Reuse Distance-Based Probabilistic Cache Replacement

SUBHASIS DAS, Stanford University
 TOR M. AAMODT, University of British Columbia
 WILLIAM J. DALLY, NVIDIA/Stanford University

This article proposes Probabilistic Replacement Policy (PRP), a novel replacement policy that evicts the line with *minimum estimated hit probability under optimal replacement* instead of the line with maximum expected reuse distance. The latter is optimal under the independent reference model of programs, which does not hold for last-level caches (LLC).

PRP requires 7% and 2% metadata overheads in the cache and DRAM respectively. Using a sampling scheme makes DRAM overhead negligible, with minimal performance impact. Including detailed overhead modeling and equal cache areas, PRP outperforms SHiP, a state-of-the-art LLC replacement algorithm, by 4% for memory-intensive SPEC-CPU2006 benchmarks.

Categories and Subject Descriptors: B.3.2 [Memory Structures]: Design Styles—Cache memories; C.4 [Performance of Systems]: Design Studies

General Terms: Design, Performance

Additional Key Words and Phrases: Cache replacement policy, reuse distance, probabilistic replacement

ACM Reference Format:

Subhasis Das, Tor M. Aamodt, and William J. Dally. 2015. Reuse distance-based probabilistic cache replacement. ACM Trans. Archit. Code Optim. 12, 4, Article 33 (October 2015), 22 pages.
 DOI: <http://dx.doi.org/10.1145/2818374>

1. INTRODUCTION

Last-level cache misses cause off-chip accesses that consume significant energy and impact performance via higher latency and limited bandwidth. Conventional replacement policies, such as LRU, perform poorly on LLCs because the easy references, with short reuse distances, have been filtered by the upper-level caches leaving a references stream dominated by moderate and long reuse distances. Even scan-resistant replacement algorithms, such as DRRIP [Jaleel et al. 2010] and SHiP [Wu et al. 2011], perform poorly on these LLC reference streams because they are unable to discriminate references with moderate reuse distances from those with long reuse distances.

Many conventional cache replacement strategies, such as LRU, are based on an “informal principle of optimality” [Aho et al. 1971] that states that hit rate is maximized by replacing the block with maximum expected time to reuse. Under two simplified program reference models, the independent reference model (IRM) [Aho et al. 1971], and the LRU stack model [Mattson et al. 1970], this principle reduces to two well-known replacement policies: the LFU and LRU policies, respectively. However, these

Authors’ addresses: S. Das and W. J. Dally, Gates Computer Science Building, 353 Serra Mall, Stanford, CA, 94305, USA; emails: {subhasis, dally}@stanford.edu; T. M. Aamodt, Department of Electrical and Computer Engineering, University of British Columbia, 2332 Main Mall, Vancouver, BC, V6T 1Z4, Canada; email: aamodt@ece.ubc.ca.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.

© 2015 ACM 1544-3566/2015/10-ART33 \$15.00

DOI: <http://dx.doi.org/10.1145/2818374>

Fig. 1. Reuse distances (top) and hit rates of accesses with different reuse distances (bottom) for mcf, cache size = 4MB, and block size = 64B.

models are a poor approximation to the access stream at the LLC. In this article, we show it is better to replace the block with minimum estimated probability of receiving a hit before being evicted.

To demonstrate the practical benefits of this approach, we introduce two techniques that combined improve hit rate over current hardware cache replacement algorithms. First, to enable calculation of hit probability, we record a coarse-grained reuse distance distribution rather than a scalar proxy for expected reuse distance. Second, to discriminate reuse distances longer than the size of the cache, we retain these distributions for blocks not currently in the cache, i.e., retain select metadata for evicted blocks [Stone 1993; O’Neil et al. 1993]. To greatly reduce overhead, PRP retains metadata at page granularity. While Takagi and Hiraki have proposed the IGDR scheme that utilizes distributions [Takagi and Hiraki 2004] to perform cache replacements, their scheme penalizes moderate reuse distances very severely. As a result, IGDR does not have a good hit rate for moderate reuse distance accesses and performs 2.3% worse than SHiP. We discuss IGDR in more detail in Section 3.

An implementation of PRP (hereafter called PRP-Full) using these two techniques improves performance by 6.0% and 4.0% and reduces off-chip traffic by 8.5% and 6.6% compared to equal-area implementations of DRRIP and SHiP, respectively, for memory-intensive benchmarks in SPEC-CPU2006 [2014]. These gains occur even though PRP-Full requires storage of 34 bits (7% overhead) with each resident cache line and 10.4 bits (2% overhead) with each non-resident line in DRAM. Additionally, PRP-Full requires a probability computation unit that adds 6pJ per miss (<0.1% of miss energy) and 0.5% to the LLC area. PRP-Full, by virtue of reducing off-chip traffic, also saves 3.3% and 2.1% full system energy over DRRIP and SHiP, respectively, despite overheads. A version of PRP using a sampling technique (called PRP-Sample), reduces the DRAM overhead to 0.4b (<0.1%) per line, with only 0.6% performance loss over PRP-Full.

We observe that an optimal replacement policy enables cache hits to blocks with reuse distances too large for current replacement policies to track. For example, the top part of Figure 1 shows the fraction of LLC accesses having different reuse distances in the

benchmark mcf. Here *reuse distance* is defined as the number of accesses to the set containing a cache block, not necessarily unique [Duong et al. 2012; Shen et al. 2007], between consecutive access to that cache block. Below we discuss the implications of using unique-references-based reuse distance. The bottom part of Figure 1 shows the fraction of accesses of different reuse distances that hit in the cache when using LRU, Belady's optimal cache replacement algorithm (OPT¹) [Belady 1966], DRRIP [Jaleel et al. 2010], SHiP [Wu et al. 2011], IGDR [Takagi and Hiraki 2004], PDP [Duong et al. 2012], and PRP replacement policies in a 4MB, 16-way associative LLC. From this figure, we observe that most of the accesses have a high reuse distance, which tend to be misses while using the LRU algorithm. DRRIP, SHiP, PDP, and IGDR all provide more hits than LRU for higher reuse distances. However, OPT has a much higher hit rate in these reuse distance bins than LRU, DRRIP, SHiP or PDP, and PRP outperforms all other policies in these bins (e.g., in the reuse bin 32-63, hit rate for PDP is 31%, for SHiP is 50%, and for PRP is 75%, while hit rate for OPT is 86%).

We also experimented with using unique reference-based reuse distance (LRU stack distance) for PRP. Using this metric has only 0.2% better performance than PRP. In this article, we focus on non-unique reference-based reuse distance because it can be computed in hardware using only counters, whereas computing LRU stack distance is complex. Prior work [Keramidas et al. 2007; Takagi and Hiraki 2004] has successfully used this definition of reuse distance for performing cache replacement.

Maintaining a coarse-grained reuse distance distribution improves cache replacement by enabling the policy to *discriminate* between moderate reuse distance lines and very long reuse distance lines. This enables the last-level cache to maintain a portion of a working set with moderate reuse distance in cache protecting it from a working set with a long reuse distance.

The contributions of this article are as follows:

- It argues for using probability of a hit instead of expected reuse distance as the principle of optimality for causal replacement algorithms.
- It observes that improving cache replacement requires maintaining enough information to distinguish reuse distances larger than the associativity of the LLC.
- It introduces a novel cache replacement algorithm, PRP, that employs detailed reuse distance distributions and metadata for non-resident lines.
- It introduces an optimized reuse distance histogram representation and a sampled tagstore approach to reduce the cost of off-chip and on-chip metadata storage.
- It shows PRP significantly reduces LLC misses across a wide range of workloads.

The rest of the article is organized as follows. We provide more background and motivation behind PRP in Section 2. PRP is described in Section 3. Then, the implementation details are given in Section 4, followed by examples of access patterns which PRP handles better in Section 5. Then, simulation methodology and results are described in Section 6 and Section 7. Related work is discussed in Section 8 before concluding.

2. MOTIVATION

This section looks more closely at the motivation for employing reuse distance distributions for replacement decisions.

Belady [1966] proposed an optimal replacement algorithm under the assumption that the future reference stream is known. In this case, the optimal replacement candidate is the line which is referenced furthest in the future. Denning et al. [1968] distinguish

¹Called MIN in Belady's paper. We use a modified version of MIN, MIN+Bypass that does not insert the line into the LLC if it is re-referenced after all the replacement candidates.

Fig. 2. Dominant line reuse distribution profiles for mcf.

optimal replacement algorithms based on whether they require future information that is unknown, which they call “unrealizable,” versus “realizable” optimal algorithms that make the best possible replacement decision given a statistical model that is assumed to accurately reflect future program behavior. They propose the independent reference model (IRM) of program behavior in which at each time the probability of accessing a block i is given by stationary probability λ_i . They argue evicting the block j with maximum *expected* reuse distance $\frac{1}{\lambda_j}$, a policy they call A_0 . Aho et al. [1971] provide a formal proof of the optimality of A_0 under the independent reference model.

A cache line with reuse given by the independent reference model tends to have a geometric (i.e., exponential) reuse distance distribution. In practice, the access sequence observed at the LLC for individual lines does not follow this model. For example, Figure 2 illustrates five dominant reuse distributions for individual cache lines from the SPEC-CPU2006 mcf benchmark. The reuse profile for each individual memory block was found by profiling then clustered using K-means. OS interference is omitted here so we can focus on application level behavior. The five bar charts on the left plot access frequency (y-axis) versus reuse distance (x-axis). The bar charts on the right show relative access frequencies to the different line distributions broken down into hits and misses when employing DRRIP and PRP on the right. Most lines have reuse distance profiles that are multimodal.

Evicting the line with maximum expected reuse distance can lead to poor replacement decisions when lines have multimodal reuse distributions. Consider a fully associative cache with capacity of 16 blocks and two replacement candidates A and B . Block A is predicted to be accessed 1,024 references in the future with probability $P = 1$. Block B , on the other hand, is predicted to be accessed either 8 reference in the future with $P = 0.5$ or 8,192 references in the future with $P = 0.5$. Block B has the higher expected reuse distance, 4,100 vs. 1,024 for A . However, it is better to replace block A because it is almost certain to be evicted before it is reused 1,024 references in the future. Block B on the other hand has a 50% chance of being hit after just 8 references. In this example, replacing the block with largest expected reuse distance leads to a poor replacement decision.

If employing the expected reuse distance to select the replacement candidate can lead to poor choices of which block to evict, the question arises as to “what is a better alternative?” Before introducing OPT, Belady [1966] informally argued, “to minimize the number of replacements, we attempt to first replace those blocks that have the lowest probability of being used again.” PRP builds upon this notion of using probability. Below we sketch a brief theoretical argument for replacing the block with minimum estimated probability of receiving a hit before eviction under OPT. Since future accesses are unknown, later we will assume recent reuse distance distributions are a good predictor of upcoming reuse distance statistics. We have observed that this probability-based metric gives a different replacement candidate than the expectation-based metric in a majority ($\approx 72\%$) of replacement decisions, implying that these two metrics are indeed significantly different.

Consider a reference stream consisting of accesses to lines $(S_1, S_2, S_3, \dots)$. Consider a single set of the cache, and assume at time t , the current contents of the set are the lines $(x_1, x_2, x_3, \dots, x_W)$, where W is the associativity of the cache. Let us denote the evicted line at time t by x_t^e (where e refers to “evicted”). We compare a particular policy F to OPT, the optimal replacement policy assuming the future reference stream is known. A difference between the miss rates of OPT and policy F happens for two reasons: (a) references hit in the OPT policy but miss in the policy F , and (b) references miss in the OPT policy but hit in the policy F . For “reasonable” replacement policies, Figure 1 shows that there are relatively few references of type (b). Hence, we focus only on the number of references that miss in policy F but hit in the OPT policy. We denote this number by Δ_F . If a reference S_i hits in OPT but misses in F , F must have evicted that line earlier. We define an indicator random variable $I_{x_t^e}$ that is 1 if the evicted line x_t^e under F would receive a hit under OPT, and 0 otherwise. The outcome of this random variable depends on the actual future reference sequence that we assume is drawn from some (unspecified) probability distribution. Then,

$$\Delta_F = \sum_t I_{x_t^e} \quad (1)$$

Taking expectation of the expressions on both sides and using the linearity of expectation, we get

$$E[\Delta_F] = \sum_t P_{x_t^e}(\text{hit}), \quad (2)$$

where $P_{x_t^e}(\text{hit})$ is the probability that x_t^e receives a hit before being evicted using OPT. Hence, we can minimize Δ_F , and thus miss rate, by replacing the line x that has the lowest $P_x(\text{hit})$, i.e., the lowest probability of hit under the OPT policy.

To employ the above approach, we require a practical approach to estimating the probability of a hit. The following section describes our approach.

3. PROBABILISTIC REPLACEMENT POLICY

A cache controller implementing PRP chooses a *victim* line from a set by selecting the candidate line L with the lowest P_L^{hit} , the probability that line L would receive a hit under optimal replacement given its current age. We condition over the current age to take into account the easy to obtain information available about the age of the line. The cache controller computes P_L^{hit} by estimating the following distributions:

- (1) The *line distribution* $P_L(t)$: the probability that the next reuse distance for line L will be t , and
- (2) the *cache distribution* $P^{\text{hit}}(t)$: the probability any line with reuse distance t would receive a hit under OPT.

Using these quantities, the hit probability is estimated as:

$$P_L^{hit} = \frac{\sum_{t > T_L} P_L(t) P^{hit}(t)}{\sum_{t > T_L} P_L(t)}. \quad (3)$$

Here T_L is the age of line L . The sum is over reuse distances t greater than T_L because the next reuse distance will be greater than the line's current age. A similar formula was used by Takagi and Hiraki's [2004] Inter-Reference Gap Distribution Replacement (IGDR) policy to compute a "weight" used to select a victim line, except that instead of using $P^{hit}(t)$, they use $\frac{1}{t}$. However, the $\frac{1}{t}$ weighting decreases much faster than the $P^{hit}(t)$ weighting. Thus, IGDR does not give sufficient "importance" to medium reuse distance accesses, which is the only class of accesses for which there is a significant gap between hit rates of OPT and LRU. For example, it can be seen that the probability of hit under OPT for the lines with reuse distance between 32 and 63 is $\approx 85\%$, while IGDR weighs such accesses by only 3%. As a result, IGDR does not improve hit rates significantly for these accesses, and actually performs 2.3% worse than SHiP.

In Equation (3), $P_L(t)$ is dependent on the line, while $P^{hit}(t)$ is independent of the line. Thus, a representation of $P_L(t)$ is stored for each line in the cache, but only one copy of $P^{hit}(t)$ is maintained. Section 4 discusses an implementation of these distributions that consumes only 24b.

3.1. Estimating the Line Distribution

If we assume the next reuse distance for a line is independent of the prior reuse distance for the same line, we can estimate $P_L(t)$ by recording the frequency $N_L(i)$ with which reuse distance i is observed for each line L . The line distribution $P_L(t)$ is then estimated using

$$P_L(t) = \frac{N_L(t)}{\sum_i N_L(i)}. \quad (4)$$

If reuse distance t is binned into K bins, then for each line, K counts, one for each bin, must be stored. We use only 4b per count, for a total of 24b for the entire histogram.

3.2. Estimating the Cache Distribution

To estimate the cache distribution $P^{hit}(t)$, we use the average hit rate of OPT for each reuse distance t over the SPEC-CPU2006 suite. The exact distribution we use in our evaluations is described in Section 4.5. In Section 7.1.4, we show that PRP works well for a wide range of synthetic cache distributions.

4. IMPLEMENTATION

Figure 3 illustrates our implementation of PRP, highlighting a single set of a 2-way LLC. On an LLC miss to a line, the line is fetched from DRAM and in parallel a victim is selected. To select a victim an array of hit probability calculators, (❶) computes $P_{L_i}^{hit}$ for each candidate line L_i using its age T_{L_i} and reuse distribution $N_{L_i}(t)$. The candidate with the lowest probability of hit is evicted to make space for the incoming line. The reuse distribution for the incoming line is initialized with the reuse profile $N_L(t)$ that was stored alongside the page table translations in the TLB (❷) and brought to the LLC alongside the memory request that initiated the LLC access (❸). Below each component is described in detail along with the representation of line timestamps, reuse distance bins, and reuse distance frequencies.

Fig. 3. Overall hardware for PRP, $N_L(t)$ is the frequency of the reuse distance t , and M_L is the last access timestamp for line L . Reuse distance distribution, $N_L(t)$, of lines are stored in the cache as metadata, and are used to make replacement decisions by evicting the line with minimum hit probability, P_L^{hit} (Section 4) [1, 2, 3]. $N_L(t)$ is stored in a separate DRAM area, and fetched with TLB misses (Section 4.4) [4, 5, 6, 7]. Sampled Tagstore is used to eliminate fetching timestamps (Section 4.4) [8, 9].

4.1. Representing Reuse Distance Histograms

To minimize space, we use a logarithmic spacing of reuse-distance histogram bins focused on the range where hit rate varies with reuse distance. We group reuse distances into H histogram bins (we found $H = 6$ was sufficient). Bin 0 records reuse distances in the interval $[1, W)$, where W is the way size of the cache. Bins $i = 1, 2, \dots, H - 2$ record reuse distances that fall in the interval $[W\alpha^{i-1}, W\alpha^i)$, where α is a constant (we found $\alpha = 2$ works well). The last bin ($i = H - 1$) records reuse distances in the range $[W\alpha^{H-2}, \infty)$, i.e., all reuse distances $\geq \alpha^{H-2}$ times the way size. In our evaluation using a 4MB, 16-way associative cache with 64B lines, the intervals are $[1, 15]$, $[16, 31]$, $\dots$, $[256, \infty)$. For the optimal policy, we observed the hit rates for accesses with reuse distance $\geq 16W$ is almost 0, whereas for reuse distance $\leq W$, it is always 1. We show in Section 7.1 that having more bins is unnecessary. Such a logarithmic bin size was also used by Keramidas et al. [2007] for reuse distance prediction. A more detailed comparison of this work with theirs is given in Section 8.1.3.

4.2. Encoding Reuse Distance Frequencies

We store a reuse distance profile $N_L(t)$ with each line L in the cache. This profile has one entry for each of the bins described above. To represent an indefinite count with a small, finite-precision counter, we halve all of the counter values for one line's histogram whenever any counter in the line overflows. For example, suppose the counter precision is 4 bits and the current counter values for the different reuse bins are $[7, 9, 2, 10, 15, 8]$. Once an access with reuse distance in interval 4 is observed, the counter value overflows leading to the halving of all the counter values. Thus, the new counter values are $[3, 4, 1, 5, 8, 4]$. This method has the added benefit that it weights recent references more heavily than older references allowing the distribution to adapt more quickly to non-stationary behavior. In our implementation, we use a 4-bit precision for all the counters. We show in Section 7.1 that this does not lead to any significant degradation in performance.

4.3. Computing Reuse Distance

In order to obtain the reuse distance profile, we need an online method to compute the reuse distance of accesses. In this section, we outline two approaches for computing reuse distance of cache accesses.

4.3.1. Timestamps in DRAM. In this approach, we keep a count M of accesses to each set of the LLC and a timestamp M_L for each line L . The age of a line, T_L is computed as $T_L = M - M_L$. When a line is reused, we increment the histogram bin $N_L(T_L)$ associated with its age and reset its timestamp to the current count $M_L = M$. To save space, we encode timestamps in units of $W/2$, half the cache way size (i.e., for our 16-way cache, we discard the low 3 bits of M when recording a timestamp M_L). Aliasing occurs if the reuse distance is greater than the range of the timestamp. However, the effect of this aliasing is small, because the geometric bin sizing aliased timestamps tend to fall in the $>16W$ bin. In practice, we found using a 10-bit timestamp was sufficient. With increasing cache associativities, the timestamp storage grows only as $\log(W)$. Thus, even with higher associative caches, the timestamp storage does not get significantly worse (e.g., a 32-way associative cache will need only one extra timestamp bit, resulting in an 11b timestamp compared to the 10b one in our implementation).

4.3.2. Sampled Tagstore. Although storing timestamps in DRAM leads to a conceptually simple solution, this approach results in significant DRAM storage and traffic overheads. In order to eliminate these overheads, we propose the Sampled Tagstore approach that does not require storage of any timestamps in DRAM and is conceptually similar to a scheme introduced by Stone [1993].

The Sampled Tagstore approach relies on two key observations: i) a cache of size C with a LRU replacement policy will only serve references of reuse distance $\leq C$, and, ii) the miss rates of caches remain approximately unchanged if the cache size is reduced by a factor of K and a randomly selected $1/K$ fraction of the original cache traffic is served from this downsized cache.

To see how the first observation can be leveraged to get the reuse distance bins of accesses, let us consider a scenario where there are two auxiliary tag arrays, one of size C and one of size $2C$, each maintained according to an LRU replacement scheme. All accesses are checked for hits in these two auxiliary tag arrays. If an access receives a hit in the smaller tag array, it can be inferred that the reuse distance of the access is $\leq C$. On the other hand, if the access hits in the larger tag array but not the smaller one, it can be inferred that the reuse distance is between C and $2C$. Finally, if the access misses in both the tag arrays, its reuse distance is probably $>2C$.

Extending this approach, we can see that to bin the reuse distance of cache accesses into the bins $(0, C]$, $(C, 2C]$, $\dots$, $(8C, 16C]$, $(16C, \infty]$, five tag arrays of sizes $C, 2C, \dots, 16C$ are required. Here C is the capacity of the LLC. However, these auxiliary tag arrays consume a significant amount of area, totaling to $31 \times$ of the LLC tag array area.

To reduce this area overhead, we use the second observation to downsize each of the tag arrays by a factor K , and send a random $1/K$ fraction of the cache traffic to the auxiliary tag arrays. Only the sampled accesses are used to build the reuse distance profile. In Section 7, we observe that a sampled tagstore with $K = 64$ has only 0.6% lower performance than storing timestamps in DRAM, while requiring an area overhead of $1/2$ of the actual tag array.

4.4. Efficiently Storing Reuse Histograms

We observed that the frequency vectors of adjacent lines in page are similar. We leveraged this observation to reduce the overhead of storing reuse distance frequency vectors by associating a single vector with a *profile block* of multiple consecutive lines. We found a profile block size consisted of 64 consecutive lines or 4KB equal to a page size works well. Section 7.1 shows the impact of profile block size. We found larger profile blocks tend to be better even when ignoring the bandwidth overhead savings.

Reuse distance histograms are collected online as an application runs and stored adjacent to the page translation in the TLB (② in Figure 3). Upon a TLB eviction,

Table I. Cache Distribution

t bin	0–15	16–31	32–63	64–127	128–255	256– ∞
$2^4 \cdot P^{hit}$	15	14	12	10	9	1

the histogram is stored in memory in a structure parallel to the page table called the *PRP Metadata Table*. This PRP Metadata Table is cached in the LLC, and to avoid recursively fetching frequency vectors for the lines holding this data, we assign them a uniform $N_L(t)$. On a TLB access that misses (4), the PRP metadata is loaded into the TLB (5) alongside the page translation (6). After accessing the LLC and computing T_L , the reuse histogram in the TLB is updated together with the response to the original memory request (7).

When using the sampled tagstore approach, a random sample of the cache accesses is looked up in the sampled tag arrays (8). The reuse distance bin of a sampled access is obtained as the minimum size of the tag array that causes a hit for that access. The reuse distance profile in the TLB is then updated with this information (9).

Note that since the PRP profile block size is an independent design parameter, it is possible to have different sizes for the profile block and physical page. This distinction might be useful in the case of large pages (2MB), where the behavior of all lines in the page might not be similar. In such cases, the PRP metadata needs to be maintained in a separate hardware structure apart from the TLB. In this case, the timestamps (M_L) in the DRAM or the LLC are not required.

The metadata associated with each page is as follows. Each page contains a frequency vector $N_L(t)$ for the page. This frequency vector is of length 24 bits (6 bins $\times$ 4 bits per frequency). Also, a last access timestamp of 10b length needs to be stored for each line in the DRAM. Thus, the total DRAM storage overhead is $10b + 24/64b \approx 10.4b$ per line. With the sampled tagstore technique, the timestamps need not be stored, and thus the overhead is reduced to 24b per page, or 0.4b per line. Note that although small in size, the size of the PRP metadata is bigger than the free space in a x86-64 PTE. Thus, this data has to be stored in a separate table and cannot be incorporated into the PTE itself.

4.4.1. Coherence of PRP Metadata. Note that PRP metadata coherence is not a correctness concern. In our implementation, we do not require any extra coherence mechanism for PRP metadata beyond what is already guaranteed by cache coherency. The PRP timestamps are maintained in the LLC and not the cores. In the case of multiple writes of PRP metadata by different cores, the cache coherence protocol ensures only one copy is written back. Scenarios such as TLB shutdown can lead to stale PRP metadata being present in the DRAM, but this is not a correctness issue.

PRP adapts to a changing line distribution (the probability of a line having different reuse distances) by halving the counters upon overflow, as described in Section 4.2. As a result of this scheme, any stale line distribution vector will be replaced by a fresh one after it is accessed at most $2^K H$ times, where K is the precision of the counters, and H is the number of bins. As we use small values for K and H (4 and 6, respectively), a stale distribution does not happen often in the studied benchmarks.

4.5. Cache Distribution

The Cache Distribution, $P^{hit}(t)$, is the probability that a line of reuse distance t will hit in the cache. This distribution is not dependent on the line. As described in Section 3.2, we use a fixed cache distribution, which is the average hit rate of the OPT policy in the selected reuse bins using the training input set. These probabilities are also quantized to 4 bits like the line distribution. For a 16-way, 4MB cache, we use the probabilities in Table I. Storing the cache distribution only requires six 4-bit registers, for a total of 24b.

Fig. 4. Probability calculator unit.

4.6. Probability Calculator Unit

Given the line distribution of a particular line, the hit probability calculator unit (Calc; in Figure 3) calculates the hit probability using Equation (3). The schematic of this unit is shown in Figure 4.

The unit utilizes the fact that Equation (3) can be rewritten as

$$P_L^{hit} = \frac{\sum_{t > T_L} N_L(t) P^{hit}(t)}{\sum_{t > T_L} N_L(t)}. \quad (5)$$

Here, $N_L(t)$ is the frequency of occurrence of reuse distance t for line L . Thus, first the frequencies of all the bins $<$ the current age bin, T_L are zeroed out. Then a dot product is done between this truncated frequency vector and the cache distribution. The dot product is then divided by the sum of the elements in the frequency vector to get P_L^{hit} .

All arithmetic is low precision, so the energy consumed is much lower than a memory access. The energy of these operations is included in our evaluation.

5. PRP EXAMPLE

This section considers an example to provide insight into PRP.

5.1. Distinguishing Reuse Distances

Modern replacement policies try to protect short reuse distance accesses from scan patterns induced by long reuse distance accesses. However, we find such policies are not able to protect *moderate* reuse distance accesses from long reuse distance accesses. An example access pattern to a single set of a 4-way LLC from the benchmark mcf is shown in Figure 5. Accesses to $A_1, A_2, \dots$ are made by instruction IA and have moderate reuse distance of 80. On the other hand, accesses to $S_1, S_2, \dots$ are made by the instruction IS and have a long reuse distance of 1,000. This pattern is created by two scans performed by two loops inside the function `price_out_impl()` in the file `implicit.c`, which is called repeatedly from `global_opt()` in the file `mcf.c`. The working set of the first loop (lines 263–264 in `implicit.c`) fits in the cache, whereas that of the second loop (especially line 269) is larger.

The middle columns of Figure 5 show the behaviors of two state-of-the-art replacement policies, DRRIP [Jaleel et al. 2010] and SHiP [Wu et al. 2011], on this pattern.

DRRIP was introduced by Jaleel et al. [2010] who start by considering the “LRU chain” used for LRU replacement as providing predictions on re-reference intervals. They point out the LRU chain makes poor predictions for workloads that contain scans and thrashing interspersed with LRU-friendly access patterns. To avoid evicting useful data, they propose several re-reference interval prediction (RRIP) mechanisms. Static RRIP (SRRIP) employs an M-bit re-reference interval prediction value (RRPV) to replace the metadata employed by pseudo-LRU algorithms. Missing references are

Fig. 5. Scan pattern handling for SRRIP, BRRIP, SHiP, and PRP.

initially inserted with an RRPV value encoding a “long” re-reference interval prediction. Under SRRIP-HP a hit changes the RRPV encoding to a “near-immediate” prediction. Upon a miss, the first block with a “distant” re-reference interval prediction is selected for eviction. Bimodal RRIP (BRRIP) inserts a small fraction ϵ of references with a “long” RRPV and the rest with a “distant” RRPV. The effect is to enable scan resistance. Dynamic RRIP (DRRIP) employs set-dueling [Qureshi et al. 2007] to select between SRRIP and BRRIP.

SHiP (Signature-based Hit Predictor), introduced by Wu et al. [2011], improves on DRRIP by predicting whether a line will hit using a predictor table. The table is trained by the hit/miss characteristics of a line under the SRRIP policy. They show that a predictor table indexed by the address of the instruction causing the miss works the best. The behavior of SHiP is shown in the third column of Figure 5.

For the access sequence in the example at the point when a replacement candidate is required, none of the lines $A_1, A_2, \dots$ have received any hits, so SRRIP is unable to distinguish between the lines belonging to the moderate vs. the long scan. On the other hand, scan resistant BRRIP chooses a small random fraction of a scan to retain in the cache. Since this selection is not dependent on the past behavior of the lines, only a small fraction of the moderate reuse distance scans are retained in the cache.

Fig. 6. Moderate reuse distance fractions and hit rates.

SHiP is trained by the hit/miss history of SRRIP. Since SRRIP itself does not cause any hits to A_i , the predictor table of SHiP is unable to learn a higher preference for A_i over S_i . Thus, SHiP is unable to cause hits to A_i as well.

PRP can distinguish between the long and moderate reuse distance lines because it stores the reuse distance histograms of the different lines. Thus, for PRP, S_2 evicts S_1 , S_3 evicts S_2 , and so on, while maintaining A_2 , A_3 , and so on in the cache.

To gauge the importance of moderate reuse distance lines, we collected the hit rates of DRRIP, IGDR, and PRP for moderate reuse distance accesses. The results are shown in Figure 6. Figure 6(a) shows the fraction of moderate reuse distance accesses for various benchmarks, and Figure 6(b) shows the hit rates of LRU, DRRIP, IGDR, SHiP, and PRP on these accesses. It can be observed that, on average, 38% of accesses are of moderate reuse distance. LRU achieves a hit rate of 1.4%, while DRRIP, IGDR, and SHiP achieve hit rates of 34%, 36%, and 38%, respectively, on this category of accesses. PRP is better than all the others and achieves a hit rate of 45% for moderate reuse distance accesses.

Necessity of storing information in DRAM: Policies such as DRRIP or DGIPPR only store information about lines that are present in the cache. The key insight is that such policies make a replacement decision based only on the *behavior of the line since last insertion*. We call this class of policies *non-discriminating*. A *discriminating* policy, on the other hand, stores metadata to differentiate between lines with the same behavior since last insertion. Thus, all variants of PRP are discriminating. We now argue why a discriminating policy is necessary to get hits to moderate reuse distance lines.

As can be observed in Figure 5, the lines $A_2, A_3, \dots$ have not yet received hits when the replacement candidate for S_2 needs to be obtained. Thus, non-discriminating policies such as DRRIP and DGIPPR cannot differentiate between S_1 and A_i , since they only look at the behavior of a line since last insertion. A discriminating policy such as PRP, on the other hand, can distinguish the A_i lines to be of moderate reuse, and thus replace the S_1 line instead correctly.

Fig. 7. Fraction of hits to lines evicted without hits in DRRIP.

To prove this point, we looked at lines which suffered two misses in a row in DRRIP, i.e., the line was evicted before receiving any hits. We then looked at the number of cases where the second miss was converted to a hit in PRP (Δ_{NH} , where NH stands for “No Hit”), and compared this number to the total number of misses reduced by PRP (Δ_{ALL}). Figure 7 plots the ratio Δ_{NH}/Δ_{ALL} for cases where PRP reduces the number of misses significantly. About 80% of the additional hits in PRP arise from the ability of PRP to know the reuse distribution of lines that have not received a hit yet. This is also a strong indication that cache replacement policies should be designed to be discriminating, i.e., take into account the past behavior of a cache line.

5.2. Evicting Dead Blocks Early

Since PRP computes the probability of hit *given the current age* of a line, blocks become available for replacement immediately after their “live period” has passed. On the other hand, PRP does not evict blocks that still have some possibility of receiving a hit at a moderately long reuse distance.

5.3. Effect of Accounting for Non-Temporality

Above we described PRP such that the incoming line is not bypassed [McFarling 1992] even if the victim line has a higher probability of hit than the incoming line. We observed that extending PRP to use bypassing based on computing P_L^{hit} for the incoming line provides only a 0.2% performance benefit over using PRP without bypass. The reason behind this phenomenon can be better understood from Figure 5. In this case, it can be observed that line S_1 evicts line A_1 even though A_1 has a higher hit probability. This results in having only 3 hits instead of 4 hits, which would have happened with bypassing. However, note that in the case of a 16-way cache, the total number of hits for a similar pattern would have decreased from 16 to 15, which is minor. Thus, as associativity increases, the benefits of using bypassing with PRP reduces.

6. METHODOLOGY

We use MARSSx86 [Patel et al. 2011], an x86-64 full system simulator to compare different cache policies. We compare PRP to DRRIP, SHiP, and PDP. For DRRIP, SHiP, and PDP, we used one extra way in the cache to compensate for the extra space being used for PRP and PRP-Sample64. These policies are denoted by DRRIP-17w, SHiP-17w, and PDP-17w. For PDP, we used the PDP-3 variant outlined by Duong et al. [2012]. We validated our implementation of DRRIP by comparing with the implementation of DRRIP provided with CMP\$im [Jaleel et al. 2008]. We also simulate WN1-4-DGIPPR for obtaining the energy consumption of a low overhead replacement policy. For all policies, we use parameters provided in the respective papers. The system parameters we use are shown in Table II. All line sizes are 64B. The extra traffic to DRAM and LLC for reading and writing PRP metadata is considered in our evaluations. Cacti [Muralimanohar et al. 2007] is used to compute cache access energies.

Table II. System Parameters

Timing parameters	
Core	4-way Out-Of-Order, 128 ROB entries, 2.4GHz
I/D TLB	64 entry, 8 way
L1 data cache	32 KB, 4 way, 4 cycle
L1 instruction cache	32 KB, 4 way, 4 cycle
L2 cache	256 KB, 8 way, 10 cycle
L3 cache	4 MB, 16 way, 30 cycle
DRAM latency	200 cycles
PRP parameters	
Reuse distance vector	6 bins, 4b precision
Timestamp precision	10b
Profile block size	4KB
Energy parameters	
Core energy / instruction	1nJ
L1(I/D) cache access	20pJ
L2 cache access	70pJ
L3 cache access	438pJ
DRAM access	25.6nJ
Timestamp access (all ways)	30pJ
Profile access (all ways)	40pJ

Table III. Hardware Storage Overheads

	DRRIP	SHiP	PDP (PDP-3)	PRP	PRP-Sample64
On-chip	16KB	142KB	24KB	277KB	270KB
DRAM	0B	0B	0B	162MB	6MB

We use SpecCPU2006 benchmarks to evaluate the various cache replacement policies. We use Pinpoints [Patil et al. 2004] to obtain up to 10 simpoints of length 500M instructions each which is representative of more than 90% of program execution. A subset of the SPEC benchmarks is simulated for which performance increases by at least 5% when LLC size is increased from 4MB to 8MB.

To evaluate the latency and energy costs of the computation involved in obtaining the hit probability, we synthesize the logic in 45nm TSMC bulk CMOS technology and obtain its latency and energy. The majority of this area was occupied by the 11-bit divider unit. One Probability Calculator unit takes 5 processor cycles to compute P_L^{hit} , and consumes 374fJ per operation. The area of an unit is equal to 0.00129mm^2 . Thus, 16 parallel probability calculator units, one for each way of the LLC, consume 6pJ to compute all the hit probabilities, and consume an area of 0.0207mm^2 , which is $<0.5\%$ of the LLC area. Maintaining the extra state in the TLB consumes an extra 5KB for PRP and 192B for PRP-Samp64. Reading this extra state is only required in the case of an L1 miss and thus is not in the critical path of TLB lookup. We assume that this read takes a single cycle and account for its latency and dynamic energy consumption in our simulations. Table III summarizes the on-chip and DRAM overheads for PRP, DRRIP, SHiP and PDP for a system with a 4MB, 16-way LLC and 8GB main memory.

7. RESULTS

In this section, we present the performance and energy results of the PRP policy. We evaluate PRP with FREQ policy (PRP), and PRP with $64\times$ sampled tagstore (PRP-Sample64). The performances of these policies on a memory-intensive subset of

(a) Performance of various replacement policies

(b) Memory traffic for various replacement policies

Fig. 8. Performance and memory traffic for various policies.

SPEC-CPU2006 are summarized in Figure 8(a). PRP has a performance advantage of 10.5% over the baseline LRU policy, as opposed to 4.5% for DRRIP-17w, 4.9% for PDP-17w, and 6.5% for SHiP-17w. PRP-Sample64 has a performance advantage of 9.9%. Thus, PRP performs 6.0% better than DRRIP-17w, 5.6% better than PDP-17w, and 4.0% better than SHiP-17w over the chosen workload set. PRP-Sample64, despite sampling only 1/64 of the accesses, performs only 0.6% worse than PRP. Over the entire SPEC-CPU2006 suite, PRP performs 2.6% better than DRRIP, 2.4% better than PDP, and 1.7% better than SHiP, with the worst performance degradation of 5.0% over DRRIP happening for gcc. None of the memory-insensitive workloads experience any significant slowdown with PRP.

Figure 8(b) shows the normalized DRAM accesses of DRRIP-17w, SHiP-17w, PDP-17w, PRP, PRP-Sample64, and OPT as compared to a baseline LRU policy. The traffic overhead introduced by reading various metadata from DRAM is shown separately. PRP decreases demand references by 17.3% over LRU, which is 9.0% better than DRRIP-17w, 9.7% better than PDP-17w, and 6.6% better than SHiP-17w. It can also be seen that, on average, PRP introduces a DRAM traffic overhead of 2.5%, which is significantly smaller than the traffic reduction achieved by PRP. PRP-Sample64 reduces this overhead to 0.8% by eliminating DRAM timestamp metadata. PRP-Sample64 also reduces the worst-case DRAM traffic overhead from 7% in the case of astar to 2%. Compared to OPT, PRP is worse by 17%.

An interesting case here is that of leslie3D. Although this benchmark receives double the number of hits in PRP as compared to DRRIP for moderate reuse distance accesses (as can be seen from Figure 6), it does not end up having significant performance advantage with PRP. The reason is that the baseline miss rate of leslie3D is very high ($\approx 80\%$), and thus the additional hits from PRP do not make a large difference in the overall number of misses. As a result, the performance is also not significantly affected.

We also obtained the full system dynamic energy savings of various PRP variants as well as DRRIP, SHiP and DGIPPR over LRU. The system energy includes dynamic

Table IV. Performance Relative to LRU of Various Replacement Schemes with Varying Cache Size

	DRRIP	SHiP	PRP	PRP-Sample64
2MB	4.5%	4.6%	9.0%	8.5%
8MB	3.1%	3.2%	10.7%	9.5%

Fig. 9. Sensitivity to various parameters of PRP.

and static energy consumption by the core, caches and DRAM. PRP saves 4.5% of the full system energy, while PRP-Sample64 saves 4.2% of the full system energy. DGIPPR, DRRIP, and SHiP save 1.2%, 1.9%, and 2.4% of the full system energy. Thus, PRP, despite having metadata overheads, saves 2.1% and 3.3% full system energy over SHiP and DRRIP, respectively. Taking into account only the memory system (caches + DRAM), PRP saves 3.4% and 4.2% energy over SHiP and DRRIP respectively.

The performance of PRP, SHiP, DRRIP and LRU for various other cache sizes and associativities were also obtained. We observed that changing the associativity to 8-way or 32-way does not change the average performance of any of the schemes by more than 0.2%. Table IV shows the average performance of the schemes relative to LRU for a 2MB and an 8MB cache. It can be seen that PRP has 4.4% performance advantage over SHiP for a system with a 2MB LLC, and a 7.5% advantage over SHiP for a system with a 8MB LLC.

7.1. Sensitivity of Design Parameters

Below we study the sensitivity of PRP design parameters.

7.1.1. Sensitivity to Profile Block Size. Figure 9(a) shows the sensitivity of PRP to the size of profile block, i.e., the group of lines whose frequency vectors are accumulated together. The performance of PRP increases as the size of the profile block increases. This is because larger profile blocks collect reuse distances of more lines and get trained faster.

7.1.2. Sensitivity to Frequency Vector Precision. Figure 9(b) shows the performance of PRP for various precisions of the frequency vector. It can be observed that PRP with frequency vector precision of 4 bits gives almost same performance as higher precisions. Also, even with 1-bit reuse frequencies, PRP is able to achieve 5.3% performance gain over LRU, which is similar to DRRIP.

7.1.3. Sensitivity to Tagstore Sampling Ratio. We used various sampling ratios for the Sampled Tagstore approach outlined in Section 4.3.2. We observed that while a $64\times$ sampling is only 0.6% worse than PRP, a $128\times$ sampling is 1.6% worse than PRP. Due to this drop in performance, we have used $64\times$ sampling in this article.

7.1.4. Sensitivity to OPT Cache Distribution $P^{hit}(t)$. To obtain the sensitivity of PRP to the OPT hit rate distribution, we also evaluate PRP using other empirical distributions, generated as follows. The P^{hit} value for Bins 0 and 5 are fixed at 15/16 and 1/16,

respectively. The P^{hit} value for Bin i , where $0 < i < 5$, is set to be $15/16 - Ki$. The performance of PRP with varying K is shown in Figure 9(d).

It can be observed that these empirical distributions do not perform significantly worse than PRP, and the performance is within 0.5% of PRP for all values of K . This happens due to the following reason. As shown in the example in Figure 5, PRP works by “distinguishing” moderate reuse distance lines from high reuse distance lines, and retaining the moderate reuse distance lines in the cache. For any gently decreasing P^{hit} , the probability of hit of a line with a long reuse distance line (such as S_i in Figure 5) will be lower than the probability of hit of a moderate reuse distance line (such as A_i in Figure 5). Thus, any monotonically decreasing OPT distribution works almost equally well. Such empirical distributions may be used in situations where obtaining the actual OPT hit rate distribution is cumbersome.

7.1.5. Sensitivity to Frequency Vector Binning. We varied α , the reuse distance bin size multiplier, to evaluate the sensitivity of PRP to the reuse vector representation. The total number of bins were appropriately scaled to cover the span from W to $16W$, W being the cache associativity. The results are shown in Figure 9(c). We can see that any α within the range 1.5–2.5 performs similarly as PRP. However, outside that range the performance of PRP degrades. With a higher value of α , the number of bins become too few to accurately discriminate between various reuse distances. On the other hand, with lower values of α , the reuse distance frequencies are distributed over a larger number of bins and thus become more noisy.

7.2. Performance Difference Between OPT and PRP

From Figure 8(b), it can be seen that OPT has 17% lower cache misses, on average, as compared to PRP. There are two major reasons why PRP falls short of OPT: (i) access sequences in benchmarks are unpredictable, and as such it is only possible for PRP to know a probability distribution of the reuse distance of the next access to a line, and (ii) inaccuracies stemming from binning the reuse distances into only six bins. To gauge the relative importance of these two factors, we ran an experiment where the *quantized future reuse distance* of lines was used to obtain the victim line instead of the future reuse distance as used by OPT. We observed that this quantization process led to a <1% increase in miss rate over OPT. Thus, it can be inferred that the unpredictability of access sequences is the major reason behind the gap between PRP and OPT.

Since PRP utilizes a probability distribution of reuse distance, it does not perform so well when the reuse distance of a line falls in a less frequent bin. For example, consider the line reuse distributions shown in Figure 2. Let us consider two lines: (1) line A belonging to type A and (2) line E belonging to type E, as shown in the figure. In a majority of cases, the reuse distance of line A is <16, while the reuse distance of line E is >16, so PRP correctly chooses to evict E over A. However, when the reuse distance of line A is 128 and the reuse distance of line E is <16, the PRP policy makes a mistake by evicting line E instead of line A. The OPT policy, by virtue of its knowledge about the complete access sequence, makes no such mistakes.

7.3. Effect of Considering Local Reuse History

While designing PRP, we have assumed that the next reuse distance of a line is *independent* of its previous reuse distance, but belong to the same distribution. This assumption is contrary to policies such as SRRIP [Jaleel et al. 2010], in which lines that receive a hit are predicted to receive hits again, i.e., the past reuse behavior is used to predict the future reuse behavior. To assess the effects of this assumption on the performance of PRP, we also experimented with a variant of PRP, *PRP-Cond*. PRP-Cond assumes that the next reuse distance of a line is dependent on the reuse

Fig. 10. S-curve for system throughput of PRP.

distance observed right before and maintains separate reuse distance distributions for all possible bins that the previous reuse distance can belong to.

We observed that PRP-Cond does not affect performance significantly and results in $\approx 1\%$ speedup over PRP. The benchmarks that benefit the most with PRP-Cond (soplex and sphinx) have repeating reuse distance patterns, which can be leveraged by PRP-Cond to get better estimates of the next reuse distance, resulting in better replacement decisions. In other cases, the next reuse distance of a line does not depend significantly on the previous reuse distance, and PRP-Cond performs similar to PRP.

Also, note that while PRP does not use reuse distance history, it does use the *last access time* of a line (T_L in Figure 3) to focus on reuse distances greater than its current access distance ($T - T_L$). As a result, PRP with the same distribution for all lines is similar to LRU. PRP improves on LRU by storing individual line distributions.

7.4. Performance for Multiprogrammed Workloads

To evaluate the effectiveness of PRP in a multiprocessor environment, we also simulated PRP for 51 random workload mixes in a 4 processor system with a shared 4MB LLC. Each mix was created by first choosing 4 random benchmarks, and then choosing a single random simpoint from each of the chosen benchmarks.

Figure 10 shows the s-curve of the system throughput metric [Eyerman and Eeckhout 2008] of PRP with respect to TA-DRRIP [Jaleel et al. 2010]. The average throughput improvement of SHiP over TA-DRRIP is also shown. It can be seen that PRP improves the system throughput by a maximum of 26.9% over TA-DRRIP and degrades the throughput by at most 4.5%. A throughput degradation is observed in 2 out of the 51 simulated cases. On average, PRP improves system throughput by 9.1%, while SHiP improves system throughput by only 2.9%.

The two workloads that experienced a slowdown both contained gcc, for which PRP degrades performance over DRRIP. PRP outperforms TA-DRRIP and SHiP when there are workloads with a significant number of moderate reuse distance accesses. TA-DRRIP considers the per-benchmark variation in re-reference interval values to insert lines from different benchmarks with different priorities. SHiP considers the variation in re-reference interval at a per-load granularity and is thereby able to gain 2.9% higher throughput than TA-DRRIP. However, as shown in Section 5, because SHiP is trained by hit/miss behavior under SRRIP, it is not able to reduce misses for instructions that always miss under SRRIP. PRP, on the other hand, considers within-benchmark variation in the reuse distance of various lines. In this way, PRP is able to preferentially evict all long reuse distance lines, while accommodating most of the moderate reuse distance lines from the programs.

8. RELATED WORK

8.1. LLC Replacement Policies

8.1.1. Scan and Thrash Resistance. A scan is a sequence of accesses that does not repeat. A sequence is said to be thrashing when it is repeated but the sequence is larger

than the cache. PRP helps improve LLC performance by addressing both patterns. Qureshi et al. [2007] propose a dynamic insertion policy (DIP) that improves performance by selectively inserting lines into the least instead of most recently used position in the recency stack. DIP adaptively selects a bimodal insertion policy (BIP) or traditional LRU. BIP improves hit rate for thrashing workloads by inserting most lines into the least recently used position and hence retaining some of the working set. BIP adapts to changing workloads by occasionally inserting some lines into the most recently used position. Rather than finding the right blocks to keep by statistically sampling, PRP tracks reuse distances, which helps it identify lines that will be reused.

Wu et al. [2011] improve on DRRIP by introducing the Signature-based Hit Predictor (SHiP) policy. SHiP associates a signature with each line and uses a table to predict whether lines with a given signature receive a hit or not. Lines that are not predicted to receive a hit are inserted with a distant RRPV and are, therefore, preferentially evicted. Unlike DRRIP, which considers any incoming line to have a long RRPV, SHiP predicts which lines are likely to not receive any hits and improve cache hit rate by replacing such lines aggressively. They propose a memory region-based signature (SHiP-Mem), a load instruction-based signature (SHiP-PC), and a signature based on the sequence of instructions prior to the load (SHiP-ISeq). They find the SHiP-PC policy to be the most performant. Similar to SHiP-Mem, PRP also stores metadata at a memory region granularity to disambiguate between lines based on their access history. However, unlike SHiP, whose prediction is based on the hit history of a line under SRRIP, PRP distinguishes between moderate and high reuse distance lines by storing a reuse distance profile. A key difference shown in Section 5 is that by storing reuse distance distributions PRP can preferentially retain even those lines that have not observed any hits but are likely to have a moderate reuse distance.

Jiménez [2013] introduced Dynamic Genetic Insertion and Promotion for PseudoLRU Replacement (DGIPPR). Building on DRRIP and other works, Jiménez reinterprets the positions in the LRU-stack. The position a block moves to upon initial insertion or a subsequent hit is governed by a generic insertion/promotion vector (IPV) that indicates the next location to move to upon a subsequent reference. To reduce storage overhead, the approach is then applied to a Pseudo-LRU encoding. While policies like DRRIP and DGIPPR are designed to tolerate scans and thrashing, as noted in Section 5, they lack reliable information about a block when it is first inserted.

8.1.2. Optimal Replacement. Rajan and Ramaswamy [2007] propose the Shepherd Cache that attempts to emulate optimal replacement for a subset of ways in the cache by using the remaining shepherd ways to “look ahead” in the access stream. This look-ahead distance is limited by the size of the shepherd cache ways to being much smaller than the reuse distances that PRP can consider.

8.1.3. Reuse Distance and Distributions. Takagi and Hiraki [2004] propose Inter-Reference Gap Distribution Replacement (IGDR). Like PRP, IGDR tracks reuse distance for sets of lines and retains reuse information for lines that are not in the cache. Rather than grouping lines based on pages, IGDR categorizes each line into one of five generic reuse classes then maintains reuse distributions for each class. The class of a line is determined based on the number of references a line has received as well as their regularity. These distributions are used along with the time of last reference to compute a weight for each replacement candidate.

A significant difference with PRP is in how IGDR weighs moderate reuse distance accesses. IGDR adopts a $\frac{1}{t}$ weighting strategy, which decreases very rapidly, and thus does not result in significant hits for moderate reuse distance lines. PRP, on the other

hand, introduces the notion of using hit probability to make replacements. This scheme weighs the accesses by $P^{hit}(t)$, which decreases slowly, and thus increases the number of hits to moderate reuse distance lines significantly, resulting in 6.3% performance gain over IGDR over the chosen set of memory-intensive SPEC-CPU2006 workloads. Another significant practical difference with PRP is the size of the histograms. IGDR maintains histograms with 256 uniformly spaced bins versus PRP's 6 geometrically sized bins. This difference means that where PRP can compute hit probabilities within a few cycles, IGDR takes much longer to compute the weights that are stored in a table for each class and periodically updated in the background.

Keramidas et al. [2007] propose a reuse distance prediction-based mechanism for doing cache replacement. The authors predict the next reuse distance of an access based on the reuse distance patterns observed by the PC that last touched the line. This work also uses a $\log_2$ -based reuse distance bucketing similar to what we propose. However, since the reuse distance prediction has a confidence associated with it, it falls back to LRU when the confidence is low. This can lead to problems for workloads where the reuse distances are not predictable. Since PRP works on the basis of probability distribution of reuse distances, it does not suffer from this problem.

Duong et al. [2012] employ online profiling of an application's overall reuse distance distribution to determine a protecting distance value used for determining cache replacement decisions. Each line contains a counter that is set to the protecting distance value on insertion. Each access to a cache set decrements the counters for each line in the set until they saturate at zero. Only lines with a protecting distance value of zero are eligible for replacement. Since PDP computes only a single protecting distance, a longer PD might leave dead blocks for too long in the cache, whereas a too short PD will lead to useful lines being evicted. PRP, by storing a distribution for every line, avoids this problem.

Das et al. [2015] propose Sub-Level Insertion Policy (SLIP), which uses line reuse distance distributions to reduce LLC dynamic energy. Though in this work we use a similar reuse distance distribution encoding, the primary focus of this work is to reduce LLC misses unlike SLIP.

8.1.4. Shadow Tag-Based Replacement. Stone [1993] proposes a cache replacement scheme using a shadow tag array to observe which lines receive a hit in a twice large cache, and preferentially evict lines that do not receive a hit in the shadow tags. Such a scheme gives preference to lines whose reuse distance lies in the range $(C, 2C]$, where C is the cache capacity. PRP, on the other hand, observes the reuse distance history of lines up to $16C$, enabling it to serve more accesses with high reuse distances.

8.1.5. Dead Block Prediction. A significant amount of work has been done toward predicting when blocks become *dead*, i.e., when they will receive no hits in the future [Lai et al. 2001; Liu et al. 2008; Lin and Reinhardt 2002; Khan et al. 2010]. Lai et al. [2001] proposed a dead block predictor that uses the last PC that touched a block to predict when a block becomes dead. Liu et al. [2008] propose CacheBurst, which uses number of *bursts* instead of number of raw references to a cache block for predicting dead blocks. Khan et al. [2010] use a sample of the LLC accesses to train a smaller and more accurate dead block predictor based on program PC.

These works declare a block to be dead when it does not receive any hits under the LRU replacement policy. However, as shown in Section 5, even blocks with very high reuse distance that will certainly be evicted under the LRU policy can receive hits under the OPT policy. In this aspect, the closest work to PRP is by Lin and Reinhardt [2002]. In this work, the authors try to predict when a block becomes dead under the OPT policy by running a collected address trace through a OPT policy simulator and detecting the last-touch PC's. Since OPT is an offline policy, this method has a

significant overhead of profiling an application. PRP, on the other hand, characterizes OPT only by the distribution P^{hit} , which can then be used for all programs.

8.2. Metadata for Evicted Blocks

Several virtual memory and database buffer replacement policies retain metadata for nonresident pages or blocks to improve replacement decisions. Examples include EELRU [Smaragdakis et al. 1999], LRU-K [O’Neil et al. 1993], and ARC [Megiddo and Modha 2003]. One challenge to adopting this practice for cache replacement is the additional storage and bandwidth costs implied. PRP mitigates these overheads by storing reuse distributions at page granularity.

9. CONCLUSION

In this article, we introduce probabilistic replacement policy (PRP), a novel LLC replacement policy. On a miss PRP estimates the probability each block in the cache set would receive a hit under optimal replacement if it were to be retained and then evicts the block with lowest hit probability. We argue using a probability calculation is more robust under the varying reuse distance intervals observed at the LLC. To implement this calculation efficiently and with low complexity, we propose several effective optimizations. Reuse distances are tracked at page granularity using low precision counters for a small number of geometrically spaced bins. To reduce off-chip storage costs reuse distances are obtained by several auxiliary tag arrays. PRP outperforms SHiP, a state-of-the-art LLC replacement algorithm, by 4.0% and reduces LLC misses by 6.6% and naturally adapts to multiprogrammed workloads.

REFERENCES

- Alfred V. Aho, Peter J. Denning, and Jeffrey D. Ullman. 1971. Principles of optimal page replacement. *J. ACM* 18, 1 (Jan. 1971), 80–93. DOI: <http://dx.doi.org/10.1145/321623.321632>
- Laszlo A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.* 5, 2 (1966), 78–101.
- Subhasis Das, Tor M. Aamodt, and William J. Dally. 2015. SLIP: Reducing wire energy in the memory hierarchy. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA’15)*. ACM, New York, NY, 349–361. DOI: <http://dx.doi.org/10.1145/1250662.1250709>
- Peter J. Denning, Y. C. Chen, and G. S. Shedler. 1968. *A Model for Program Behavior Under Demand Paging*. Technical Report RC-2301. IBM Research.
- Nam Duong, Dali Zhao, Taesu Kim, Rosario Cammarota, Mateo Valero, and Alexander V. Veidenbaum. 2012. Improving cache management policies using dynamic reuse distances. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*. IEEE Computer Society, Washington, DC, 389–400. DOI: <http://dx.doi.org/10.1109/MICRO.2012.43>
- Stijn Eyerman and Lieven Eeckhout. 2008. System-level performance metrics for multiprogram workloads. *IEEE Micro* 28, 3 (2008), 42–53.
- Aamer Jaleel, Robert S. Cohn, Chi-Keung Luk, and Bruce Jacob. 2008. CMP \$ im: A pin-based on-the-fly multi-core cache simulator. In *Proceedings of the 4th Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), co-located with ISCA*. 28–36.
- Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr, and Joel Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). In *ACM SIGARCH Computer Architecture News*, Vol. 38. 60–71.
- Daniel A. Jiménez. 2013. Insertion and promotion for tree-based PseudoLRU last-level caches. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*. ACM, New York, NY, 284–296. DOI: <http://dx.doi.org/10.1145/2540708.2540733>
- G. Keramidas, P. Petoumenos, and S. Kaxiras. 2007. Cache replacement based on reuse-distance prediction. In *Proceedings of the 25th International Conference on Computer Design (ICCD’07)*. 245–250. DOI: <http://dx.doi.org/10.1109/ICCD.2007.4601909>
- Samira Manabi Khan, Yingying Tian, and Daniel A. Jimenez. 2010. Sampling dead block prediction for last-level caches. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’10)*. 175–186.

- An-Chow Lai, Cem Fide, and Babak Falsafi. 2001. Dead-block prediction & dead-block correlating prefetchers. In *Proceedings. 28th Annual International Symposium on Computer Architecture*. IEEE, 144–154.
- Wei-Fen Lin and Steven K. Reinhardt. 2002. Predicting last-touch references under optimal replacement. Technical Report CSE-TR-447-02. University of Michigan, Ann Arbor, MI. <http://www.eecs.umich.edu/techreports/cse/2002/CSE-TR-447-02.pdf>.
- Haiming Liu, Michael Ferdman, Jaehyuk Huh, and Doug Burger. 2008. Cache bursts: A new approach for eliminating dead blocks and increasing cache efficiency. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 41)*. IEEE Computer Society, Washington, DC, 222–233. DOI: <http://dx.doi.org/10.1109/MICRO.2008.4771793>
- Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. 1970. Evaluation techniques for storage hierarchies. *IBM Syst. J.* 9, 2 (1970), 78–117.
- Scott McFarling. 1992. Cache replacement with dynamic exclusion. In *Proceedings of the 19th Annual International Symposium on Computer Architecture (ISCA'92)*. ACM, New York, NY, 191–200. DOI: <http://dx.doi.org/10.1145/139669.139727>
- Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A self-tuning, low overhead replacement cache. In *FAST*, Vol. 3. 115–130.
- Naveen Muralimanohar, Rajeev Balasubramonian, and Norm Jouppi. 2007. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40)*. IEEE Computer Society, Washington, DC, 314. DOI: <http://dx.doi.org/10.1109/MICRO.2007.30>
- Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. 1993. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*. ACM, New York, NY, 297–306. DOI: <http://dx.doi.org/10.1145/170035.170081>
- Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. 2011. MARSS: A full system simulator for multicore x86 CPUs. In *Proceedings of the 48th Design Automation Conference*. ACM, New York, NY, 1050–1055.
- Harish Patil, Robert Cohn, Mark Charney, Rajiv Kapoor, Andrew Sun, and Anand Karunanidhi. 2004. Pinpointing representative portions of large Intel Itanium programs with dynamic instrumentation. In *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, 81–92.
- Moinuddin K. Qureshi, Aamer Jaleel, Yale N. Patt, Simon C. Steely, and Joel Emer. 2007. Adaptive insertion policies for high performance caching. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*. ACM, New York, NY, 381–391. DOI: <http://dx.doi.org/10.1145/1250662.1250709>
- Kaushik Rajan and Govindarajan Ramaswamy. 2007. Emulating optimal replacement with a shepherd cache. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 40)*. IEEE Computer Society, Washington, DC, 445–454. DOI: <http://dx.doi.org/10.1109/MICRO.2007.14>
- Xipeng Shen, Jonathan Shaw, Brian Meeker, and Chen Ding. 2007. Locality approximation using time. In *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07)*. ACM, New York, NY, 55–61. DOI: <http://dx.doi.org/10.1145/1190216.1190227>
- Yannis Smaragdakis, Scott Kaplan, and Paul Wilson. 1999. EELRU: Simple and effective adaptive page replacement. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'99)*. ACM, New York, NY, 122–133. DOI: <http://dx.doi.org/10.1145/301453.301486>
- SPEC. 2014. SPEC CPU 2006. Retrieved from <http://www.spec.org/cpu2006/>.
- Harold S. Stone. 1993. *High-Performance Computer Architecture* (3ed.). Addison-Wesley, 74–75.
- Masamichi Takagi and Kei Hiraki. 2004. Inter-reference gap distribution replacement: An improved replacement algorithm for set-associative caches. In *Proceedings of the 18th Annual International Conference on Supercomputing (ICS'04)*. ACM, New York, NY, 20–30. DOI: <http://dx.doi.org/10.1145/1006209.1006213>
- Carole-Jean Wu, Aamer Jaleel, Will Hasenplaugh, Margaret Martonosi, Simon C. Steely Jr, and Joel Emer. 2011. SHIP: Signature-based hit predictor for high performance caching. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 430–441.

Received June 2015; revised August 2015; accepted August 2015