University of Maryland Technical Report UMIACS-TR-2007-33

Enhancing LTP-Driven Cache Management Using Reuse Distance Information

Wanli Liu and Donald Yeung
Department of Electrical and Computer Engineering
University of Maryland at College Park
{wanli,yeung}@eng.umd.edu

Abstract

Traditional caches employ the LRU management policy to drive replacement decisions. However, previous studies have shown LRU can perform significantly worse than the theoretical optimum, OPT [1]. To better match OPT, it is necessary to aggressively anticipate the future memory references performed in the cache. Recently, several researchers have tried to approximate OPT management by predicting last touch references *[2, 3, 4, 5]. Existing last touch predictors (LTPs) either correlate last touch references with* execution signatures*, like instruction traces [3, 4] or last touch history [5], or they predict* cache block life times *based on reference [2] or cycle [6] counts. On a predicted last touch, the referenced cache block is marked for early eviction. This permits cache blocks lower in the LRU stack–but with shorter reuse distances–to remain in cache longer, resulting in additional cache hits.*

This paper investigates two novel techniques to improve LTP-driven cache management. First, we propose exploiting reuse distance information to increase LTP accuracy. Specifically, we correlate a memory reference's last touch outcome with its global reuse distance history*. Our results show that for an 8-way 1 MB L2 cache, a 74 KB RD-LTP can reduce the cache miss rate by 11.5% and 14.5% compared to LvP and AIP [2], two state-of-the-art last touch predictors. These performance gains are achieved because RD-LTPs exhibit a much higher prediction rate compared to existing LTPs, and RD-LTPs often avoid evicting LNO last touches [5], increasing the proportion of OPT last touches they evict. Second, we also propose predicting actual reuse distance values using* reuse distance predictors *(RDPs). An RDP is very similar to an RD-LTP except its predictor table stores exact reuse distance values instead of last touch outcomes. Because RDPs*

predict reuse distances, we can distinguish between LNO and OPT last touches more accurately. Our results show an 80 KB RDP can improve the miss rate compared to an RD-LTP by an additional 3.7%.

1 Introduction

The performance of the cache memory hierarchy is critical to the overall performance of modern computer systems. In particular, the policies used to manage the contents of caches can have a major impact on hit rates, and hence, memory hierarchy effectiveness. Traditional caches employ the LRU policy to drive replacement decisions. However, previous studies have shown LRU can perform significantly worse than the theoretical optimum, OPT [1], especially for large and highly associative caches commonly found at the L2 level [5, 7]. These studies suggest an opportunity exists for more sophisticated replacement algorithms to provide higher performance.

To improve upon LRU and better match OPT, it is necessary to aggressively anticipate the future memory references performed in the cache. In the case of OPT, perfect knowledge about the *reuse distance* of memory references is available to the replacement algorithm, allowing it to always evict the block used furthest in the future. Recently, several researchers have tried to approximate such omniscient OPT management by predicting *last touch references* [2, 3, 4, 5]. On a predicted last touch, the referenced cache block is marked for early eviction since it is unlikely to be re-referenced prior to becoming the LRU block. This permits cache blocks lower in the LRU stack–but with shorter reuse distances–to remain in cache longer, resulting in additional cache hits.

At the heart of such sophisticated replacement algorithms are the last touch predictors (LTPs) used to predict last touch references and drive replacement decisions. To date, two major approaches have been considered for LTPs. The first approach correlates last touch references with *execution signatures*. Signature types that have shown the greatest promise include instruction traces [3, 4] and last touch history [5]. The second approach identifies last touches by predicting *cache block life times* based on either reference [2] or cycle [6] counts. In this approach, a last touch is assumed whenever the predicted life time of a block in the cache expires.

This paper investigates two novel techniques for improving LTP-driven cache management. First, we propose exploiting reuse distance information to increase LTP accuracy. Like last touches, reuse distances

associated with individual memory references also exhibit repeating patterns which can be captured by a hardware predictor. Specifically, we correlate a memory reference's last touch outcome with its *global reuse distance history* and the memory instruction's PC, and store the correlation in a hardware table. We call such a hardware structure a *reuse distance last touch predictor* (RD-LTP). To determine reuse distances, RD-LTPs observe a cache block's position in the LRU stack at reference time. Consequently, RD-LTPs can track reuse distances for blocks that remain in the cache. By augmenting the cache with *shadow tags* [8], RD-LTPs can also monitor the reuse distances of recently evicted cache blocks. This enables RD-LTPs to track LRU last touches even when cache management deviates from a true LRU policy due to early evictions.

Our results show that for an 8-way 1 MB L2 cache, a 74 KB RD-LTP can reduce the cache miss rate by 11.5% and 14.5% compared to LvP and AIP [2], two state-of-the-art last touch predictors. These performance gains are achieved for two reasons. First, RD-LTPs exhibit a much higher prediction rate, predicting 64.0% of the LRU last touches compared to only about 17% for LvP and AIP. Second, by using a simple MRU policy to select blocks for early eviction, RD-LTPs often avoid evicting LNO last touches [5], increasing the proportion of OPT last touches they evict.

Second, we also investigate techniques to further improve how we distinguish between LNO and OPT last touches, thus enabling even higher quality early eviction decisions. Specifically, we propose predicting actual reuse distance values using *reuse distance predictors* (RDPs). An RDP is identical to an RD-LTP except its predictor table stores exact reuse distance values instead of last touch outcomes. To be effective, RDPs must track reuse distances larger than the cache's natural LRU stack depth. We rely on the same shadow tags in RD-LTPs to provide the deeper reuse distance information. Because RDPs predict reuse distances, we can more exactly determine which cache blocks are used the farthest in the future, thus identifying OPT last touches more accurately. Our results show a 80 KB RDP can improve the miss rate compared to an RD-LTP by an additional 3.7%.

The remainder of this paper is organized as follows. After discussing related work in Section 2, Section 3 introduces our predictors. Then, Section 4 presents our experimental methodology. Next, Section 5 evaluates cache management policies that use RD-LTPs, and Section 6 evaluates cache management policies that use RDPs. Finally, Section 7 concludes the paper.

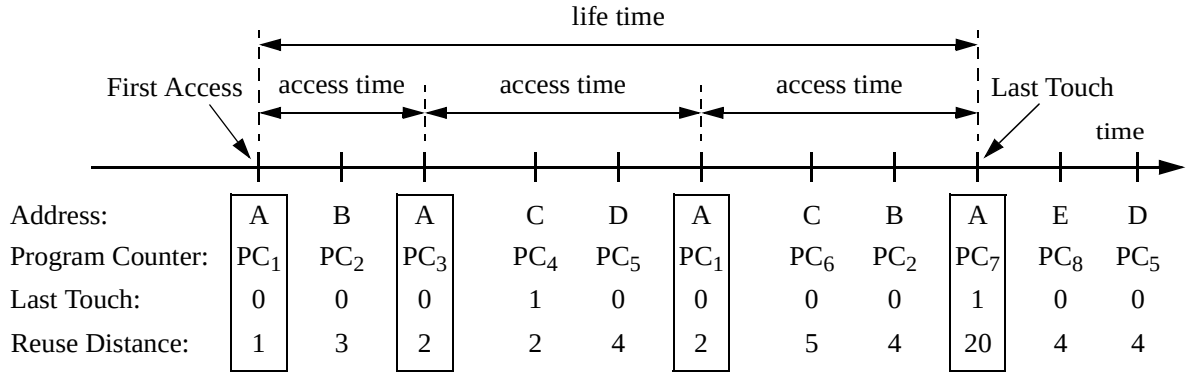


Figure 1. A memory address trace, and various information associated with the trace used to predict last touch and/or reuse distance. Information includes memory addresses, memory reference program counters, last touch history, and reuse distance. Access and live times are indicated for a sequence of references to the memory address A.

2 Related Work

This paper is closely related to previous work on last touch prediction. Several proposals for predicting a cache block's last touch have been explored. To illustrate the different approaches, Figure 1 shows a timeline of memory references performed on the memory location A (indicated by the boxes), beginning with a cache-missing reference that allocates A's block in cache, and ending with a last touch reference. Memory references performed on other locations in between references to A are also shown, and various runtime information associated with all the memory references is indicated below the timeline.

Existing LTPs predict last touches based on either signatures, life times, or access times. Signature-based LTPs associate each last touch reference with an execution signature that captures the runtime context for the last touch. In particular, Lai's LTP [3, 4] forms a signature from instruction traces. Truncated addition is performed on the sequence of program counter values for each memory reference to a cache block, thus encoding the trace of memory instructions leading up to the block's last touch. For example, in Figure 1, the sum $PC_1 + PC_3 + PC_1$ truncated to the desired length is an instruction trace signature for the last touch reference performed by PC_7. While effective for L1 caches, Lin and Reinhardt [5] show instruction trace signatures are far less accurate for large and highly-associative caches commonly found at the L2 level. Instead, they find L2 last touches are better correlated to last touch history. For each memory reference, the last touch history specifies a single bit–"0" for not a last touch and "1" for last touch–as indicated in

Figure 1. A memory reference's last touch signature is formed by concatenating the history bits from the N preceding memory references to the same cache block. For high prediction accuracy, $N = 16$ to 32 is required [5].

The Inter-Reference Gap (IRG) model [9] is another signature-based predictor similar to our approach. IRGs correlate predicted future reuse length (*i.e.* the number of memory references before the next reference to the same location) values with histories of previous reuse length values. In that regard, it's similar to our RDP. The main difference is our signatures and predicted values are *reuse distances*, not reuse lengths. Also, IRGs only accumulate history locally to a single memory block whereas our approach uses global history.

In contrast to signatures, Kharbutli and Solihin [2] predict last touches by observing either cache block life times or access times. When the life time of a block in the cache expires, the memory reference at the time of expiration is predicted as a last touch. Alternatively, if no reference to a cache block occurs after the access time elapses, then the most recent reference is predicted as a last touch. To quantify life and access times, Kharbutli and Solihin count memory references. In particular, the number of references to a cache block from the first access to the last touch quantifies life time, while the number of interceding references between two touches to the same cache block quantifies access time. For example, in Figure 1, the cache block containing location A has a life time of 4 and an access time of 2 (the worst-case time value is chosen). The counters for predicting life and access times are stored in hardware predictors, called LvP and AIP, respectively.

In addition to using memory reference counts, cache block life times (and hence last touches) can also be predicted based on cycle counts. Cache Decay [6] and Adaptive Mode Control [10] observe the number of cycles that have elapsed since a block's most recent reference, and marks the block as dead when the elapsed time exceeds a certain threshold. Another approach ties cache block liveness to program or runtime system execution [11]. For example, after a method referencing a block's data terminates or the block's data has been garbage collected, the cache block is assumed to have received its last touch. Such cycle-based and program-based approaches have been used to save energy (*e.g.*, predicted dead blocks are powered down to eliminate leakage), but the goal of determining a cache block's last touch is the same.

The predictors studied in this paper are signature-based predictors, but they differ from existing techniques

in two major ways. First, instead of using instruction traces, last touch history, or reuse lengths, we form signatures from sequences of reuse distance values. Moreover, our signatures are global in that they aggregate information from different memory locations; previous signatures contain information from a single memory location only. And second, compared to previous LTPs, we also predict more detailed reuse information. LTPs essentially predict a binary reuse outcome: the distance to the next use of a cache block is either greater than or less than the cache associativity, implying the current memory reference is a last touch or not a last touch, respectively, assuming LRU. For LRU last touches, we also predict how far into the future the next reference will be. This more exact information can be used to help distinguish between LNO and OPT last touches.

3 Predictors

This section presents our predictors. We begin by discussing global reuse distance history (Section 3.1), the key information used in our signatures. Then, we describe how predictions are performed in RD-LTPs (Section 3.2). Finally, we discuss LNO last touches, and introduce RDPs (Section 3.3).

3.1 Global Reuse Distance History

All our predictors correlate predicted outcomes with sequences of reuse distance values, called *reuse distance history*. Moreover, these sequences are *global* because they are formed from back-to-back memory references, not just references to the same memory location. The bottom of Figure 1 shows the sequence of reuse distance values for our example memory reference timeline. The sequence labels each memory reference with its reuse distance, *i.e.* the number of unique memory locations referenced before the next reference to the same location. For example, the first reference to B has a reuse distance of 3 because A, C, and D are referenced before B is referenced again. Let us define the *previous reuse distance* of a memory reference as the reuse distance of the most recent reference to the same location. For example, the second reference to C has a previous reuse distance of 2 because its previous reference to C (performed by PC_4) has a reuse distance of 2. Given these definitions, a memory reference's global reuse distance history contains the N previous reuse distance values immediately preceding the reference, where N is the history length. For example, the global reuse distance history of the last reference to A, assuming $N = 2$, is $\{2, 3\}$ because the two preceding

memory references to C and B have previous reuse distances of 2 and 3, respectively.

To enable the use of global reuse distance history in hardware predictors, we make two simplifications. First, we consider reuses at cache block granularity rather than individual memory words–*i.e.*, A–D in Figure 1 represent cache block addresses. This makes sense since cache management decisions are made at the cache block level anyways. And second, we compute each reuse distance value across references to the same cache set rather than across all memory references–*i.e.*, A–D in Figure 1 map to the same set.

With these simplifications, we can easily compute the previous reuse distance for certain memory references in hardware, and hence form global reuse distance histories, as long as the cache maintains LRU ordering between cache blocks. In particular, on a cache hitting memory reference, the per-set reuse distance of the previous memory reference to the same cache block is simply the block's position in the cache set's LRU stack. This permits us to track memory references' reuse distances so long as the associated cache blocks remain in cache. (Equivalently, we can observe any reuse distance between 0 and $CA - 1$, where CA is the cache associativity). Unfortunately, we cannot track the reuse distance for memory references whose blocks leave the cache since their associated LRU stack information is lost. When a cache block leaves the cache, we assign a reuse distance of CA to the last memory reference performed on the block (*i.e.*, its last touch), signifying the true reuse distance is unknown but is at least CA. For example, in Figure 1, the global reuse distance history for the last reference to D, assuming $N = 2$, $CA = 8$, and the reference to E is a cache miss, is $\{2, 8\}$.

3.2 Predictor Hardware

Having discussed global reuse distance history and how it's computed, we now present our predictors. We begin by describing how we predict last touches using RD-LTPs. Our other predictor, the RDP, employs very similar hardware, and will be discussed in Section 3.3.

Figure 2 shows the hardware organization of an RD-LTP, and illustrates the different steps involved in performing predictions and updating predictor state. An RD-LTP requires four additions to a conventional cache. First, a *global reuse distance history array* (GRDH array) is needed to store the per-set global reuse distance histories. Second, each cache tag is augmented with a *last touch (LT) bit* as well as a *signature*

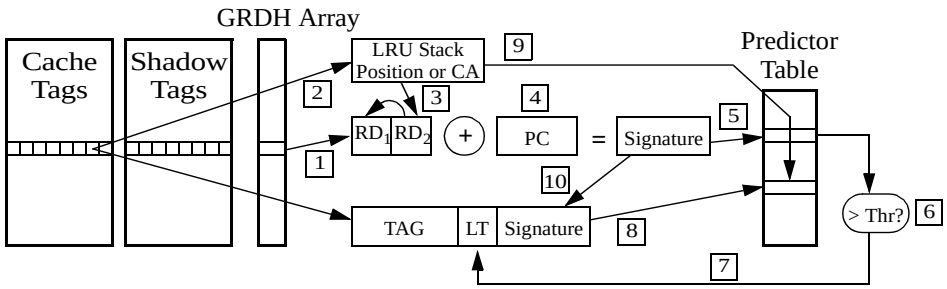


Figure 2. Reuse distance last touch predictor organization and actions. Steps 1–7 perform a prediction. Steps 8–10 update the predictor.

field containing the signature observed during the block's most recent reference. (In Figure 2, we assume an 8-way set associative cache, so there are 8 tag, LT, and signature fields in the main tag array). Third, a *shadow tag array* [8] is included to extend the LRU stack depth of the cache. Like normal tags, each shadow tag also includes a signature field as well (but no LT bit). Finally, a central *predictor table* is needed to store the prediction outcomes.

Labels "1" – "7" in Figure 2 illustrate the different steps for a last touch prediction on a cache hit. First, we read the GRDH array entry corresponding to the referenced cache set (label "1"). This entry contains the concatenated reuse distances for the N memory references prior to the current reference that map to the referenced set. We construct the N-lengthed global reuse distance history for the current memory reference by observing the LRU stack position of the referenced cache block (label "2"), and appending it to the GRDH entry (label "3"). (In Figure 2, we assume $N = 2$). Next, we XOR the global reuse distance history with the memory reference's PC (label "4") to form the signature for the current reference. This signature is used to index the predictor table (label "5"), producing a saturating counter whose value is compared against a threshold (label "6"). If the counter value is greater than the threshold, a last touch outcome is predicted; otherwise, a not lost touch outcome is predicted. The predicted outcome is written into the cache block's LT field (label "7"). In addition to cache hits, RD-LTPs also make predictions on cache misses. The same 7 steps in Figure 2 are performed, except CA is appended to the GRDH entry instead of the cache block's LRU stack position (see Section 3.1).

Labels "8" – "10" illustrate the different steps for updating the predictor. In particular, the hit/miss outcome of the current memory reference validates the correctness of the last touch prediction for the previous

reference to the same cache block. To permit updating the predictor with this actual outcome information, the signature associated with the previous prediction is stored along with the tag of the referenced cache block. This signature is used to index into the predictor table (label "8") so that the previous prediction's saturating counter can be updated (label "9"). If the current reference is a cache hit, the update can occur at reference time; in this case, the counter is decremented to reflect the cache hit. If the current reference is a cache miss, then the cache block (and hence its signature) is no longer in cache, and thus the update must occur at eviction time before the signature is lost. In this case, the counter is incremented to reflect the impending cache miss. Lastly, the current memory reference's signature is stored with the cache tag (label "10") to enable a predictor update on the next reference to the same cache block.

As Figure 2 shows, the ability to monitor the position of cache blocks in LRU stacks is critical to RD-LTPs. Unfortunately, once cache blocks leave the cache, RD-LTPs cannot track their reuse distances. This can become problematic when the cache acts on predictions to evict blocks early. In particular, if an incorrect last touch prediction leads to an early eviction, it is impossible to detect the misprediction and update the predictor accordingly since the cache block is no longer in cache when the next reference to the block (which would have been a cache hit) occurs. The cache not only suffers an additional miss, but the predictor will likely make the same misprediction in the future. Worse yet, the additional cache misses that such incorrect last touch predictions trigger also corrupt the global reuse distance history, inserting CA values into the history instead of the actual reuse distances. This can cause additional mispredictions and cache misses down the road.

To address this problem, we augment the cache with shadow tags, as shown in Figure 2. In particular, we implement a shadow tag array containing SA shadow tags. When a cache block is evicted, we remove its data from the cache, but retain its tag in a shadow tag entry. We maintain LRU ordering between *all* tags (normal and shadow), thus extending the cache's LRU stack depth by SA. This allows us to track the reuse distances of recently evicted cache blocks, including those evicted early due to last touch predictions, for as long as they remain in the shadow tag array (*i.e.*, until they become the least recently used among both normal and shadow blocks). The extended reuse distance visibility provided by the shadow tags allows us to identify premature evictions caused by last touch mispredictions, and hence, avoid corruption of global

reuse distance history and improve cache performance.

3.3 LNO vs OPT Last Touches

Like all previous LTPs, RD-LTPs predict the last touches observed under an LRU policy, a natural consequence of the fact that the underlying cache management policy is itself LRU. However, many LRU last touches are not last touches under OPT. These references are referred to as LRU non-OPT, or LNO, last touches [5]. LNO last touches typically have a reuse distance that is only slightly larger than CA, so they can be converted into cache hits if the referenced blocks are kept in cache a bit longer. In particular, when multiple cache blocks are marked as LRU last touches simultaneously, evicting those blocks with larger reuse distances in favor of those with shorter reuse distances can keep the soon-to-be-referenced blocks in cache longer than an LRU policy would, perhaps long enough to convert what would be cache misses under LRU into cache hits.

In this paper, we propose two methods for distinguishing LNO and OPT last touches. The first method works with our RD-LTP, and employs a very simple heuristic: when multiple cache blocks are marked by the RD-LTP as LRU last touches, pick the most-recently-used block that has been marked. We find the MRU marked block is often an OPT last touch. In Section 5, we will present results that validate this claim, and provide more intuition behind why it works. The second method takes a more direct approach: predict the actual reuse distance for each marked LRU last touch block. Then, we can simply pick the block with the largest predicted reuse distance. One challenge of the second method is the reuse distances we need are necessarily larger than CA since the cache blocks of interest are guaranteed to be LRU last touches. Observing such long reuse distances requires LRU stacks larger than CA. Recall from Section 3.2 that RD-LTPs already extend the LRU stack using shadow tags to improve prediction accuracy. Such shadow tags can also track reuse distances beyond CA for distinguishing LNO and OPT last touches.

To enable the second method, we propose reuse distance predictors, or RDPs. Our RDPs predict the exact reuse distance up to depth $CA + SA$. They are very similar to RD-LTPs. In particular, they use signatures based on global reuse distance history, so all the mechanisms in Figure 2 for creating signatures and indexing into the predictor table remain the same. The main difference is RDPs store actual reuse distance values in

the predictor table instead of saturating counters. Notice, on a cache hit, we actually know the exact reuse distance value by observing the position of the referenced cache block in the LRU stack. An RD-LTP uses this information to form signatures (label "3" in Figure 2), but ignores it when updating the predictor (label "9"). An RDP updates its predictor with this exact reuse distance value. Specifically, the predictor entry is set to the observed LRU stack position of a cache block on a hit to the cache tags (including both normal and shadow tags); otherwise, it is set to $CA + SA$, signifying a miss in all the tags. Another (more minor) difference is the LT field must be replaced with a reuse distance value field to store predicted reuse distances. It is important to emphasize that RDPs can only provide limited reuse distance information (between 0 and $CA + SA$). However, as we will see in Sections 5 and 6, this is an important range.

4 Experimental Methodology

The remainder of this paper conducts an in-depth evaluation of our predictors from Section 3, applying them for cache management and quantifying the resulting cache performance. Our evaluation focuses on managing the L2 cache since this is an especially critical part of the memory hierarchy for modern high-performance CPUs. As part of our evaluation, we also compare our techniques against existing LTPs and several ideal cache management algorithms.

Our evaluation employs trace-driven simulation. We use the in-order processor model from the M5 simulator [12] configured with an L1 and L2 cache to simulate several uniprocessor benchmarks. The M5 simulator was instrumented to record the post-L1 memory address trace seen at the input to the L2 cache during execution-driven simulation. Along with each traced L2 reference, we also record the PC of the instruction performing the reference. After running all the M5 simulations, we replayed the L2 memory traces on a trace-driven cache simulator. The top portion of Table 1 reports the parameters we used for the L1 and L2 caches in the M5 simulations. The same L2 cache configuration used in M5 was also used in the trace-driven cache simulations.

The cache simulator includes architectural models for an RD-LTP and an RDP. In the bottom portion of Table 1, we report the configuration parameters for the predictors. We use a global reuse distance history of length 2. Since the simulated cache is 8-way set associative, each reuse distance value in the history is between 0–8 (0–7 encode the 8 positions in the LRU stack, while 8 encodes references not found in the LRU

Cache Parameters			
L1 I-cache		16 Kbyte, 2-way set associative, 64 byte blocks	
L1 D-cache		16 Kbyte, 2-way set associative, 64 byte blocks	
L2 U-cache		1 Mbyte, 8-way set associative, 64 byte blocks	
Predictor Parameters			
History Length	2	Shadow Tags	8 per cache set, 15 bits each
Reuse Distance Values	3 bits	RD-LTP Table Entries	2 bits
Predictor Table	1024 entries	RDP Table Entries	4 bits
Signature Size	10 bits		

Table 1. Cache and predictor parameter settings.

stack). We encode reuse distances of 0 and 1 using the same value, thus enabling a compact 3-bit encoding. (We did not notice any performance degradation due to this simplification). This results in a 6-bit global reuse distance history. For the predictor table, we assume 1024 entries. To form the 10-bit signature needed to index the table, we pad the 6-bit history with 4 leading 0s, and XOR the result with the 10 least significant bits of the memory reference's PC.[1] For each set in the cache, we augment the normal tags with 8 shadow tags. Puzak's thesis [8] shows many references have reuse distances slightly beyond the cache associativity. Using $SA = CA$ enables RDPs to track these important short reuses. 8 shadow tags is probably overkill for RD-LTPs; nevertheless, we use the same number of shadow tags to maintain uniformity. Finally, the predictor table entries differ in size depending on the type of predictor. RD-LTP table entries contain a 2-bit saturating counter while RDP entries contain a 4-bit reuse distance value. (The 4-bit RDP table entry encodes reuse distance values between 0–16 using the same compact encoding trick described earlier; 0–15 encode the 16 positions in the LRU stack including shadow tags, and 16 encodes references not found in the LRU stack).

During trace-driven simulations, the modeled predictors are used to drive cache management decisions. For the RD-LTP, the predictor marks the LT bit for all predicted last touch blocks (see Section 3.2). On a cache miss, we first consider marked blocks for eviction, then we consider all remaining blocks in LRU order. If more than one block is marked, we evict the MRU block among them, as discussed in Section 3.3. For the RDP, the predictor provides a reuse distance value for each cache block (see Section 3.3). On a cache miss, we first consider for eviction the blocks with reuse distance = 16 in MRU order (similar to the MRU ordering in RD-LTPs). If no such block exists, we evict the block with the largest reuse distance value. If there are multiple blocks with the largest reuse distance value, we evict the MRU block among them.

[1]Since M5 instructions are 4 bytes wide, we divide the PC by 4 prior to truncating to 10 bits. This removes the two least significant 0 bits in all instruction PCs.

High Potential			Low Potential		
App	Skip Ins	Type	App	Skip Ins	Type
ammp	2,600M	FP	perlbmk	1,700M	Int
art	200M	FP	eon	100M	Int
bzip2	1,100M	Int	gzip	200M	Int
gcc	2,100M	Int	gap	200M	Int
mcf	2,100M	Int	apsi	2,300M	FP
mesa	500M	FP	fma3d	1,900M	FP
parser	1,000M	Int	equake	400M	FP
sixtrack	3,700M	FP	lucas	800M	FP
twolf	2,000M	Int	swim	400M	FP
vortex	100M	Int	applu	800M	FP
vpr	300M	Int			
wupwise	3,400M	FP			

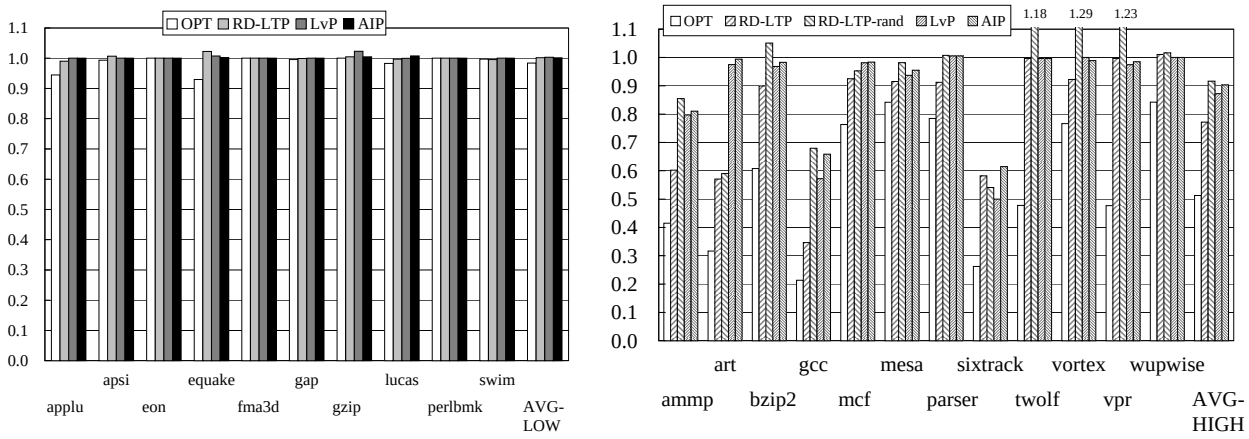
Table 2. SPEC CPU2000 benchmarks used to drive our cache simulations.

We selected 22 SPEC CPU2000 benchmarks to drive our simulations. Table 2 lists these benchmarks. As Section 5 will explain, we identified 12 "interesting" benchmarks out of this original 22 (the ones in the High Potential column) to focus on in our study. For all the benchmarks, we use the pre-compiled Alpha binaries from Chris Weaver[2] which are built with the highest level of compiler optimization. All of our benchmarks use the reference input set. We selected simulation regions by using SimPoint [14] to analyze the first 16 billion instructions (or the entire execution) of each benchmark, and picked the earliest representative region reported by SimPoint. When acquiring our memory traces in M5, we fast-forward each benchmark to its representative region (the columns labeled "Skip Ins" in Table 2 report the number of fast forwarded instructions). Then, we turn on tracing, and simulate for 2 billion instructions. Lastly, the columns labeled "Type" in Table 2 report each benchmark's type, either integer or floating point.

To enable a comparison against existing LTP techniques, we implemented the AIP and LvP last touch predictors [2] described in Section 2. To the best of our knowledge, these predictors represent the state-of-the-art for LTP-driven cache management, both in terms of performance and hardware cost. They have been shown to outperform several other existing LTPs [2]. To provide more insight into our predictors' performance, we also implemented the OPT policy [1] as well as several other oracle predictors which we will explain in Section 5.

Finally, an important issue is the hardware cost of our technique. Given the configuration parameters in Table 1, our RD-LTP and RDP incur 74 and 80 Kbyte of additional storage, respectively, on top of a

[2]These SPEC CPU2000 Alpha binaries are distributed as part of the SimpleScalar tools [13], and are available at the SimpleScalar website.



A. Low Potential B. High Potential

Figure 3. Cache miss rates for the A. low potential and B. high potential benchmarks achieved by RD-LTP, LvP, AIP, RD-LTP-rand, and OPT. The averages across each benchmark category are reported in the bars labeled "AVG-LOW" and "AVG-HIGH," respectively.

conventional cache. This extra hardware is needed to implement the per-block prediction and signature fields, shadow tags, GRDH array, and predictor tables illustrated in Figure 2. Compared to the 1 Mbyte L2 cache these predictors are used to manage, this represents less than 8% overhead for both predictors. In comparison, Kharbutli and Solihin report 61 and 57 Kbyte of additional storage for AIP and LvP, respectively, assuming a 512 Kbyte L2 cache [2]. For a 1 Mbyte L2 cache, this overhead increases to 82 and 73 Kbytes, respectively, which is very similar to our overhead. Unfortunately, the predictor table suggested in Kharbutli and Solihin's previous study achieved poor performance for several of our benchmarks. Hence, in our study, we use infinite predictor tables for LvP and AIP. (With infinite tables, our LvP and AIP performance is similar to what is reported in [2]).

5 Last Touch Prediction Results

We begin our evaluation by studying the performance achieved when driving cache management decisions using our RD-LTP predictor. Later, in Section 6, we will evaluate the RDP predictor.

5.1 Performance Evaluation

Figure 3 presents the cache miss rates achieved by RD-LTP, and compares them against LvP, AIP, and OPT. (The bars labeled "RD-LTP-rand" will be explained later in Section 5.2). All of the miss rates in Figure 3 have been normalized to the miss rate achieved by an LRU policy for the same benchmark. Our

first result is that intelligent cache management provides very little benefit in several SPEC benchmarks, especially when LRU and OPT are separated by a small difference in performance. Since OPT is theoretically optimal, a small LRU-OPT difference implies there isn't much opportunity for improvement. We divide our benchmarks into two categories based on this observation. Benchmarks with an LRU-OPT difference of 10% or less are placed in the low potential category, while benchmarks with an LRU-OPT difference greater than 10% are placed in the high potential category. The 10 benchmarks in Figure 3A are the low potential benchmarks, and the bars labeled "AVG-LOW" report the geometric mean across these benchmarks. As the AVG-LOW bars show, RD-LTP, LvP, and AIP are all within 1% of the performance achieved by LRU. Since there's no room for improvement, we do not consider these benchmarks further in this paper.

The 12 benchmarks in Figure 3B are the high potential benchmarks, and the bars labeled "AVG-HIGH" report the geometric mean across these benchmarks. Unlike the low potential benchmarks, there is significant opportunity for performance gains in the high potential benchmarks as the LRU-OPT difference is nearly 50%. As the AVG-HIGH bars show, RD-LTP capitalizes on this opportunity, reducing the miss rate over LRU by 22.8%. In addition, compared to existing LTPs, RD-LTP achieves a miss rate that is 11.5% and 14.5% lower than LvP and AIP, respectively. In particular, RD-LTP outperforms both LvP and AIP in 8 out of the 12 benchmarks, outperforms AIP alone in 1 benchmark, and matches the performance of LvP and AIP in 1 benchmark. These improvements reduce by 28.0% the performance gap separating LvP/AIP from OPT.

To provide insight into how RD-LTP achieves its performance gains, Figure 4 shows the accuracy of the predictions performed by RD-LTP, LvP, and AIP. Each bar in Figure 4 breaks down the last touch outcomes for each predictor into 3 categories. Components labeled "Correct Prediction" report predictions that correctly identify LRU last touch references; components labeled "Not Predicted" report LRU last touch references that are not identified by the predictor; and components labeled "Wrong Prediction" report the predictions that incorrectly identify LRU last touch references (*i.e.*, these references are not LRU last touches). All bars are normalized to the total number of LRU last touch references in the corresponding benchmark, with the last group of bars reporting the average across the 12 benchmarks.

As Figure 4 shows, RD-LTP correctly predicts a much larger fraction of the LRU last touch references

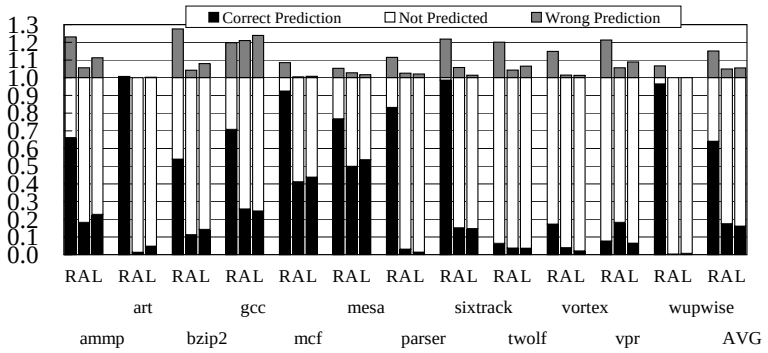
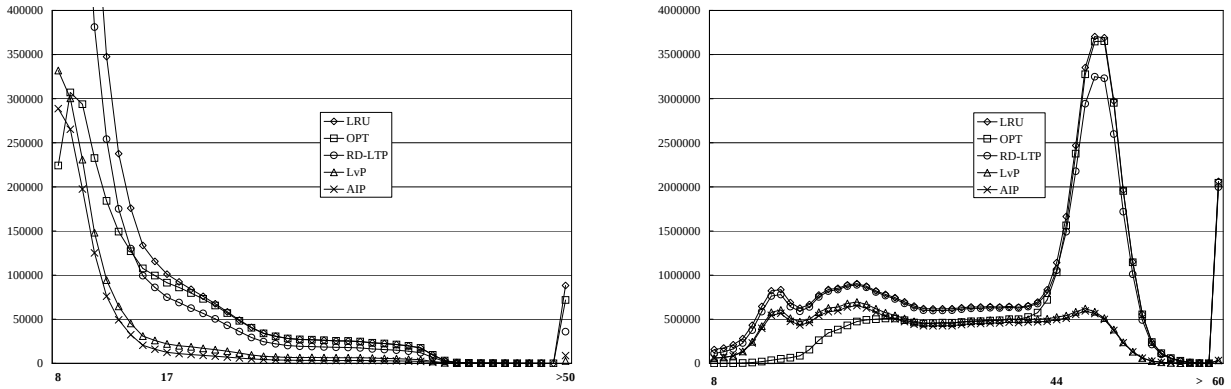


Figure 4. Prediction accuracy of RD-LTP, LvP, and AIP.

than either LvP or AIP. On average, RD-LTP identifies 64.0% of the LRU last touches compared to only 16.0% and 17.4% for LvP and AIP, respectively. Because RD-LTP correctly identifies a greater number of LRU last touches, it has the *potential* to perform a larger number of beneficial early evictions. (In a moment, we will discuss how RD-LTP capitalizes on this potential). Unfortunately, the higher prediction rate is also accompanied by a larger number of mispredictions. As the "Wrong Prediction" components in Figure 4 show, RD-LTP incurs 15.1% mispredictions whereas LvP and AIP incur only 5.5% and 4.9%, respectively. Such mispredictions can lead to premature evictions, converting some LRU cache hits into cache misses. However, RD-LTP's higher prediction rate far outweighs the negative consequences of its mispredictions.

These prediction accuracy results demonstrate RD-LTP is a more effective predictor than LvP and AIP. We credit three factors. First, last touch events are highly correlated to global reuse distance history. Our signatures simply identify more last touches. Second, RD-LTP's shadow tags improve predictor training. As discussed in Section 3.2, once the cache management hardware begins acting on predictions and performing early evictions, the LRU last touch outcomes of blocks that leave the cache early cannot be observed. Our shadow tags allow us to continue tracking recently evicted blocks, thus permitting us to observe LRU last touches even when replacement deviates from a strict LRU order. Finally, because RD-LTP's accuracy is inherently higher than LvP and AIP, it can be applied more aggressively. RD-LTP predicts *all* memory references; in contrast, LvP and AIP avoid predicting memory references with low accuracy (both predictors employ confidence mechanisms). The smaller pool of predicted memory references in LvP and AIP further reduces their total correct predictions.



A. AMMP B. MCF

Figure 5. Last touch reference histograms under LRU, OPT, LvP, AIP, and RD-LTP cache management for the A. AMMP and B. MCF benchmarks. CRD = 17 and 44 for AMMP and MCF, respectively.

5.2 Quality of Evictions

While RD-LTPs achieve high prediction rates, this alone does not explain their performance advantage. As discussed in Section 3.3, not all LRU last touch evictions are profitable. In particular, LNO last touches can be converted into cache hits by retaining them in cache a bit longer. Hence, simply predicting a large number of LRU last touches and evicting them does not guarantee high performance. It's also important to select the *best* last touch candidates for eviction.

To provide further insight, Figure 5 shows several histograms of last touch references under different cache management policies. For different reuse distances (X-axis), each histogram plots the number of last touch references exhibiting that reuse distance (Y-axis). The histograms for LRU and OPT include *actual* last touch references (*i.e.*, all of these lead to evictions); the histograms for RD-LTP, LvP, and AIP include *predicted* LRU last touch references (*i.e.*, only a subset of these lead to evictions). The rightmost point in each histogram reports the cumulative count for all reuse distances beyond the end of the X-axis. Figures 5A and B report histograms for two typical benchmarks, AMMP and MCF, respectively.

First, let us consider the LRU and OPT histograms in Figure 5. Notice the number of OPT last touches is always smaller than the number of LRU last touches. The difference between the LRU and OPT histograms constitutes the LNO last touches (the evictions that are last touches under LRU but not under OPT). Most importantly, notice that beyond some reuse distance, practically all OPT last touches are also LRU last touches, *i.e.* there are very few LNO last touches. For example, in AMMP, beyond a *critical reuse distance*

				% LRU Predicted > CRD			% OPT Predicted < CRD			
	CRD	Type	%OPT	RDLTP	LvP	AIP	RDLTP	LvP	AIP	Ratio
ammp	17	A	36.2%	70.0%	21.6%	12.2%	209.9%	73.8%	62.1%	5.0
art	14	A	37.2%	98.7%	3.8%	4.1%	439.0%	21.6%	4.2%	7.3
bzip2	29	A	33.1%	59.9%	12.6%	13.5%	103.6%	28.9%	21.1%	3.4
gcc	17	A	4.2%	38.3%	2.6%	5.1%	362.9%	121.5%	127.1%	171.6
mcf	44	B	66.7%	88.8%	16.9%	16.2%	183.2%	138.2%	129.4%	1.0
mesa	54	B	75.0%	87.3%	65.4%	57.0%	126.9%	67.8%	79.6%	0.5
parser	21	B	51.6%	90.7%	1.1%	2.8%	116.6%	2.3%	4.8%	1.1
sixtrack	11	A	0.5%	68.5%	68.5%	68.5%	65.8%	69.6%	72.1%	4.0
twolf	12	A	19.9%	9.3%	5.1%	5.0%	14.5%	8.1%	8.3%	6.3
vortex	35	B	79.6%	14.2%	0.5%	3.7%	62.0%	12.0%	11.4%	1.1
vpr	12	A	6.8%	13.7%	11.6%	8.3%	16.5%	13.6%	12.2%	14.3
wupwise	64	B	98.8%	95.9%	0.0%	0.0%	1653.1%	3.8%	3.7%	0.2
AVG	27.5		42.5%	61.3%	17.5%	16.4%	279.5%	46.8%	44.7%	18.0

Table 3. Statistics related to last touch reference histograms for all 12 high potential benchmarks.

(CRD) of 17, 90% or more of the LRU last touches are also OPT last touches. The same happens for MCF, but at CRD=44. This makes sense: LRU last touches with large reuse distances have no hope of becoming cache hits, so they are also likely to be OPT last touches. This implies that beyond CRD, *it doesn't matter which LRU last touches we evict–all of them are profitable.* We find that all benchmarks exhibit this property, though the exact CRD value is application dependent. In Table 3, the column labeled "CRD" reports the CRDs for all our benchmarks. As the last row in Table 3 shows, on average, there are very few LNO last touches beyond a reuse distance of 27.5.

An important question then is what fraction of the OPT last touches occur beyond CRD? If a large number do, then predicting LRU last touches essentially predicts most of the OPT last touches as well. However, if many OPT last touches occur below CRD, then LRU last touch prediction alone does not uniquely identify the OPT last touches since there are lots of LNO last touches in that case. We will call a benchmark "type A" if 50% or more of its OPT last touches occur beyond CRD; otherwise, we will call it "type B." In Figure 5, we see AMMP is a type A benchmark, while MCF is a type B benchmark. In AMMP, only 36.2% of the OPT last touches occur beyond CRD, whereas in MCF, 66.7% of the OPT last touches occur beyond CRD. The two columns in Table 3 labeled "Type" and "%OPT" report each benchmark's type and percentage of OPT last touches beyond CRD. As Table 3 shows, there are 7 type A benchmarks and 5 type B benchmarks. On average, 42.5% of the OPT last touches occur beyond CRD. These results show a little less than half of OPT last touches are directly predictable via LRU last touch prediction; however, the other half occur side-by-side with a significant number of LNO last touches.

Now, let us consider the RD-LTP, LvP, and AIP histograms in Figure 5. First, we examine how many of the LRU last touches occurring beyond CRD are correctly predicted by each predictor. As Figure 5 shows, RD-LTP achieves a significantly larger coverage of the LRU last touches beyond CRD. For AMMP, RD-LTP predicts 70.0% of these long-reuse LRU last touches, while LvP and AIP predict only 21.6% and 12.2%, respectively. For MCF, RD-LTP predicts 88.8% of the long-reuse LRU last touches, while LvP and AIP predict only 16.9% and 16.2%, respectively. In Table 3, the three columns labeled "% LRU Covered > CRD" report the percentage of LRU last touches beyond CRD predicted by RD-LTP, LvP, and AIP across all our benchmarks. On average, RD-LTP predicts 61.3% of the long-reuse LRU last touches, while LvP and AIP predict only 17.5% and 16.4%, respectively. From these results, we conclude that RD-LTP's higher prediction rate translates into a noticeably larger coverage of the long-reuse LRU last touches compared to LvP and AIP. Note that for high performance, it is critical to predict these long-reuse LRU last touches since they are all likely to be OPT last touches.

Next, we examine the LRU last touches that occur below CRD. Compared to LvP and AIP, Figure 5 shows RD-LTP makes significantly more short-reuse LRU last touch predictions. However, the number of such short-reuse predictions exceeds even the number of OPT last touches. Consequently, many of the LRU last touches that RD-LTP predicts are LNO last touches. In Table 3, the three columns labeled "% OPT Predicted < CRD" report the number of predicted LRU last touches below CRD in RD-LTP, LvP, and AIP. To convey the amount of "over-prediction," we report these statistics as a percentage of OPT (instead of LRU) last touches. As Table 3 shows, RD-LTP predicts more LRU last touches than OPT last touches in 8 benchmarks. LvP and AIP predict fewer last touches than OPT in all benchmarks except for two. On average, RD-LTP predicts 2.80 times as many short reuse last touches as OPT, while LvP and AIP predict slightly less than half as many as OPT.

These results show RD-LTP predicts a large number of LNO last touches amongst the short-reuse LRU last touches. Hence, it is possible for RD-LTP to detrimentally evict these LNO last touches, thus missing opportunities for cache hits. We find this does not happen frequently for the following reason. Because RD-LTP makes a large number of LRU last touch predictions, it is almost always the case that multiple cache blocks are marked at eviction time. As discussed in Section 3.3, we evict the *MRU block* amongst all

the marked blocks. In many cases, the MRU block is an OPT last touch rather than an LNO last touch. To understand why, we must look at the relative number of long-reuse and short-reuse predicted LRU last touches. In Table 3, the column labeled "Ratio" reports the ratio of long-reuse (greater than CRD) to short-reuse (smaller than CRD) LRU last touches among all LRU last touches predicted by RD-LTP. This data shows that in 2 benchmarks, long-reuse predictions dominate (ratio > 1.0); in 7 benchmarks short-reuse predictions dominate (ratio < 1.0); and in 3 benchmarks, neither dominates (ratio ≈ 1.0). For the first case, most marked blocks have reuse distance $>$ CRD. As already discussed, it doesn't matter which blocks we evict in this case since all are likely to be OPT last touches. MRU is just as good as any other policy. For the second case, most marked blocks have reuse distance $<$ CRD. Because these are short reuse distances, marked blocks that are deeper in the LRU stack are significantly closer to their next reuse than marked blocks higher up in the LRU stack. By evicting the MRU block, we dramatically increase our chances of retaining the older block(s) long enough to experience a cache hit. Finally, for the last case, marked blocks have an equal chance of exhibiting short or long reuse distances. The MRU policy does not provide any benefit in this case (though it doesn't hurt either). Fortunately, only 3 benchmarks fall into this category.

To illustrate the MRU policy is critical, the "RD-LTP-rand" bars in Figure 3 report the miss rate for our RD-LTP predictor, but instead of picking the MRU block amongst marked blocks at eviction time, we pick a block randomly. As Figure 3 shows, RD-LTP-rand performs significantly worse than RD-LTP. In fact, RD-LTP-rand even performs slightly worse than LvP and AIP.

6 Reuse Distance Prediction Results

Figure 6 presents our RDP results. In Figure 6, the bars labeled "RDP" report the miss rates achieved when driving cache management decisions using an RDP across our high potential benchmarks. For comparison, the miss rates achieved by RD-LTP and OPT have been included from Figure 3. All bars are normalized to the LRU miss rate for each benchmark, and the group of bars labeled "AVG" report the geometric mean across all the benchmarks. Comparing the RDP and RD-LTP bars, we see RDP provides an additional 3.7% miss rate reduction over RD-LTP on average. And comparing RDP to the LvP and AIP results from Figure 3, we see RDP improves the miss rate by 14.9% and 17.8%, respectively, over existing LTP techniques.

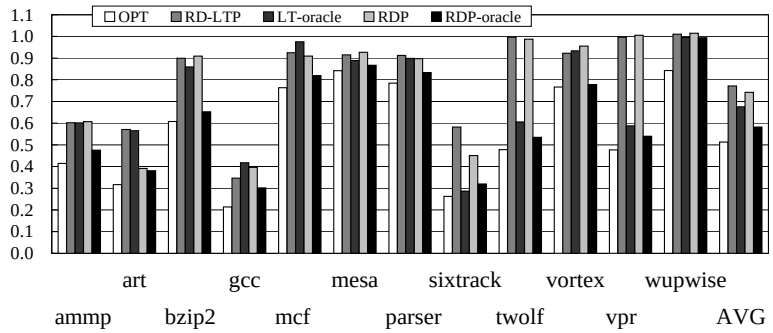


Figure 6. Cache miss rates achieved by OPT, RDP-LTP, LT-oracle, RDP, and RDP-oracle for the high potential benchmarks.

The additional benefit achieved by RDPs is due to the detailed reuse distance information provided by the predictor. As discussed in Section 5.2, while some predictions identify LRU last touches beyond CRD which are likely to be OPT last touches, many predictions identify LRU last touches below CRD that may possibly be LNO last touches. In the latter case, RDPs help by providing the exact reuse distance for marked blocks. Hence, when multiple blocks are marked, the block referenced farthest in the future can be identified directly, instead of using the MRU heuristic discussed in Sections 3.3 and 5.2.

To further understand our RDP result, Figure 6 also reports two ideal cache management algorithms, LT-oracle and RDP-oracle. LT-oracle is RD-LTP with perfect last touch information. In LT-oracle, LRU last touch blocks are always marked perfectly. But like RD-LTPs, LT-oracle still uses the MRU policy to select a marked block for eviction.[3] RDP-oracle is RDP with perfect reuse distance information. In RDP-oracle, LRU last touch blocks are always labeled with their actual reuse distances perfectly. As Figure 6 shows, LT-oracle improves upon RD-LTP by 12.5%. This represents the performance lost by RD-LTP due to predictor inaccuracy (*i.e.*, the "Not Predicted" and "Wrong Prediction" components in Figure 4). In addition, Figure 6 also shows RDP-oracle improves upon LT-oracle by 13.8%. This performance difference represents the actual potential benefit of exact reuse distance information. Unfortunately, RDP does not fully achieve this potential, as demonstrated by its 3.7% performance gain over RD-LTP. RDP's inability to achieve its full potential is due to inaccuracies in predicting the exact reuse distance.

[3]Although LT-oracle has perfect last touch information, the MRU policy may still mistakenly evict LNO last touches over OPT last touches. In fact, LT-oracle may perform worse than RD-LTP if the additional correct last touch predictions expose more LNO last touches for eviction.

7 Conclusion

This paper advances the state-of-the-art in LTP-driven cache management by investigating two novel techniques. First, we propose RD-LTPs, a new signature-based LTP that correlates last touch outcomes with global reuse distance history and the memory instruction's PC. To determine reuse distances, RD-LTPs observe a cache block's position in the LRU stack at reference time. By augmenting the cache with shadow tags, RD-LTPs can also monitor the reuse distances of recently evicted cache blocks. Our results show that for an 8-way 1 MB L2 cache, a 70 KB RD-LTP can reduce the cache miss rate by 11.5% and 14.5% compared to LvP and AIP, respectively. We find RD-LTPs exhibit a much higher prediction rate, predicting 64.0% of the LRU last touches compared to only about 17% for LvP and AIP. Second, we also find RD-LTPs often avoid evicting LNO last touches by employing a simple MRU policy to select blocks for early eviction amongst all LRU last touches.

Second, we also propose RDPs, a new technique that predicts actual reuse distance values. An RDP is very similar to an RD-LTP except its predictor table stores exact reuse distance values instead of last touch outcomes. Because RDPs predict reuse distances, we can better determine which cache blocks are used farthest in the future, thus distinguishing LNO and OPT last touches more precisely. Our results show an 80 KB RDP can improve the miss rate compared to an RD-LTP by an additional 3.7%.

References

[1] L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1996.

[2] M. Kharbutli and Y. Solihin, "Counter-Based Cache Replacement Algorithms," in *Proceedings of the International Conference on Computer Design*, (San Jose, CA), October 2005.

[3] A.-C. Lai and B. Falsafi, "Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction," in *Proceedings of the 27th International Symposium on Computer Architecture*, 2000.

[4] A.-C. Lai, C. Fide, and B. Falsafi, "Dead-Block Prediction & Dead-Block Correlating Prefetchers," in *Proceedings of the 28th International Symposium on Computer Architecture*, 2001.

[5] W.-F. Lin and S. K. Reinhardt, "Predicting Last-Touch References under Optimal Replacement," CSE-TR 447-02, University of Michigan, 2002.

[6] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *Proceedings of the 28th International Symposium on Computer Architecture*, pp. 240–251, 2001.

[7] W. A. Wong and J.-L. Baer, "Modified LRU Policies for Improving Second-Level Cache Behavior," in *Proceedings of the 6th International Symposium on High-Performance Computer Architecture*, pp. 49–60, 2000.

[8] T. R. Puzak, "Analysis of Cache Replacement Algorithms." February 1985.

[9] V. Phalke and B. Gopinath, "An inter-reference gap model for temporal locality in program behavior," *SIGMETRICS*, pp. 291–300, 1995.

[10] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive Mode Control: A Static-Power-Efficient Cache Design," *ACM Transactions on Embedded Computing Systems*, vol. 2, no. 3, 2002.

[11] G. Chen, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, and M. Wolczko, "Tracking Object Life Cycle for Leakage Energy Optimization," in *Proceedings of the ISSS/CODES joint conference*, (Newport Beach, CA), October 2003.

[12] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 Simulator: Modeling Networked Systems," *IEEE Micro*, vol. 26, pp. 52–60, July/August 2006.

[13] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," CS TR 1342, University of Wisconsin-Madison, June 1997.

[14] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *Proceedings of the 10th International Conference on Parallel Architectures and Compilation Techniques*, September 2001.