Instruction-based Reuse Distance Prediction Replacement Policy

Pavlos Petoumenos, Georgios Keramidas, Stefanos Kaxiras

Instruction-based Reuse Distance Prediction Replacement Policy

Pavlos Petoumenos
ECE, University of Patras, Greece
ppetoumenos@ece.upatras.gr

Georgios Keramidas
Industrial Systems Institute, Patras, Greece
keramidas@ece.upatras.gr

Stefanos Kaxiras
ECE, University of Patras, Greece
kaxiras@ece.upatras.gr

Abstract— This paper presents a new cache replacement policy based on Instruction-based Reuse Distance Prediction (IbRDP) Replacement Policy originally proposed by Keramidas, Petoumenos, and Kaxiras [5] and further optimized by Petoumenos et al. [6]. In these works [5,6] we have proven that there is a strong correlation between the temporal characteristics of the cache blocks and the access patterns of instructions (PCs) that touch these cache blocks. Based on this observation we introduced a new class of instruction-based predictors which are able to directly predict with high accuracy at run-time when a cache block is going to be accessed in the future, a.k.a. the reuse distance of a cache block. Being able to predict the reuse distances of the cache blocks permits us to make near-optimal replacement decisions by "looking into the future."

In this work, we employ an extension of the IbRDP Replacement policy [6]. We carefully re-design the organization as well as the functionality of the predictor and the corresponding replacement algorithm in order to fit into the tight area budget provided by the CRC committee [3]. Since our proposal naturally supports the ability to victimize the currently fetched blocks by not caching them at all in the cache (Selective Caching), we submit for evaluation two versions: the base-IbRDP and the IbRDP enhanced with Selective Caching (IbRDP+SC).

Our performance evaluations based on a subset of SPEC2006 applications show that IbRDP achieves an IPC improvement of 4.66% (arithmetic average) over traditional LRU, while IbRDP+SC is able to further increase its distance compared to the baseline LRU to 6.04%. Finally, we also show that IbRDP outperforms the previous state of the art proposal (namely Dynamic Insertion Policy or DIP [7]) by 2.32% in terms of IPC (3.81% for the IbRDP+SC).

1. INTRODUCTION

With fast advances in processor technology, the rapidly increasing imbalance between the relative speeds of processors and the main memory is the predominant problem that computer architects are trying to mitigate. With each access to the main memory taking hundreds of processor cycles, caches play an important role in bridging this performance gap. Therefore an increasingly larger portion of the chip silicon area budget is devoted to on-chip cache hierarchies (more than 60%). However, prior studies have shown that only a small fraction of the cache blocks actually hold data that will be referenced before evicted from the cache (called *live blocks* [4]). Cache efficiency and not cache capacity is the correct metric to characterize the performance of the cache [2]. Cache efficiency can be improved if more live blocks are stored in the cache without increasing its capacity. There are two ways to improve cache efficiency: i) by devising sophisticated prefetching mechanisms and ii) by improving the underlying replacement policy of the cache.

In this paper we investigate the design and implementation of a replacement policy for last-level caches (L3 caches) for single-threaded applications which adheres to the storage restrictions imposed by the committee of the first Cache Replacement Championship (CRC-1) [3]: 8 bits per cacheline and 1 Kbit additional storage, for a total of 129 Kbit with unlimited logic complexity. The proposed mechanism leverages on our previous proposal, the Instruction-based Reuse Distance Predictor (IbRDP) Replacement Policy [5,6].

The remainder of this paper is organized as follows: Section 2 outlines the overall design, details the functionality of our new Instruction-based Reuse Distance Predictor and presents our simplified technique to collect at run-time the reuse distances of the cache blocks (reuse distance sampling). Section 3 describes our underlying replacement policy based on the information supplied by the IbRDP. Section 4 presents the hardware costs and demonstrates that our proposal adheres to the contest rules. Section 5 depicts our experimental results in terms of IPC compared to the traditional LRU and DIP [7] and Section 6 concludes this work.

2. PREDICTOR DESIGN AND FUNCTIONALITY

The effect of caching is fully determined by the program locality or the data reuse patterns of the memory references. As the memory hierarchy becomes deeper and deeper its performance increasingly depends on our ability to predict program locality. Previous work discloses mainly two ways of locality analysis: compile-time analysis and profiling. Ideally, a prediction scheme is needed that can be used at *run-time* and can be both efficient and accurate. In our previous work [5,6], we demonstrate that it is possible to quantify the temporal characteristics of the cache blocks at run-time by predicting the cache block reuse distances (measured in intervening cache accesses) based on the instructions (PCs) that touch the cache block.

Figure 1 outlines our proposal for predicting the reuse distances of the cache blocks at run-time: the Instruction based Reuse Distance Predictor or **IbRDP** (Figure 1

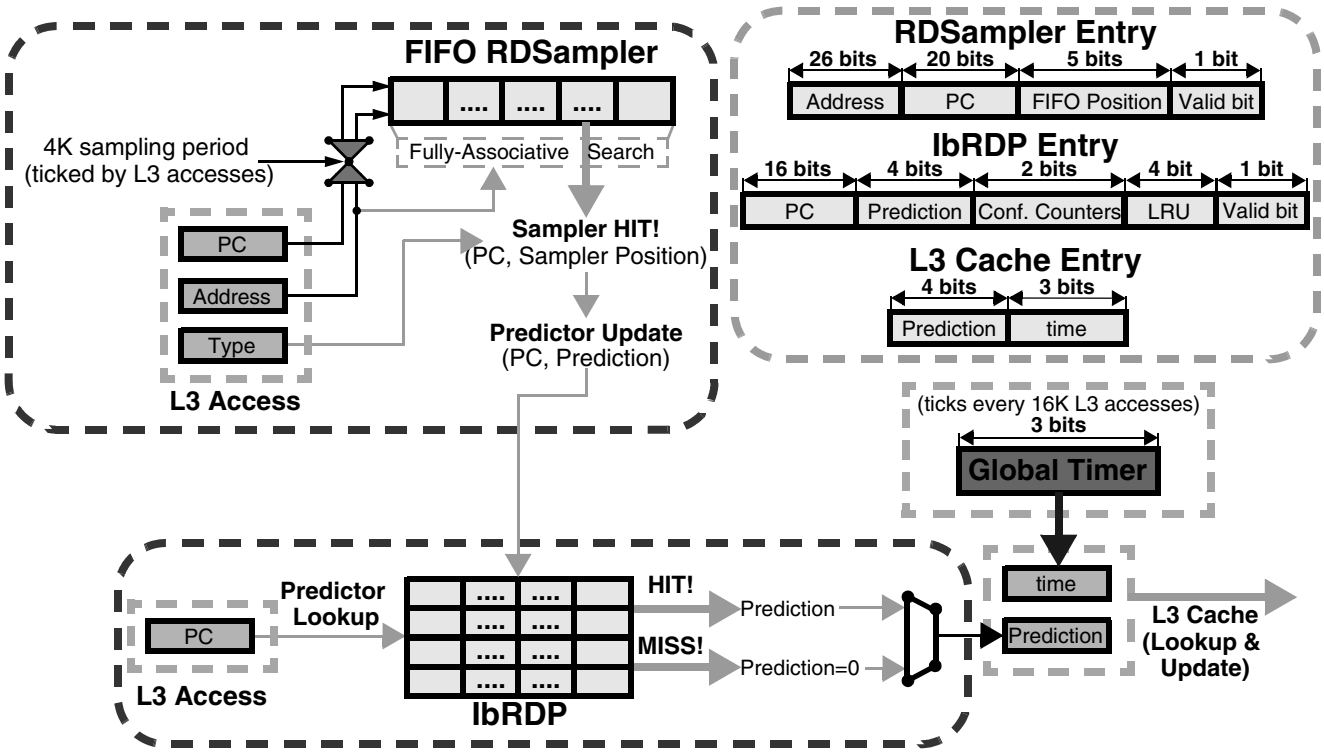

Figure 1. Overall design of the proposed mechanism: Instruction-based Predictor (bottom), reuse distance FIFO Sampler (top left), detailed format of entries (top right).

bottom). IbRDP is responsible to store the reuse distance predictions of the cache blocks. Due to area and performance constrains the reuse distance predictions are not represented with their actual scalar value but by using a granularity of 8K L3 accesses[1]. On every load or store L3 access the IbRDP is triggered by the instruction (PC) that caused the access (*Predictor Lookup*). In case of a hit, the IbRDP simply outputs the corresponding prediction (zero if the lookup misses).[2] An IbRDP hit means that this PC has already appeared in the program execution. The mechanism which is in charge of capturing and forwarding (*Predictor Update*) the correlations between PCs and reuse distances is the **Reuse-Distance Sampler** or **RDSampler** (Figure 1).

The **RDSampler** is organized as small FIFO queue (32 entries) sampling the L3 access stream. A new entry is inserted in the sampler every 4K accesses (empirically derived). Since it is a FIFO, its size and sampling rate determine the maximum reuse distance it can capture: *max_reuse_distance=size* × *sampling_period*. For example, our 32-entry FIFO sampling every 4K accesses can "see" the reuse distances of up to 32×4K=128K cache accesses. Apart from the insertion of a new entry in the **RDSampler**, for every L3 access a fully-associative search is also performed, triggered by the requested address. In case of a hit, we mark the hit entry as invalid, we calculate the quantized reuse distance of the access based on the position of the entry in the FIFO queue and we use this information (along with the PC) to update the IbRDP (*Predictor Update*). If the IbRDP does not contain information for the sampled PC, the LRU entry of the corresponding IbRDP set is filled with the information provided by the **RDSampler**. Otherwise the sampled reuse distance is compared with the prediction stored in the predictor. If they match, the confidence counter (2-bit field in each entry of IbRDP) is incremented otherwise decremented. Finally, the predictor entry can change its reuse distance prediction (predictor replacement) only if its confidence counter is zero.

[1] Granularities finer than 8K do not offer much additional useful information and make harder the prediction of the reuse distances (it is unlikely to encounter the exact same reuse distance over and over again due to the noise introduced at the access stream).

[2] According to our study a predictor size of 256 entries is more than enough to capture all the significant PC's in a typical program phase. Additional entries would hold information for rarely executed load/stores that may not be easy to predict and do not significantly improve the effectiveness of our prediction.

3. Underlying Replacement Policy

Having the quantized reuse distance information for each cacheline affords us to approximate an optimal replacement algorithm because we can *"see"* the future (as in Belady's OPT [1]). On a miss we search the cache set. The pseudocode of the proposed replacement policy is depicted in Figure 2. Our aim is to find the cacheline that is going to be accessed farthest in the future (`time_left` variable in the pseudocode). The `time_left` (line 15-16 in the

```
1:  procedure FIND_REPLACEMENT()
2:     max=-1
3:     // SC controls whether Selective
4:     //   Caching will be applied or not
5:     SC=FALSE
6:
7:     if SC == TRUE then
8:        if curr_miss.prediction == 15 then
9:           candidate=curr_miss
10:          return candidate
11:       end if
12:    end if
13:
14:    for way=1, number_of_ways do
15:       time_left=way.prediction + \
16:                 way.timer - global_timer
17:       time_idle=global_timer – way.timer
18:       if time_left > max then
19:          max = time_left
20:          candidate = way
21:       end if
22:       if time_idle > max then
23:          max = time_idle
24:          candidate = way
25:       end if
26:    end for
27:
28:    if SC == TRUE then
29:       if curr_miss.prediction > max then
30:          candidate=curr_miss
31:       end if
32:    end if
33:    return candidate
34: end procedure
```

Figure 2. The Underlying Replacement Algorithm.

preudocode) of a cacheline equals the time it was last accessed (the 3-bit time field stored in every cacheline) plus its reuse distance prediction (the 4-bit prediction field also stored in every cacheline) minus the current time (the 3-bit quantized value of the global timer). In other words, a cacheline's `time_left` is the (predicted) number of accesses that separate the present moment from its next access. Of course due to the tight area budget requested by the championship committee [3] (max 8 bits per cacheline) all the available information (time field, prediction field and global timer) is quantized and aliasing will occur. We systematically explored the granularity of the prediction and timestamp quanta in order to achieve a good balance between accuracy and storage requirements.

In addition to the granularity problem there is another issue that we should take into account. What happens with the cache blocks which have no reuse distance prediction or whose `time_left` is negative —the predicted time of the access has passed. In this case, we need to also look into the past. The `time_idle` variable (line 17) estimates the difference between the line's last access and the global timer i.e., how long the line has remain unaccessed. The longer a line remains unaccessed the higher the probability that this line is not useful [4]. These variables give us two ways to identify candidates for replacement, with both ways being quantified by exactly the same metric: time measured in L3 accesses. The eventual candidate will be the line with the largest `time_idle` (past) or `time_left` (future) value, because it either has remained unused or will remain unused longer in the cache than any other currently allocated cacheline. This guarantees that if we have prediction information, the replacement decision will be based on it; otherwise we revert to an LRU-like replacement.

The above algorithm (line 14 to 26) represents our base-IbRPD replacement policy. An extension of the algorithm is to victimize the currently fetched block by not caching it at all in the L3 cache (Selective Caching). This happens *if* the prediction for the currently fetched block is larger than the `time_left` or the `time_idle` value of the block selected by the base-IbRDP (lines 28-32). A final touch on the algorithm is an optimization which we experimentally found to be very beneficial. If the prediction for the currently fetched block is 15 (the maximum value), we choose to bypass the cache without even comparing against the other cachelines in the set (lines 7-12).

4. HARDWARE COST AND RULE COMPLIANCE

This section breaks down the area cost of the components used by our proposal and summarizes the overall cost. Recall that even though we submitted for evaluation two versions of our proposal (the base-IbRDP and the IbRDP enhanced with Selective Caching), both approaches account for exactly the same area budget. The detailed storage requirements are:

RDSampler: 32 entries organized as a fully-associative buffer. Each entry contains: 26 bits for the address under investigation (lowest 32 bits minus the byte offset) plus 20 bits for the PC plus 5 bits containing the position of the entry in the FIFO buffer plus 1 valid bit. Total bits per entry: 52. Total bits for the FIFO sampler: 1664 bits. (We experimentally found that keeping more bits for either the PC or the recorded address produces marginal benefits).

IbRDP: 256 entries organized as a 16-way set-associative cache (16 sets). Each entry contains: 16 bits for the recorded PC (20 bits minus the bits needed for indexing) plus 4 bits for the prediction information plus 2 bits for the confidence counters plus 4 bits for the local LRU information plus 1 valid bit. Total bits per entry: 27. Total bits for the predictor: 6912 bits.

L3 Cache: 16K cachelines (1MB, 16-way set-associative cache). For management related purposes each block contains: 4 bits for the prediction plus 3 bits for the timestamp. Total bits per entry: 7. Total bits for the L3 Cache: 112 Kbits.

Extra variables: i) 12 bits for accounting of the 4K sampling period, ii) 3 bits of the higher portion of the

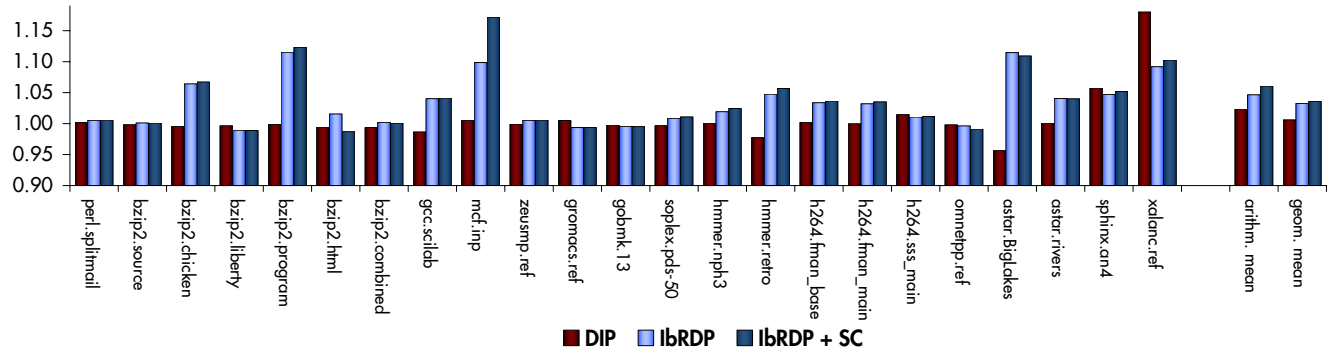

Figure 3. IPC for the DIP, IbRDP, and IbRDP+SC normalized to LRU.

accesses counter which provides the timestamp, and iii) 14 bits of the lower part of the accesses counter. Total bits for the extra variables: 29. The overall budget for our proposal amounts to 120.4 Kbits, below the 129 Kbit limit [3].

5. EVALUATION

For our simulations, we relied on the standard version of the framework provided by the organizing committee of CRC-1 [3]. Details about the processor and the memory configuration can be found in [3]. Performance evaluations were based on 23 benchmark-input combinations from the SPEC2006 suite. The benchmark selection was done according to the following criterion: we exclude the benchmarks that do not report more than 1% decrease of their execution time when the L3 cache is doubled. Excluded benchmarks have either very low or very high cache requirements and do not benefit much from advanced replacement decisions. All executables were built with a 64-bit version of gcc-4.2 using the -O2 and -msse3 flags. To generate instructional traces, 40B instructions were skipped and a trace 100M instructions long was obtained.

Figure 3 depicts the performance improvements in terms of IPC over LRU. The first bar in each set of bars represents the IPC improvement for DIP [7], while the second and the third bar illustrate the results for the base-IbRDP and the IbRDP+SC. Our results indicate that the benefits are significant for both the base-IbRDP and the IbRDP+SC. In the base-IbRDP case, *bzip2.program* shows the greatest performance gains over LRU (11.52%), while significant speedups are reported in the other benchmarks. A 11.46% speedup is achieved in *astar.BigLakes*, 9.87% in *mcf.inp*, and 6.43% in *bzip2.chicken*. Compared to DIP, our approach is almost always superior. IbRDP outperforms DIP by 2.32% on average (arithm.) and in no case degrades performance as much as DIP (up to 4.36%). When the selective cache technique is employed, our approach clearly outperforms DIP: IbRDP+SC increases IPC relative to DIP by up to 16.66% and 3.81% on average (arithm.).

6. CONCLUSIONS

In this paper we propose an Instruction-based Reuse Distance Prediction Replacement Policy suitable for last-level caches. The proposed policy was based on our previous works presented in [5,6]. We carefully simplify the structure and the functionality of our predictor in order to fit into the tight area budget provided by CRC-1. We submit two versions for evaluation: the base-IbRDP and IbRDP with Selective Caching. We evaluate our proposals with a 129 Kbit storage budget in the CRC-1 framework. The evaluation results show a 4.66% IPC improvement (arithm. average) over LRU for the base-IbRDP (6.04% for the IbRDP+SC) by using a subset of the SPEC2006 suite.

7. ACKNOWLEDGEMENTS

This work is supported by the EU-FP6 Integrated Project, Scalable computer ARChitecture (SARC), Contract No. 27648 and the EU-FP7 ICT Projects, "A highly efficient adaptive multi-processor framework (HEAP)," Contract No. 247615, and "Embedded Reconfigurable Architecture (ERA)," Contract No. 249059.

8. REFERENCES

[1] L. A. Belady. "A study of replacement algorithms for a virtual-storage computer." *IBM Systems Journal*, 1966.

[2] D. Burger, J. R. Goodman, and A. Kagi. "The declining effectiveness of dynamic caching for general-purpose microprocessors." *Technical Report, Computer Sciences Dept, Univ. of Wisconsin,* 1995.

[3] Cache Replacement Championship Homepage, 2010. *http://www.jilp.org/jwac-1/*

[4] S. Kaxiras, Z. Hu, and M. Martonosi. "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power." *Proc. of the International Symposium on Computer Architecture*, 2001.

[5] G. Keramidas, P. Petoumenos, and S. Kaxiras. "Cache Replacement Based on Reuse Distance Prediction." *Proc. of the International Conference on Computer Design,* 2007.

[6] P. Petoumenos, G. Keramidas, and S. Kaxiras. "Instruction-based Reuse Distance Prediction for Effective Cache Management." *Proc. of the International Symposium on Systems, Architectures, Modeling, and Simulation*, 2009.

[7] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely. Jr., and J. Emer. "Adaptive insertion policies for high-performance caching." *Proc. of the International Symposium on Computer Architecture*, 2007.