SIP: Speculative Insertion Policy for High Performance Caching

Hongil Yoon
Tan Zhang
Mikko H. Lipasti

Technical Report #1676

June 2010

SIP: Speculative Insertion Policy for High Performance Caching

Hongil Yoon
Computer Sciences Department
University of Wisconsin-Madison
ongal@cs.wisc.edu

Tan Zhang
Dept. of Elec. and Comp. Eng.
University of Wisconsin-Madison
tzhang32@wisc.edu

Mikko H. Lipasti
Dept. of Elec. and Comp. Eng.
University of Wisconsin-Madison
mikko@engr.wisc.edu

Abstract

High performance cache mechanisms have a great impact on overall performance of computer systems by reducing memory-access latency. Least-Recently Used (LRU) mechanism can achieve good performance in small workload; however, it suffers from thrashing caused by memory-intensive application. To address this challenge, dynamic insertion policy-DIP, which dynamically switches between LRU and an alternative policy, has recently been proposed. The algorithm, however, applies either one of the two policies to the entire cache based on the total number of misses. Therefore, such algorithm is not flexible enough to adjust to the different memory access pattern of each set.

In this paper, we propose a novel approach, called SIP (Speculative Insertion Policy), to achieve high-performance caching via *workload speculation mechanism*. By utilizing *memory-access direction* and *cache misses/hits* , SIP estimates the size of per-set workload and dynamically selects an *optimized* policy for each set, leading to significant improvement in cache performance with only 13Kbits additional storage requirement over LRU for 1MB cache.

1. Introduction

Least-Recently Used (LRU) policy is well known for its good performance in small workload. However, for applications that work on fairly large workload, especially for those larger than the cache capacity, LRU mechanism suffers from frequent cache misses. This situation is referred as thrashing. To deal with this problem, several alternative algorithms have been proposed [3][4][5][6][7]. Recently, a dynamic insertion policy (DIP) was proposed [6]. DIP dynamically switches between two policies- LRU and BIP. However, this mechanism assigns one of the two policies for the entire cache, and thus cannot adjust to the different access behavior of each set.

In this paper, we propose a novel approach, called SIP (*Speculative Insertion Policy*), to achieve high-performance caching via *workload speculation mechanism*. This mechanism can utilize memory access characteristics in an application to speculate the size of workload for each cache set. Based on the estimated workload size, an appropriate cache-replacement policy is dynamically selected at a set-by-set basis, thus leading to significant performance improvement with a small amount of storage overhead.

The contributions of our work are summarized as follows:

- We present a novel algorithm-SIP, which dynamically selects between LRU and BIP for each set based on the estimated workload size.
- We have analyzed the memory-access patterns of the SPEC2006 Benchmark suite[1] with respect to (i) the number of *sequential* cache misses and (ii) the change in the direction of memory accesses during execution. The results of the analysis have been utilized for SIP to detect thrashing.
- We present a possible hardware design as well as storage overhead for implementing SIP.
- We compared the performance of SIP with that of other policies, such as LFU, NRU, and DIP. Our evaluation shows that SIP can slightly outperform all the other testing policies that have achieved the best performance on each benchmark.
- We also measured the performance of SIP under various configurations such as different size of the history register and that of the workload counter.

The remainder of this paper is organized as follow: §2 provides a brief background to variants of LRU, such as LIP, BIP and DIP. §3 describes our proposed policy-SIP. §4 evaluates experimental results. §5 describes hardware implementation and storage overhead. §6 concludes the paper.

2. Background - Dynamic Insertion Policy

M. K. Qureshi et al. introduced two promising algorithms–*LRU Insertion Policy (LIP)* and *Bimodal Insertion Policy (BIP)*–to address the thrashing incurred by conventional LRU [6]. When cache miss occurs, LIP inserts the new data into the *least-recently-used* cache line instead of the *most-recently-used* cache line done by LRU. If cache hit happens at the newly inserted data again, the data is promoted to the *most-recently-used* line. If a miss occurs in between, this data will be evicted first. With this protection mechanism, the rest of the cache lines are effectively preserved. BIP is an improved version of LIP, which is more flexible in choosing the insertion place during cache line replacement, hence achieving better performance for applications with changing access pattern.

They also proposed *Dynamic Insertion Policy*(DIP), which dynamically selects an optimized policy between BIP and LRU based on the number of misses incurred during runtime. Based on this mechanism, two variants-DIP-Global and DIP-Set Dueling were proposed. These two techniques are distinguished by the number of sets used to keep track of cache misses: DIP-SD assigns a limited number of sets as sampling sets whereas DIP-Global employs all the sets. Despite the different tracking mechanisms, these two algorithms have the same drawback. Each time they assign the same replacement policy to the majority of the sets, and thus are unable to adjust to different access behavior on different sets. As a result, a new dynamic replacement policy with a set-by-set resolution is needed to further enhance cache performance.

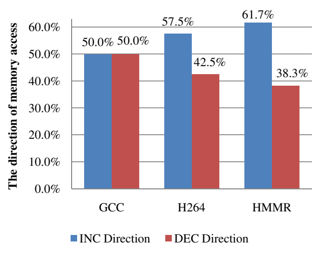
3. Workload Speculation Algorithm for SIP

3.1 Analysis on Memory-Access Pattern

We have analyzed the memory-access patterns of the SPEC2006 Benchmark suite with respect to (i) the number of *sequential* cache misses (Fig. 1 a) and (ii) changes in the direction of memory access during execution (Fig. 1 b). We choose two applications from the benchmark–GCC and HMMR–to discuss how these two memory-access characteristics of applications affect the performance of a cache-replacement

SCM	GCC	HMMR
1	**77.2%**	51.1%
2	10.3%	2.2%
3	4.2%	2.0%
4	2.2%	3.7%
5	1.3%	3.3%
6	0.9%	3.5%
7-Max	3.9%	**34.2%**

(a)



(b)

Figure 1. Analysis on Memory-Access Pattern

policy. When LRU and BIP policies are considered, LRU leads to the best performance on GCC (total hit-rate 43.7%), whereas BIP works the best for HMMR(total hit-rate 48.8%). When LRU is used in HMMR, thrashing arises (total hit-rate 10.3%). The analysis results are summarized as follows:

- **Sequential cache-miss count.** We have tracked every cache access to record how many times sequential cache misses(SCM) occur and how long each SCM lasts. Fig. 1(a) shows the number of SCM for each application. When using LRU as the replacement policy, there was only 4% of more than seven SCM in GCC while 38% of the SCM occurred in executing HMMR. In addition, the number of SCM ranges from 1 to 224 for GCC, whereas 1 to 472 for HMMR. As a result, a frequent and long SCM indicates the likelihood of thrashing.
- **Changes in the direction of cache access.** For each cache set, we compared the tag value of the currently accessed line with that of the previously accessed line. If the current tag is greater than the previous tag, we consider this as the cache access being in upward direction, and vice versa; when the previous tag equals to the current tag, we consider this as the access being in the same direction of the previous access.

  Fig. 1(b) shows the average percentages of upward and downward memory access. In GCC, for which LRU works best, the percentages of upward and downward access are balanced, which implies a non-biased accessing direction. In HMMR, on the other hand, upward access takes more percentage, which indicates a biased accessing direction. This directional characteristic is also observed from our simulation on other applications. Hence, we conclude that a *biased* access direction can also be used as an indicator of possible thrashing.

3.2 SIP - Speculative Insertion Policy

Based on the analysis results mentioned in §3.1, we developed a new algorithm that dynamically selects an *optimized* replacement policy for each set. The selection criteria is based on the size of workload per set that is estimated via cache hits/misses as well as memory-access directions on each set. If the estimated workload size of a cache set is smaller than cache associativity, LRU is selected for the set, otherwise, BIP is chosen.

Workload speculation mechanism: To estimate the workload size of each cache set during execution, we introduce a *history register* and a *workload counter* (WC) for each set. The history register is a shift register to keep track of the direction of each cache access while WC is a saturated counter to estimate the size of workload for each set. The information about memory-access direction and cache misses/hits are utilized to estimate the size of workload for each set as follow:

Alg. 1 Algorithm for Replacement policy decision in SIP

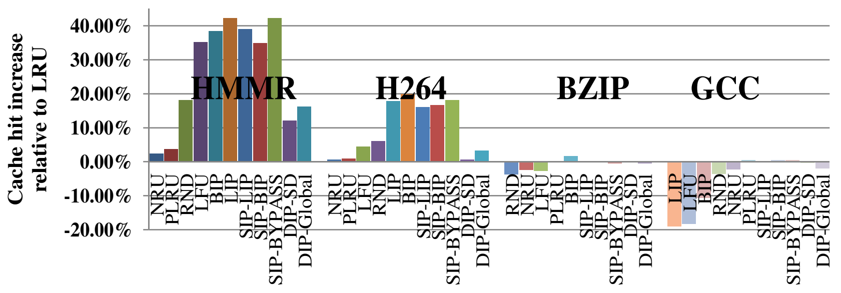
```
Require: WC ← 0 : Workload Counter
Require: HR ← 0 : History Register

[1] if cache hit {
[2]    if current tag ≠ previous tag
[3]       if current policy ≠LRU
[4]           Increase WC
[5]       else
[6]           Decrease WC
[7] } else { // cache miss
[8]    Shift HR
[9]    if current tag > previous tag
[10]      Set current history bit to 0
[11]   else
[12]      Set current history bit to 1
[13]   Calculate GAD based on HR
[14]          // GAD: General Access Direction
[15]   if GAD has not changed
[16]      if Current direction == GAD
[17]      Increase WC // penalty to LRU(Thrashing)
[18]   else //GAD has changed
[19]      if current policy ≠LRU
[20]         Decrease WC // penalty to BIP
[21]      else // current policy is BIP
[22]         Increase WC // penalty to LRU
[23]}
[24] if WC is saturated
[25]    WC = #associativity + 1 // stay in BIP
[26] if WC > #associativity
[27]    Choose BIP as a new policy
[28] else
[29]    Choose LRU as a new policy
```

- When a cache hit occurs, WC is changed to reinforce the current replacement policy under the assumption that a cache-hit event indicates the current policy works well. To prevent the counter from being saturated too fast, we exclude consecutive cache hits from being reinforced.(lines [1]–[6] in Alg.1)
- When a cache miss occurs, the cache controller compares the tag bits of newly inserted line with the tag bits of previously accessed line to detect access direction for this set. The history register shifts in the outcome of the comparison, which represents the current access direction. By calculating how many bits in the history register representing upward direction and how many representing decreasing direction, a new general access direction for each set can be obtained.(lines [8]–[13] in Alg.1)

  Condition1: If the general access direction has not changed and the most recent access direction stored in the history register is the same as the general direction, based on the analysis mentioned in §3.1, WC is increased to favor BIP. (lines [15]–[17] in Alg.1)

  Condition2: If the general access direction has changed, A penalty applies to the current policy by changing WC to favor the opposing policy. (lines [18]–[22] in Alg.1)

In order to promptly shift from BIP to LRU according to the change of the access pattern, the counter is reset to be slightly biased to BIP once it is saturated.(line [25] in Alg.1) When WC is greater than cache associativity, which is the selection threshold, the cache controller assumes that the cache set is working under memory-intensive workloads, thus choosing BIP. Otherwise, LRU is selected (lines [24]–[29] in Alg.1).


(a)

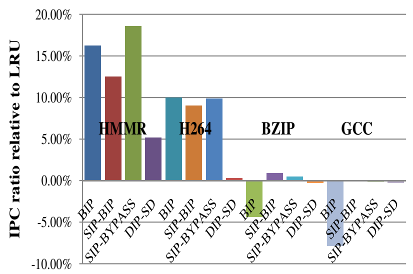

(b)

Figure 2. (a) Hit-rate comparison of SIP and other testing algorithms. (b) IPC comparison of LRU variants.

3.3 Classification of SIP Behavior

In this section, we consider three typical situations to understand how SIP works.

- Situation 1: *Thrashing with a biased access direction*

  In this case, due to condition 1, the workload counter is increased and will eventually excess the threshold. Based on the counter value, the cache controller chooses BIP to protect the data in the cache.

- Situation 2: *Turning point between thrashing and small workload*

  In general, random access pattern causes consecutive cache misses accompanied by frequent changes in general access direction. In this case, according to the condition 2, the workload counter fluctuates around the threshold, and thus the cache controller alternately selects between LRU and BIP, which guarantees a certain degree of protection for data already in the cache.

- Situation 3: *Processing a small amount of workload*

  When a cache set is working under a small workload, much smaller than the size of the cache set, the value of the workload counter would be small as a result of short SCM and the LRU policy would be chosen.

3.4 Variants of SIP–$SIP_{BYPASSING}$ and SIP_{LIP}

We introduce two variants of SIP, in which BYPASSING and LIP are crafted into SIP instead of BIP, respectively. Through bypassing data(without inserting data into cache), $SIP_{BYPASSING}$ can provide further protection to the data under thrashing, hence achieving better performance; on the contrary, by eliminating the use of the BIP counter, SIP_{LIP} can reduce hardware cost.

4. Evaluation

4.1 Simulation Configuration

To evaluate the performance of SIP, we employed a simulation framework based on the CMP$im simulator in single-thread mode[2]. The memory hierarchy contains three level caches. Each cache supports 64-byte lines. The first-level cache is split into a 32KB instruction cache and a 32KB data cache, which are 4-way and 8-way set associative, respectively. The second level cache is 256KB and 8-way set-associative. The Last Level Cache (LLC) is 1MB and 16-way set-associative. Our SIP is applied to LLC.

4.2 Performance Evaluation under SPEC2006

Fig. 2(a) shows the relative hit rate among the testing policies–SIP variants, LRU variants, Pseudo LRU (PLRU), Least Frequently Used (LFU), Random (RND) and Not Recently Used (NRU). In addition, Fig. 2(b) shows the relative IPC among SIP variants and LRU variants. All the baseline caches use LRU. We selected four benchmarks under which the cache hit-rate of LRU is over 10%.

- For HMMR and H264, which have large workload, LRU variants, such as BIP, LIP and DIP, perform much better than LRU. Our proposed policy-SIP_{BIP} outperforms LRU by 12.5% in IPC and 34.9% in hit-rate. One of SIP variants, $SIP_{BYPASSING}$, achieves the best performance among all testing policies. We analyzed the behavior of SIP_{BIP} for each set in more detail in §4.5.
- For GCC, in which LRU achieves the best performance among all the testing policies other than SIP, SIP_{BIP} outperforms LRU by 0.39% in hit-rate and shows the same IPC as that of LRU as shown in Fig. 2(b). Moreover, SIP_{BIP} surpasses BIP by 11% in hit-rate, which demonstrates the benefits rendered by dynamically selecting *optimized* policy during run-time.
- For BZIP, SIP_{BIP} does not surpass either LRU or BIP in average hit-rate. A possible reason is that the gap between the hit rates of BIP and LRU is too small for the SIP controller to decide which policy to use; as a result, it chooses between LRU and BIP in a fluctuating manner, as described in Situation 2 (§3.3).

In conclusion, when compared to other testing policies that show the best performance on each benchmark, IPC improvement of SIP_{BIP} ranges from +0.86% (over LRU-GCC) to -3% (over BIP-HMMR).

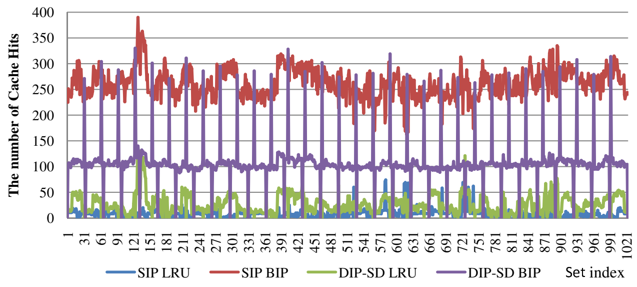
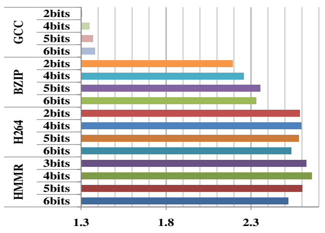


Figure 3. Comparison performance of SIP and DIP-SD for each set

4.3 Analysis on Policy Selection

Fig. 3 shows the numbers of cache hits on each set for SIP_{BIP} and DIP-SD under HMMR benchmark. In DIP-SD, there are two sampling sets solely assigned with LRU and BIP to com-



(a)

Bench	6-bit	4-bit	2-bit
HMMR	2.612	2.606	2.602
H264	2.582	2.580	2.582
BZIP	2.358	2.359	2.355
GCC	1.371	1.370	1.370

(b)

Figure 4. Variations of SIP: (a) IPC of SIP for different Working-set Counter size; (b) IPC of SIP for different History bit size

pare the numbers of misses incurred by each policy respectively. We observe that the sampling set with BIP policy incurs high hit rate while the sampling set with LRU incurs low hit rate. Hence, the value of the PSEL counter, which is the selection threshold, fluctuates between LRU and BIP due to cache misses experienced in LRU-sampling set. As a result, the rest of the cache sets incur medium hit rate. For SIP, most of the cache sets choose BIP during execution, hence achieving much higher hit-rate at the majority of the cache set.

4.4 Variation in Counter Bits

Fig. 4(a) records the IPC of SIP under different sizes of workload counters. We can see that the 4-bit and 5-bit counters achieve better performance. The 5-bit counter outperforms the 4-bit counter in GCC and BZIP, but loses in HMMR and H264. The rationale behind this is that a smaller counter is easier to saturate, hence more biased to BIP: a larger counter takes more time to approach the selection threshold, therefore more inclined to LRU. In order to achieve decent performance among a wide range of applications, an unbiased configuration is required. To this end, we decide to set the number of counter bits to 5. Since the half of the maximum counter value is the same as cache associativity, our policy chooses between LRU and BIP fairly

4.5 Variation in History Bits

Fig. 4(b) shows the IPC of SIP under different numbers of history bits. From the simulation result, we can observe that varying the number of history bits has little impact on performance. The reason is that a small number of history bits are sufficient to filter out the interference of access fluctuation on calculating the general direction.

5. Hardware Design

To implement SIP_{BIP} in hardware, a small amount of storage is required for each set to store the following information bits: a 2-bit history register, a 5-bit workload counter, a current policy bit, a general direction bit, and a 4-bit index to indicate the most recently accessed line for tag comparison. Also, 4 position bits are needed for each cache line to track the MRU and LRU position. And a 5-bit BIP counter is used for entire cach. In a 1MB-16 set associative cache, the storage overhead is 77Kbits in total, among which 64Kbits are for original LRU, and 13Kbits are the additional overhead incurred by SIP_{BIP}.

Fig. 5 describes the proposed hardware design to implement SIP_{BIP}. A workload calculator serves as the controller to estimate the size of workload for each set and to dynamically choose between BIP and LRU based on the estimated workload size. Inside the workload calculator, there is a compara-

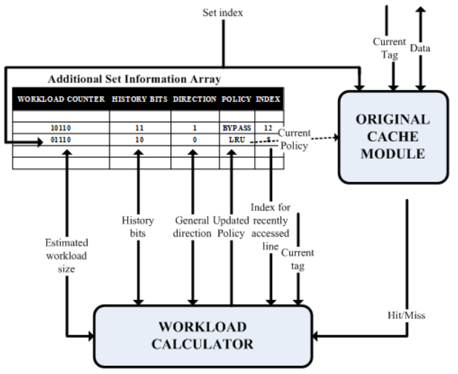


Figure 5. Hardware implementation of SIP

tor that compares the current tag with the previous tag in the same set to decide the current access direction. Also, some combinational logics are used to obtain the general access-direction. Another comparator is used to detect changes between current and previous general directions.

For $\text{SIP}_{\text{BYPASSING}}$, although it shows the best performance among all the testing policies, it requires an additional register to store the previous tag value for each set. The reason is that the cache controller bypasses data blocks under thrashing, and hence the previous tag value cannot be retrieved from the most recently accessed line as is done by SIP_{BIP}. In total, $\text{SIP}_{\text{BYPASSING}}$ requires additional 48Kbits for a 64-bit machine and 16Kbits for a 32-bit machine. Based on the performance evaluation, however, such significant increase in storage overhead of $\text{SIP}_{\text{BYPASSING}}$ is only traded for 1.57% performance gain in IPC over SIP_{BIP}. Therefore, we can conclude that SIP_{BIP} is an optimal policy in terms of high performance and low storage overhead.

6. Conclusion

In this paper, we present a new approach, called SIP, to improve cache performance by dynamically selecting an optimized replacement policy for each cache set via *workload speculation mechanism*. Our experimental results show that SIP can achieve decent performance among all the testing policies under a wide range of applications.

References

[1] SPEC2006 standard performance evaluation corporation. Available at "http://www.spec.org/cpu2006/".

[2] A. J. et al. Cmp$im: A pin-based on-the-fly multi-core cache simulator. 2003.

[3] R. S. et al. Effective shaping of cache behavior to workloads. In *IEEE Micro, Vol. 39*, 2006.

[4] O. M. M. K. Qureshi, D. N. Lynch and Y. N. Patt. A case for mlp-aware cache replacement. In *International Symposium on Computer Architecture*, 2006.

[5] N. Megiddo and D. S. Modha. Arc: A self-tuning, low overhead replacement cache. In *Proceeding of the 2nd USENIX Conference on File and Storage Technologies*, 2003.

[6] M. K. Qureshi. Adaptive insertion for high-performance caching. In *International Symposium on Computer Architecture*, 2007.

[7] M. K. Qureshi. Set-dueling-controlled adaptive insertion for high-performance caching. In *IEEE Micro, Vol. 28, Issue 1, January*, 2008.