. LRU predicts that any block that is referenced now is likely to be re-referenced again in immediate future. This strategy works best for Recency-Friendly Access patterns, which has high temporal locality re-referencing.

This strategy however does not work for cases where a block is accessed only few times and then just sits in the cache without being re-referenced in future. This happens when thrashing access pattern or mixed access pattern exists. This causes cache to be polluted by blocks that are not accessed in near future leading to no space for new blocks.

The paper proposes Static Re-reference Interval Prediction (SRRIP) method that addresses these issues giving an optimum prediction for cache blocks. This method assigns re-reference prediction value to each block in the cache based on how likely it is to be re-referenced in near future vs distant future. Initially, all blocks are assigned distant prediction value. At every miss, a block with distant prediction value is found and replaced. This replaced block is assigned an intermediate prediction value called long prediction that is a little less than distant prediction value. This is done to allow the cache system to learn this prediction value. Over time. It proposes two ways to do this learning - Frequency priority and Hit priority. Hit priority method makes re-reference prediction value to zero on every hit thus predicting near immediate re-reference. Frequency prediction decrements the re-reference prediction value on every hit thus gradually learning the prediction. So for high frequency of hits gives more near immediate prediction and vice versa.

We use this method to

**Description of the proposed improvement:**

We have presented the problem for the implementation of the different form of the Static Re Reference Input prediction policy. We plan to replicate the Frequency priority state re reference input prediction policy (FP-SRRIP) and the Hit priority static re-reference input prediction policy. The way we perform this is by using simplesclar as presented before. The code we changed is the structures in the cache.h and added a new function in the cache.c.

**In cache.h**

* in enum cache\_policy function
  + added HP\_SRRIP and FP\_RRIP
* in struct cache\_blk\_t
  + added int re\_refernce\_value;

**In cache.C**

* added function   
  struct cache\_blk\_t\* SRRIP\_update\_replace(struct cache\_set\_t \*set)
  + THis function is mainly for the cache miss and the major structure of the RRIP total implementation
    - Search the blocks starting from Tail to Head
      * Find (Re\_Reference\_value == ((2^N)-1))
        + if found

move block to head

Re\_Reference\_value = ((2^N)-2))

Return block

* + - * + else if the max value > Re\_Reference\_value

max = Re\_Reference\_value

* + - if nothing returned
      * increment all blocks in the set
        + Re\_Reference\_value += (((2^N)-1) - max)
    - Search the blocks starting from Tail to Head again
      * similar execution to the first search
* In struct cache\_create
  + added re\_reference\_value = ((2^N) - 1);
    - this is added to every block in all the sets
* in cache\_char2policy
  + added cases for HP\_SRRIP, DRRIP, and FP\_SRRIP
* in cache\_config
  + added the flags for HP\_SRRIP, FP\_SRRIP, and DRRIP
* in cache\_access
  + For MISS
    - added case for HP\_SRRIP which is similar to FP\_SRRIP
    - SRRIP case
      * **replacement** = SRRIP\_update\_replace(the set number from the tag(&cp->sets[set]))
      * move the replacement block to the head
  + For HIT
    - FP-SRRIP
      * Block - Re-Reference Value--
      * Block -> Move to Head using **update\_way\_list**
    - HP-SRRIP
      * Cache Hit
        + Block - Re-Reference Value = 0
        + Block -> Move to Head using **update\_way\_list**
  + For Fast\_hit
    - FP-SRRIP
      * Block - Re-Reference Value--
    - HP-SRRIP does nothing

**Sample Run:**

**Simple simulator setup:**

1. Download simplesim-3v0e.tgz
2. untar the downloaded file
   1. tar xzvf simplesim-3v0e.tgz
3. compile the simulator
   1. make config-alpha
   2. make
4. After you get it to work execute ‘sim-outorder’

Benchmark 2006 setup

1. Download the benchmarks and arguments- <http://www.cs.binghamton.edu/~msim/benchmarks/spec2K6/>
2. Download the scripts

[http://students.cse.tamu.edu/khkim/teaching/csce614/runscripts.tg](http://students.cse.tamu.edu/khkim/teaching/csce614/runscripts.tgz)z

* 1. Untar the file using tar command

1. move the scripts to a file
2. move benchmarks to a file
3. Make a file called tests in the same location as the benchmarks and scripts
4. run the scripts
   1. cd (spec2000args dir)/(benchmark)
   2. ./RUN(benchmark) (simplescalar dir)/sim-outorder (spec2006 dir)/benchmark (simplescalar options)
   3. cd tests
   4. EXAMPLE EXECUTABLE

./RUNgcc ../../Project/CSCE614project/simplesim-3.0/sim-outorder ../../spec2000binaries/gcc00.peak.ev6 -max:inst 50000000 -fastfwd 20000000 -redir:sim test\_output\_HP\_SRRIP.txt -bpred 2lev -bpred:2lev 1 256 4 0 -bpred:ras 8 -bpred:btb 64 2 -cache:dl1 dl1:128:64:16:l -cache:dl2 ul2:2048:64:16:h

**Performance Analysis:**

**Data:**

**CSS Benchmark 2006**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| FP-SRRIP |  |  |  | HP-SRRIP |  |  |
| m | Miss rate | IPC |  | m | Miss rate | IPC |
| 1 | 0.0139 | 1.1221 |  | 1 | 0.0139 | 1.1221 |
| 2 | 0.0133 | 1.1229 |  | 2 | 0.0138 | 1.1225 |
| 3 | 0.0123 | 1.1243 |  | 3 | 0.0139 | 1.1225 |
| 4 | 0.0121 | 1.1245 |  | 4 | 0.014 | 1.1225 |
| 5 | 0.0121 | 1.1245 |  | 5 | 0.014 | 1.1226 |
|  |  |  |  |  |  |  |
| LRU | 0.0139 | 1.1221 |  |  |  |  |
|  |  |  |  |  |  |  |
| FP-SRRIP Average | |  | HP-SRRIP Average | |  |  |
| Miss rate | IPC |  | Miss rate | IPC |  |  |
| 0.4236 | 0.2616 |  | 0.4236 | 0.2616 |  |  |
| 0.4242 | 0.2608 |  | 0.4237 | 0.2612 |  |  |
| 0.4252 | 0.2594 |  | 0.4236 | 0.2612 |  |  |
| 0.4254 | 0.2592 |  | 0.4235 | 0.2612 |  |  |
| 0.4254 | 0.2592 |  | 0.4235 | 0.2611 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  | LRU |  | FP-SRRIP | |  | HP-SRRRIP |
| Cache size | IPC | MISS Rate | IPC | Miss Rate | IPC | MISS Rate |
| 256KB | 1.0908 | 0.0473 | 1.0954 | 0.046 | 1.0924 | 0.0483 |
| 512KB | 1.1117 | 0.0249 | 1.1152 | 0.0242 | 1.1137 | 0.0248 |
| 1MB | 1.1221 | 0.0139 | 1.1243 | 0.0123 | 1.1225 | 0.0139 |
| 2 MB | 1.1254 | 0.0106 | 1.1254 | 0.0106 | 1.1254 | 0.0106 |

**MCF Benchmark 2006:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| FP-SRRIP |  |  |  | HP-SRRIP |  |  |
| m | Miss rate | IPC |  | m | Miss rate | IPC |
| 1 | 0.8647 | 0.7225 |  | 1 | 0.8647 | 0.7225 |
| 2 | 0.8534 | 0.7255 |  | 2 | 0.8277 | 0.7338 |
| 3 | 0.8494 | 0.7265 |  | 3 | 0.7638 | 0.7255 |
| 4 | 0.8486 | 0.7268 |  | 4 | 0.6574 | 0.7807 |
| 5 | 0.8484 | 0.7268 |  | 5 | 0.6489 | 0.7841 |
|  |  |  |  |  |  |  |
| LRU | 0.8647 | 0.7225 |  | FP-SRRIP Average | |  |
| Random | 0.86 | 0.7257 |  | Miss rate | IPC |  |
|  |  |  |  | 0 | 0 |  |
|  |  |  |  | 0.013068116 | -0.004152249 |  |
|  |  |  |  | 0.017693998 | -0.005536332 |  |
|  |  |  |  | 0.018619174 | -0.005951557 |  |
|  |  |  |  | 0.018850468 | -0.005951557 |  |
|  |  |  |  |  |  |  |
|  | LRU |  | FP-SRRIP |  |  | HP-SRRRIP |
| Cache size | IPC | MISS Rate | IPC | Miss Rate | IPC | MISS Rate |
| 256KB | 0.7225 | 0.8647 | 0.7265 | 0.8494 | 0.7255 | 0.7638 |
| 512KB | 0.7486 | 0.7664 | 0.7599 | 0.7281 | 0.8076 | 0.4886 |
| 1 MB | 0.649 | 0.0452 | 0.9572 | 0.0253 | 0.9551 | 0.0291 |
| 2 MB | 0.966 | 0.0114 | 0.966 | 0.0114 | 0.966 | 0.0114 |

|  |  |
| --- | --- |
| HP-SRRIP Average | |
| Miss rate | IPC |
| 0 | 0 |
| 0.042789407 | -0.015640138 |
| 0.116687869 | -0.004152249 |
| 0.239736325 | -0.080553633 |
| 0.249566324 | -0.085259516 |

**BZIP2 Benchmark 2006:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| FP-SRRIP |  |  |  | HP-SRRIP |  |  |
| m | Missrate | IPC |  | m | Missrate | IPC |
| 1 | 0.4375 | 1.3837 |  | 1 | 0.4375 | 1.3837 |
| 2 | 0.4496 | 1.3876 |  | 2 | 0.5319 | 1.3855 |
| 3 | 0.4508 | 1.3919 |  | 3 | 0.5819 | 1.3959 |
| 4 | 0.4526 | 1.3932 |  | 4 | 0.61 | 1.4029 |
| 5 | 0.4549 | 1.3938 |  | 5 | 0.6298 | 1.4052 |
|  |  |  |  |  |  |  |
| LRU | 0.4375 | 1.3837 |  | FP-SRRIP Average | |  |
| Random | 0.547 | 1.3672 |  | Missrate | IPC |  |
|  |  |  |  | 0 | 0 |  |
|  |  |  |  | -0.027657143 | -0.00281853 |  |
|  |  |  |  | -0.0304 | -0.00592614 |  |
|  |  |  |  | -0.034514286 | -0.00686565 |  |
|  |  |  |  | -0.039771429 | -0.00729927 |  |
|  |  |  |  |  |  |  |
|  | LRU |  | FP-SRRIP |  |  | HP-SRRRIP |
| Cache size | IPC | MISS Rate | IPC | Miss Rate | IPC | MISS Rate |
| 256KB | 1.3837 | 0.4375 | 1.3919 | 0.4508 | 1.3959 | 0.5819 |
| 512KB | 1.4627 | 0.2491 | 1.4686 | 0.2496 | 1.4583 | 0.3156 |
| 1 MB | 1.52 | 0.1305 | 1.5185 | 0.1358 | 1.5159 | 0.1425 |
| 2 MB | 1.553 | 0.0609 | 1.549 | 0.0629 | 1.5504 | 0.0632 |

|  |  |
| --- | --- |
| HP-SRRIP Average | |
| Missrate | IPC |
| 0 | 0 |
| -0.215771429 | -0.00130086 |
| -0.330057143 | -0.00881694 |
| -0.394285714 | -0.01387584 |
| -0.439542857 | -0.01553805 |

**ART Benchmark 2006:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| FP-SRRIP |  |  |  | HP-SRRIP |  |  |  |  |
| m | Miss rate | IPC |  | m | Miss rate | IPC |  |  |
| 1 | 0.6821 | 0.6035 |  | 1 | 0.6821 | 0.6035 |  |  |
| 2 | 0.6222 | 0.6212 |  | 2 | 0.3769 | 0.6591 |  |  |
| 3 | 0.4843 | 0.6454 |  | 3 | 0.3774 | 0.6591 |  |  |
| 4 | 0.3906 | 0.6625 |  | 4 | 0.3795 | 0.6575 |  |  |
| 5 | 0.3906 | 0.6625 |  | 5 | 0.3804 | 0.6574 |  |  |
|  |  |  |  |  |  |  |  |  |
| LRU | 0.6821 | 0.6035 |  | FP-SRRIP Average | |  | HP-SRRIP Average | |
|  |  |  |  | Miss rate | IPC |  | Miss rate | IPC |
|  |  |  |  | -0.2446 | 0.7802 |  | -0.2446 | 0.7802 |
|  |  |  |  | -0.1847 | 0.7625 |  | 0.0606 | 0.7246 |
|  |  |  |  | -0.0468 | 0.7383 |  | 0.0601 | 0.7246 |
|  |  |  |  | 0.0469 | 0.7212 |  | 0.058 | 0.7262 |
|  |  |  |  | 0.0469 | 0.7212 |  | 0.0571 | 0.7263 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | LRU |  | FP-SRRIP | |  | HP-SRRRIP |  |  |
| Cache size | IPC | MISS Rate | IPC | Miss Rate | IPC | MISS Rate |  |  |
| 256KB | 0.5386 | 0.8338 | 0.5385 | 0.8352 | 0.5355 | 0.9493 |  |  |
| 512KB | 0.5386 | 0.8331 | 0.5386 | 0.8331 | 0.562 | 0.7353 |  |  |
| 1MB | 0.6035 | 0.6821 | 0.6454 | 0.4843 | 0.6591 | 0.3774 |  |  |
| 2 MB | 1.0911 | 0.0478 | 1.1005 | 0.032 | 1.1293 | 0.0299 |  |  |

**%Difference of Miss Rate FP-RRIP:**

**%Difference of IPC FP-RRIP:**

**%Difference of Miss Rate FP-RRIP:**

**%Difference of IPC FP-RRIP:**

**Results:**

With the data we present multiple comparisons of the cases that were presented in the paper. Over all the FP-RRIP compared better than the HP-RRIP and LRU. The data averages out to present a 25% increase in performance for miss rate and IPC. This is due to the fact we implemented a block moving system for the RRIP rather than keep the blocks stationary.  
**Conclusion:**

There is an increase in the need for a better cache system that addresses problem related to block replacement. It should handle accurate predictions for near immediate and distant re-references. The paper presented a FP and HP SRRIP methods that would resolve this issue. We replicated these methods adding a few modifications of ours to improve the results. The major modification is that we moved the blocks in the set to incorporate a better scan for the replacement block. In the end our implementation increased performance drastically as presented in the results.

**Division of work:**

For the division of work it was divided between all group members. Overall work was code implementation, execute and experiment to produce results, compare the results to the paper, write up the presentation, and finally write up the report. For this division of work Larry and Amey both worked on the coding of the implementation. For experimentation Amey produced the results as Larry handled the results and produced the graphs and organization. Last Larry and Amey both worked on the report and the presentation based off the results.

**Bibliography:**

1. Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). SIGARCH Comput. Archit. News 38, 3 (June 2010), 60-71. DOI=10.1145/1816038.1815971 http://doi.acm.org/10.1145/1816038.1815971