

# An Analytical Model for Cache Replacement Policy Performance \*

Fei Guo and Yan Solihin  
Dept. of Electrical and Computer Engineering  
North Carolina State University  
{fguo, solihin}@ece.ncsu.edu

## ABSTRACT

Due to the increasing gap between CPU and memory speed, cache performance plays an increasingly critical role in determining the overall performance of microprocessor systems. One of the important factors that affect cache performance is the cache replacement policy. Despite the importance, current analytical cache performance models ignore the impact of cache replacement policies on cache performance. To the best of our knowledge, this paper is the first to propose an analytical model which predicts the performance of cache replacement policies. The input to our model is a simple circular sequence profiling of each application, which requires very little storage overhead. The output of the model is the predicted miss rates of an application under different replacement policies. The model is based on probability theory and utilizes Markov processes to compute each cache access' miss probability. The model uses realistic assumptions and relies solely on the statistical properties of the application, without relying on heuristics or rules of thumbs. The model's run time is less than 0.1 seconds, much lower than that of trace simulations. We validate the model by comparing the predicted miss rates of seventeen Spec2000 and NAS benchmark applications against the miss rates obtained by detailed execution-driven simulations, across a range of different cache sizes, associativities, and four replacement policies, and show that the model is very accurate. The model's average prediction error is 1.41%, and there are only 14 out of 952 validation points in which the prediction errors are larger than 10%.

## Categories and Subject Descriptors

C.4[Performance of Systems]:Modeling techniques;  
I.6.5[Simulation and Modeling]:Model Development-  
*Modeling methodologies*

**General Terms:** Algorithms, Design, Experimentation, Performance

\*This work is supported in part by the National Science Foundation through grant CNS-0406306, Faculty Early Career Development Award CCF-0347425, and by North Carolina State University.

Permission to make digital or hard copies of all or part 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 bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*SIGMetrics/Performance'06*, June 26–30, 2006, Saint Malo, France.

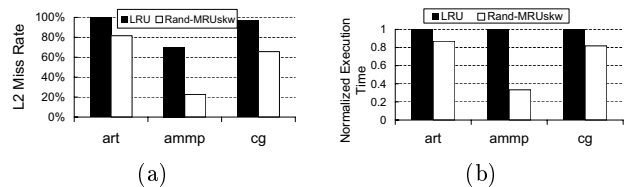
Copyright 2006 ACM 1-59593-320-4/06/0006 ...\$5.00.

**Keywords:** Cache performance, Analytical model, Cache replacement policy

## 1. INTRODUCTION

Due to the increasing gap between CPU and memory speed, cache performance plays an increasingly critical role in determining the overall performance of microprocessor systems. Cache replacement policy, in addition to cache associativity and block size, is an important factor that affects cache performance for a fixed cache size. The performance variation between different cache replacement policies can be quite significant in many cases.

To illustrate the extent of the performance variation, Figure 1 compares the L2 cache miss rates and normalized execution time of three memory-intensive Spec2000/NAS applications under two replacement policies: Least Recently Used (LRU) replacement which is the most popular implementation in caches, and random replacement policy which replaces more recently used lines with a higher probability than less recently used lines (Rand-MRU<sub>skw</sub>) (Section 3.3). The figure shows that the applications achieve much lower L2 cache miss rates (by up to 67%) and execution time (by up to 67%) with Rand-MRU<sub>skw</sub> compared to LRU. Although the very high performance variation shown in the figure is not typical across all applications, it clearly shows that replacement policy is a factor that should be taken into account in designing a cache and modeling cache performance. Furthermore, we also found that a majority (10 out of 17) of the Spec2000/NAS applications tested achieve more than 5% lower miss rates under Rand-MRU<sub>skw</sub> versus under LRU on at least one cache size configuration. Note that our observation in Figure 1 is not new. It confirms the observation from other studies that the cache performance is significantly affected by the choice of replacement policies [2, 3, 15, 25].



**Figure 1:** The L2 cache miss rate (a), and execution time (b), for *art*, *ammp* and *cg* under LRU and Rand-MRU<sub>skw</sub> replacement policies. The L2 cache is 8-way, 512-KB, and has 64-B block size (Table 2).

Despite the extent of the performance variation, current analytical cache performance models ignore the impact of

cache replacement policies on performance. They assume a certain replacement policy in their theoretical derivation, such as LRU [7, 8, 10, 18, 19, 20, 22], or fully random [1, 4, 14]. Assuming a LRU or random replacement policies greatly simplifies cache performance modeling, and the assumption was relatively valid for small caches which tend to have a low associativity or are direct-mapped. However, modern lower level caches (L2 and L3 caches) are often highly associative and large, increasing the role of replacement policy in determining cache performance. The lack of an analytical model for cache replacement policy performance introduces several practical limitations. First, without an analytical model for cache replacement policy performance, we lack the understanding of what factors affect cache replacement policy performance. Such an understanding can be very valuable for guiding cache design, or choosing compiler optimizations that avoid generating codes which perform poorly under a certain cache replacement policy. In addition, hardware design trends point toward a more adaptive cache management (such as in [3]), which may include integrating more than one cache replacement policies in a single cache. Such an adaptivity would require the ability to predict the replacement policy that performs better given the characteristics of the applications. In lieu of an analytical model, chip designers have to resort to simulations to model cache replacement policy performance. Since simulations are very time consuming for realistic workloads, they are expensive or impractical to use in some situations, such as for guiding the compilation process or for run-time adaptation.

The contribution of this paper is an analytical cache model that predicts the performance of cache replacement policies. To the best of our knowledge, this is the first attempt to arrive at such a model. The input of our model is the *circular sequence profiling* [8] of each application. The profiling, to be discussed more in Section 4, can be easily collected, and requires very little storage. Only one profiling run is required for each application to generate the predicted performance across different replacement policies. The output of the model is the predicted miss rates for a given application. The model is based on probability theory and utilizes Markov processes to compute each cache access' miss probability. The model uses realistic assumptions and relies solely on the statistical properties of each application's access pattern (i.e. it does not employ any heuristics or rules of thumbs). Replacement policies are represented by a *replacement probability function* (RPF) which specifies the probability of a line in different LRU stack position to be replaced on a cache miss. Many replacement policies, such as LRU, Random, Not Most Recently Used (NMRUx), and Skewed Random, can be specified or approximated by an RPF.

Our model allows an in-depth analysis of how an application's behavior impacts its performance under different cache replacement policies. This analysis is difficult to achieve with simulation models, because they are limited by the discrete and possibly narrow behavior patterns available in current benchmark suites. Behaviors that are not represented by current benchmark suites cannot be analyzed. In addition, any two applications usually differ in more than one factor, making it difficult to isolate any particular factor's contribution to performance. The model achieves significant advantages compared with trace simulations in term of running time and storage overhead for studying cache replacement policy performance. The model takes a constant time of less than 0.1 seconds to generate a miss rate estimate on an Intel Xeon 2.0-GHz processor platform, com-

pared to  $\Theta(\text{number of events})$  run time needed by a trace simulation, which is in the order of hours-days for realistic workloads. The model also requires a small and constant storage overhead for the profiling information, compared to  $\Theta(\text{number of events})$  in a trace simulation.

We validate the model by comparing the predicted miss rates of seventeen Spec2000 and NAS benchmark applications against cycle-accurate execution-driven simulations. The model is very accurate, achieving a prediction error (the absolute difference between the predicted and simulated miss rates) of 1.41% on average, and 20% in the worst case. There are only 14 out of 952 validation points in which the prediction errors are larger than 10% across all experiments with different cache sizes, associativities, and four replacement policies. Finally, to illustrate one possible practical use of the model, we present a case study that analyzes the relationship between cache access patterns of an application with its performance under different replacement policies. The case study reveals that temporal reuse patterns of applications play a major part in deciding how they perform under different replacement policies.

The rest of the paper is organized as follows: Section 2 describes related work, Section 3 presents the analytical model, Section 4 describes the profiling mechanism, Section 5 details the evaluation setup, and Section 6 presents validation results and the case study. Finally, Section 7 summarizes the findings.

## 2. RELATED WORK

Many cache performance models have been proposed in the past. They are used for guiding compiler optimization [7, 10, 18], evaluating cache design space [1, 9, 19], studying the impact of access patterns or program behaviors [4, 14, 20], studying the impact of context switches in time-shared processors [22], and studying the impact of cache sharing on Chip Multi-Processor (CMP) [8]. However, none of above models can predict the performance of different cache replacement policies. In fact, they assume a certain replacement policy such as LRU [7, 8, 10, 18, 19, 20, 22] or fully random [1, 4, 14], which serves as a base for their models' theoretical derivation. To the best of our knowledge, our study is the first attempt to arrive at a model that predicts cache replacement policy performance.

Some existing models are based on stack distance (or reuse distance) profiling [4, 7], which can only model LRU replacement policy. In order to model other replacement policies, we need a profiling technique that can capture an application's temporal reuse pattern in greater details. For that reason, we use circular sequence profiling proposed by Chandra et al. [8]. However, our model is completely different than that of [8]. While they model the impact of cache sharing on Chip Multi-Processor assuming a cache with the LRU replacement policy, we use circular sequence profiling to predict the cache performance across different replacement policies.

## 3. THE ANALYTICAL CACHE MODEL

This section presents our cache prediction model. It presents the scope and assumptions of the model (Section 3.1), overview of the model (Section 3.2), basic definitions (Section 3.3), model input (Section 3.4), and the model (Section 3.5).

### 3.1 Scope and Assumptions

**Scope.** Although our model can be applied to L1 or lower level caches (such as L2 or L3 caches), we only evaluate the performance of the L2 cache. L1 cache usually has a low

cache associativity, and therefore the performance variation of different cache replacement policies is typically small. On the other hand, lower level caches typically have a high cache associativity, and the difference in performance between different replacement policies can be large.

The class of replacement policies that the model covers are ones in which their behavior can be represented statistically with a Replacement Probability Function or RPF (Definition 5), including many popular cache replacement policies such as Least Recently Used (LRU), Random, Not  $x$  Most Recently Used (NMRU $x$ ), and various Skewed Random policies. Replacement policies that are PC-dependent or address-dependent are less common due to their complexity in hardware implementation and are not modeled. In addition, some replacement policies cannot be directly represented by a single RPF. However, Section 3.3 shows that in such a case, multiple RPFs can be used to represent their range of behavior.

**Assumptions.** We choose to summarize an application’s temporal behavior with a single profile. Although applications may exhibit changes in their temporal behavior over time, exhibiting different application phases, in practice we find that the average behavior is sufficient to produce an accurate cache miss rate prediction. This is an implementation choice rather than a fundamental limitation since the model can predict phase-specific performance of the application if phase-specific profiles are input to the model.

We use a per-cache circular sequence profile in conjunction with using prime modulo cache indexing [13], to reduce the variation of accesses across cache sets. This is not a limitation of the model because if it is needed, the model can predict the miss rate of each cache set separately using set-specific profiles. We choose a single profile for the entire cache to keep the profiling very simple.

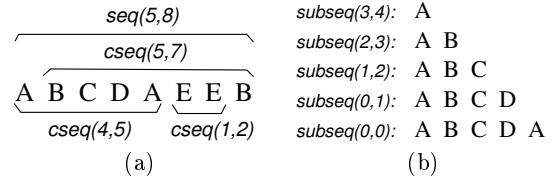
### 3.2 Model Overview

We use the *circular sequence profile* [8] of an application to capture its temporal reuse behavior. We also use a *replacement probability function* (RPF) to model the statistical behavior of a replacement policy. To predict the application’s miss rate under the replacement policy of interest, the model takes several steps. First, for each possible circular sequence that may occur in the application, it computes the probability that the circular sequence’s target access is a cache miss. To compute such probability, the model reconstructs the circular sequence starting from a subsequence that only includes the first access of the circular sequence. We then use a Markov model that keeps track of the current state of the target address (e.g. the target address’ position in the LRU stack) and provides transition probabilities to all possible new states. The transition probabilities take the RPF into account. At each iteration, we include one additional access from the circular sequence into the subsequence, while the Markov model records its new state and transition probability as a result of the inclusion. We repeat the iteration until we have fully included all accesses from the circular sequence. At that time, we check the state of the target address and determine the probability that the target access is a cache miss. Secondly, we use the distribution of circular sequences from the application’s circular sequence profile to compute its overall miss rates, by summing up the miss probability of each circular sequence. The outcome of this step is a polynomial expression of the predicted cache miss rate. The final step is to use Newton-Raphson polynomial root finding algorithm to derive the predicted miss rate for the application.

### 3.3 Basic Definitions

**DEFINITION 1.** : A **sequence** of accesses of an application, denoted as  $seq(d, n)$ , is a series of  $n$  cache accesses to  $d$  distinct block addresses, where all the accesses map to the same cache set. A **circular sequence** of accesses from an application, denoted as  $cseq(d, n)$ , is a special case of  $seq(d, n)$  where the first and the last accesses are to the same block address, and there are no other accesses to that address [8].

For a sequence  $seq(d, n)$ ,  $n \geq d$  necessarily holds. For a circular sequence  $cseq(d, n)$ ,  $n \geq d + 1$  necessarily holds. In a sequence, there may be several, possibly overlapping, circular sequences. The relationship of a sequence and circular sequences is illustrated in Figure 2a. In the figure, there are eight accesses to five different block addresses that map to a cache set, and three circular sequences in a sequence.



**Figure 2:** Illustration of different types of sequences: the relationship between a sequence and circular sequences (a) and subsequences of a circular sequence (b).

In addition to the sequence and circular sequence definitions from [8], we add several new definitions:

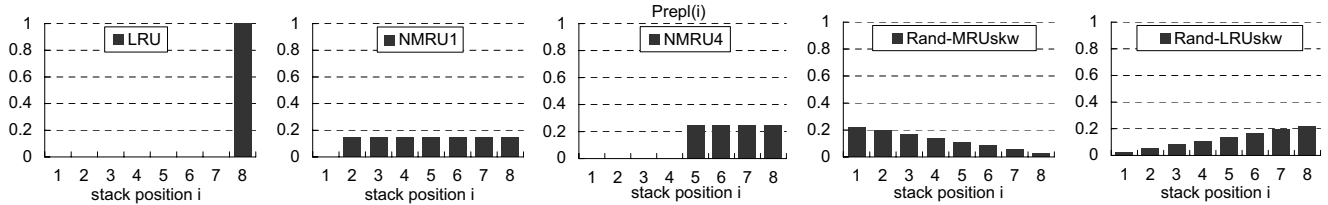
**DEFINITION 2.** : A **target access** of a circular sequence is the last access in that circular sequence. A **target block** is the block address that is referenced by the target access of a circular sequence.

For the circular sequence “A B C D A” in Figure 2a, the target access is the second access to block A, and A is the target block. The relevance of the target access definition to our modeling purpose is that each cache access is a target access of a circular sequence.<sup>1</sup> As pointed out by Chandra et al. [8], it is easy to deduce that for an  $A$ -way associative LRU cache, the target access of a circular sequence  $cseq(d, n)$  results in a cache miss if  $d > A$ , or a cache hit if  $d \leq A$ . Such a property is no longer true for replacement policies in general, where we need to compute the probability that the target access of each circular sequence results in a cache miss.

**DEFINITION 3.** : A **subsequence** of a circular sequence  $cseq(d', n')$ , denoted as  $subseq(d, n)$ , consists of the first  $n' - n$  accesses of the circular sequence, to  $d' - d$  distinct addresses, where  $n \leq n'$  and  $d \leq d'$ .

Therefore, for a  $subseq(d, n)$ ,  $n$  denotes the number of remaining accesses in the circular sequence  $cseq(d', n')$  that are not included in the subsequence, and  $d$  denotes the number of remaining distinct addresses in the circular sequence that are not included in the subsequence. Figure 2b illustrates various subsequences of the circular sequence “A B C D A”.

<sup>1</sup>The first access of a block address is an exception. Such an access will always result in a miss regardless of the replacement policy used. Thus, the number of such accesses is constant across different replacement policies.



**Figure 3: Replacement probability functions (RPF) of several replacement policies assuming an 8-way associative cache.**

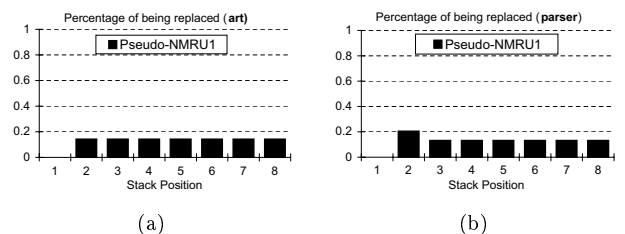
DEFINITION 4. : An access is **distinct** w.r.t. (with respect to) a sequence if the block address that is accessed does not appear in the sequence. An access is **non-distinct** w.r.t. a sequence if the block address that is accessed appears as one of the accessed address in the sequence.

DEFINITION 5. : A **Replacement Probability Function (RPF)**, denoted as  $P_{repl}(\cdot)$ , is a probability function, where each  $P_{repl}(i)$  specifies the probability of a cache block on the  $i^{th}$  LRU stack position to be replaced on a cache miss to the same set.<sup>2</sup>

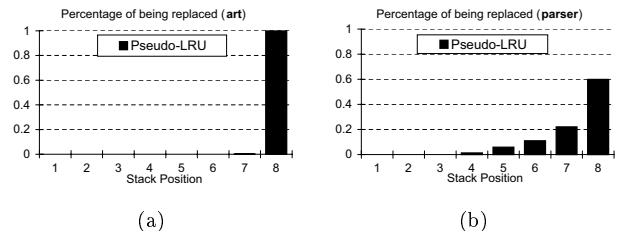
Figure 3 shows the RPFs of several cache replacement policies for an 8-way associative cache. Since LRU only replaces a block that is the least recently used,  $P_{repl}(8) = 1$ , while  $P_{repl}(i) = 0$  for  $i \in [1, 7] \cup [9, \infty)$ . We define NMRUx as a policy that replaces a block from the blocks that are not the  $x$  most recently used blocks, with equal probability. For example, in NMRU4, the four most recently used blocks cannot be replaced, while the others can be replaced with an equal probability. By definition, a fully random replacement policy is equal to NMRU0, and LRU replacement policy is equal to NMRU7. Rand-MRUskw is a random replacement policy which is linearly skewed toward replacing more recently used blocks with a higher probability than less recently used blocks. Rand-LRUskw is a random replacement policy which is linearly skewed toward replacing less recently used blocks with a higher probability than more recently used blocks.

Although there is a large number of replacement policies that can be represented by the RPF, it is not the goal of this paper to perform an exhaustive performance comparison of a large number of replacement policies. Hence, our discussion and evaluation will focus on ones shown in Figure 3. Note that an RPF merely summarizes the statistical behavior of a replacement policy but does not dictate a particular implementation of the policy. In particular, the implementation of a replacement policy may not maintain the LRU stack information shown in its RPF. For example, one way to implement NMRU1 without the keeping LRU stack information is to use an XOR-based implementation, in which an *MRU-way register* is kept per cache set to record which way in the set was most recently accessed. In addition, it keeps a single 3-bit *miss counter* for the cache that counts from 001, 010, 011, ..., 111, and back to 001, 010, ..., skipping “000”. On a cache miss in a particular set, the way selected for replacement is determined by XOR-ing the set’s MRU-way register and the cache’s miss counter. Since the counter skips the value “000”, the selected way for replacement is guaranteed not to be the way that was most recently accessed. Figure 4

shows the actual per-application RPF obtained through simulation, showing the best fit case (art) and the worst fit case (parser), compared to the NMRU1’s theoretical RPF. The figure shows that even though the XOR-based implementation does not keep LRU stack information, it statistically approximates NMRU1’s RPF really well.



**Figure 4: Per-application RPF for XOR-based NMRU1 implementation obtained through simulation, showing the frequency distribution of L2 cache lines replaced from various LRU stack positions. The LRU stack is maintained only for obtaining the RPF, and is not used for cache replacement decisions.**



**Figure 5: Per-application profiled RPF for pseudo-LRU policy, showing the frequency distribution of L2 cache lines replaced from various LRU stack positions.**

In addition, we also investigate whether existing non-LRU stack replacement policies can be represented by RPFs. For this purpose, we evaluate a binary-tree replacement policy from a patent, described as pseudo-LRU by Zoubi et al. [2], and mentioned in the implementation of IBM Power4 processors [11]. We show the pseudo-LRU RPFs for art and parser in Figure 5 that are collected through simulation. We found that for a majority of applications, the RPF closely resembles that of LRU (much like art’s RPF in Figure 5a). However, in some cases, its RPF shows a more scattered distribution (Figure 5b), but still not too different from LRU. In the case where a replacement policy’s behavior cannot be represented by a single RPF, it can be represented by multiple RPFs. For example, we can represent the pseudo-LRU with the two RPFs in Figure 5, or more if needed. Each RPF

<sup>2</sup>Necessarily,  $P_{repl}(i) \geq 0$  for  $i = 1 \dots A$ ;  $P_{repl}(i) = 0$  for  $i = A + 1, \dots, \infty$ , and  $\sum_{i=1}^A P_{repl}(i) = 1$ .

will result in a predicted miss rate. The predicted miss rates collectively form sample points from which the range of miss rates can be statistically estimated.

### 3.4 Input from Profiling

Since circular sequence profiling is not our contribution, we will describe it separately in Section 4. From modeling point of view, circular sequence profiling provides the number of occurrences of a circular sequence for each value of  $n$ 's and  $d$ 's, denoted as  $N(cseq(d, n))$ . However, to simplify the model, we use the average value of  $n$  (i.e.,  $\bar{n}$ ), instead of treating it as a random variable (Section 3.5.1). Thus, we only need to count and store the total number of circular sequences and  $\bar{n}$  for each  $d$ . Therefore, the profiling information size only depends on the size of  $d$ , which we limit to three times the associativity that we want to predict for. This limits the profiling information storage to less than 1 Kbytes per application.

### 3.5 Prediction Model

#### 3.5.1 Basic Prediction Steps

Let  $d$  denote the number of distinct addresses and  $n$  denote the number of accesses in a circular sequence  $cseq(d, n)$ . Let  $d_{max}$  and  $n_{max}$  denote the maximum number of distinct addresses and number of accesses that our model considers, respectively. Let  $A$  denote the cache associativity. Let  $P_{miss}$  denote the predicted miss rate. If we assume that the cache accesses have an identical miss probability,  $P_{miss}$  also denotes the probability of a cache access results in a cache miss.

The model predicts the miss rates of replacement policies using the following basic steps:

1. For each of  $d = 1 \dots d_{max}$ , compute the weighted average of  $n$  (i.e.  $\bar{n}$ ) by considering the distribution of  $cseq(d, n)$  collected through circular sequence profiling. From this point on, we use  $cseq(d, \bar{n})$  in place of  $cseq(d, n)$ .
2. For each circular sequence  $cseq(d, \bar{n})$ , compute  $P_{miss}(cseq(d, \bar{n}))$ : the probability that its target access (i.e. the last access) is a cache miss. This step takes into account the given replacement policy's RPF. The result of this step is an  $\bar{n}^{th}$  degree polynomial function of  $P_{miss}$ , i.e.  $P_{miss}(cseq(d, \bar{n})) = \mathcal{P}^{\bar{n}}(P_{miss})$ .
3. Compute the total probability of cache misses ( $P_{miss}$ ) by summing each individual  $P_{miss}(cseq(d, \bar{n}))$  over all circular sequences. If  $m$  is the maximum value of  $\bar{n}$  for all  $d$ 's, the result of this step is an  $m^{th}$ -degree polynomial expression:  $P_{miss} = \mathcal{P}^m(P_{miss})$ .
4. Solve the expression  $\mathcal{P}^m(P_{miss}) - P_{miss} = 0$  using a polynomial root finding technique, such as the Newton-Raphson algorithm [5]. The root,  $P_{miss}$ , is the predicted L2 cache miss rates.
5. Return to Step 3 for each application to predict its miss rate under the same replacement policy. Return to Step 2 for each different replacement policy.

Note that since Step 2 only depends on the replacement policy and does not depend on the application's profile, we perform the step off-line to pre-compute the resulting polynomial function of  $P_{miss}$  (i.e.,  $\mathcal{P}^{\bar{n}}(P_{miss})$ ) for all possible values of  $d$  and  $\bar{n}$ . As a result, for each application, we only need to perform Step 3 and 4 using the pre-computed polynomial function. We will now describe how each step is performed.

#### Step 1: Compute $\bar{n}$ for each $d$

$\bar{n}$  is computed by taking an average over all possible values of  $n$  for each value of  $d = 1 \dots d_{max}$ :

$$\bar{n} = \frac{\sum_{n=d+1}^{n_{max}} (N(cseq(d, n)) \times n)}{\sum_{n=d+1}^{n_{max}} N(cseq(d, n))} \quad (1)$$

$N(cseq(d, n))$  is obtained through circular sequence profiling. In implementation, we set  $n_{max}$  to 200 because  $n > 200$  is very rare. We also set  $d_{max}$  to 3 times the cache associativity ( $A$ ), because the target access of a circular sequence with  $d > 3A$  is most likely a cache miss regardless of the replacement policies used. More precisely, we assume that  $N(cseq(d, n)) = 0$  for  $n > 200$ , and lump  $\sum_{d=3A+1}^{\infty} N(cseq(d, n))$  into  $N(cseq(3A+1, n))$ .

#### Step 2: Obtain the Expression for $P_{miss}(cseq(d, \bar{n}))$

Step 2 is a very important component of the model. Before getting to the specifics of the step, it is useful to mention several conventions and a definition. First, we refer to the MRU entry in the LRU stack as the top of the stack (position 1), and LRU entry as the bottom of the stack (position  $A$ , where  $A$  = the cache associativity). A cache block  $X$  is said to be positioned higher in the stack than a cache block  $Y$  if the stack position of  $X$  has a smaller number than the position of  $Y$ .

When a cache block  $X$  at stack position  $i$  is accessed, then each block in the  $(j-1)^{th}$  stack position is moved to the  $j^{th}$  position, where  $j = 1 \dots i$ . Then  $X$  is placed at the top of the stack. When there is a cache miss to a block  $X$ , a block in the cache is selected to be replaced according to the replacement policy. If the replaced block is in stack position  $k$ , each block in the  $(j-1)^{th}$  position is moved to the  $j^{th}$  position, where  $j = 1 \dots k$ . Then  $X$  is placed at the top of the stack.

**DEFINITION 6.** : For a subsequence  $subseq(d, n)$  of a circular sequence  $cseq(d', n')$ , a **state** is a tuple  $(d, n, p)$  where  $p$  denotes the current stack position of the target block.

**Basic Approach.** To compute  $P_{miss}(cseq(d, \bar{n}))$ , we employ a set of Markov models where each model consists of a set of states and state transition probabilities. We begin from the smallest (initial) subsequence of a circular sequence, and iteratively add an access to the subsequence until the subsequence becomes the entire circular sequence. At each iteration, we track the state after adding an access and compute the state transition probabilities. To reconstruct how a circular sequence  $cseq(d', n')$  is formed, an *initial subsequence*  $subseq(d' - 1, n' - 1)$  that only contains the first access of the circular sequence is formed. The initial state that corresponds to the initial subsequence is  $(d' - 1, n' - 1, 1)$ , because the target block would be located at the top of stack after the first access to the block (hence,  $p = 1$ ). At each iteration, an access from the circular sequence is added to the subsequence, until a *terminal subsequence*  $subseq(0, 1)$  is reached, where only the target access of the circular sequence has not been included in the subsequence. The state that corresponds to the terminal subsequence is  $(0, 1, p)$ , where  $p$  may range from 1 to  $A$ .

**State Transition Diagram.** Figure 6 shows the current state  $(d, n, p)$  in a bold circle, and the possible resulting new states after an access is added to the current subsequence. There are eight transition cases that lead to one of five new states. Each transition case is a combination of several events. *Dist* and *NoDist* events indicate that the access is either distinct or non-distinct, respectively. *Miss* and *Hit*

events indicate whether the access results in a cache miss or a cache hit, respectively. *Rp* and *NoRp* events indicate whether the access results in the replacement of the target block or not, respectively. Finally, *Shift* and *NoShift* events indicate whether the access results in the target block to be shifted down in the LRU stack or not, respectively.

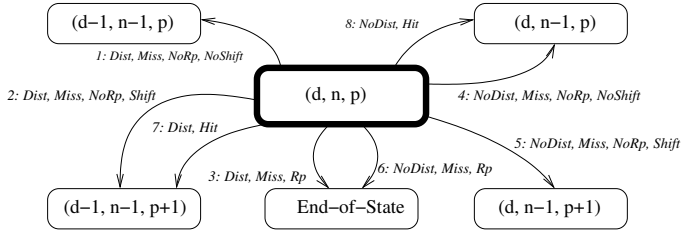


Figure 6: State transition diagram of the model.

After adding an access to the current subsequence, the number of remaining accesses that has not been included in the subsequence reduces by 1, resulting in states with  $n - 1$  accesses (Case 1, 2, 4, 5, 7, and 8 in Figure 6). In addition, when the access is distinct, the number of distinct accesses that has not been added to the subsequence reduces by one, resulting in a state with  $d - 1$  distinct accesses (Case 1, 2, and 7). Otherwise, the number of remaining distinct accesses is unchanged (Case 4, 5, and 8). When an access results in the replacement of the target block, the state transitions to the *End-of-State* (EOS) because from this point on, the target block is no longer in the cache and the target access will definitely result in a cache miss (Case 3 and 6). Finally, if the cache access results in shifting the target block into a lower stack position, the new state indicates position  $p + 1$  (Case 2, 5, and 7). Otherwise, it does not change the stack position of the target block (Case 1, 4, and 8). Note that for Case 7, a distinct access that results in a cache hit indicates that the address that is accessed appears prior to the subsequence. Thus, it can only be in a lower stack position compared to the target block. This causes the target block to be shifted down by one position.

Table 1 details each transition case, the events that corresponds to the case, and the transition probability for each case. The table uses several new notations. Let  $P_{dist}$  denote the probability that a cache access is distinct w.r.t. the subsequence  $subseq(d, n)$ . If we assume that distinct accesses are distributed evenly in the circular sequence, then  $P_{dist} = \frac{d}{n}$ , because there are remaining  $d$  distinct accesses in the circular sequence that are yet to appear out of the remaining  $n$  accesses. Let  $P_{shift}$  denote the probability that a cache miss results in shifting the target block one position down in the LRU stack. This situation can only happen when the miss replaces a block that is positioned lower in the stack than the target block. Therefore,  $P_{shift} = \frac{\sum_{i=p+1}^A P_{repl}(i)}{1 - P_{repl}(p)}$ . Finally, the table also uses  $P_{miss}$  defined in Section 3.5.1 and  $P_{repl}(\cdot)$  defined in Definition 5. In Table 1, we assume that each event type is independent of others. The combination of events' joint probability can then be calculated by multiplying their individual probabilities.

**Special Scenarios.** There are two special scenarios that slightly impact the state transition in Figure 6 and Table 1. The first scenario happens when none of the blocks in the subsequence has been replaced from the cache. In this case, a cache miss is always due to a distinct access, because if it were a non-distinct access, it would have found the block in the cache. Therefore, Case 4, 5, and 6 in Figure 6 are ignored because they are no longer possible. In addition,

the transition probability of Case 8 (*NoDist, Hit*) changes to  $1 - P_{dist}$ , because a cache miss cannot happen in Case 8 (i.e.,  $P_{miss} = 0$ ). This case is identified when the number of distinct accesses from the circular sequence  $cseq(d', n')$  that have been included in the subsequence  $subseq(d, n)$  is equal to the stack position of the target block, i.e.  $d' - d = p$ .

The second special scenario happens when the target block has reached the bottom stack position. In this case, a distinct access in the subsequence must be a cache miss because all addresses accessed prior to the circular sequence must have been shifted out of the LRU stack. Therefore, Case 7 in Figure 6 is ignored because it is no longer possible. In addition, since in this scenario  $p = A$ ,  $P_{shift} = 0$  by definition.

Finally, the first and second special scenarios may be satisfied at the same time because they are not disjoint.

**Final Expression.** Taking into account the general state transition in Figure 6 and the two special scenarios, the probability that the target access of a circular sequence results in a cache miss can be computed by using an inductive probability formula. Figure 7 defines  $S(d, n, p)$ , a recursive function that directly corresponds to the state transition of our model. Note that  $P_{miss}(cseq(d', n'))$  is computed through the recursive function as  $S(d' - 1, n' - 1, 1)$  because we do not need to take into account the first and the last access of the circular sequence. Since  $P_{miss}$  is the only unknown in the recursive function, the final expression of  $P_{miss}(cseq(d', n'))$  will be a polynomial function of  $P_{miss}$ . Finally, under the assumption that event types are independent, Figure 7 indicates that if two replacement policies have same RPF, they will produce the same cache miss rate for an application, because  $P_{repl}(\cdot)$  is the only application-independent variable in the recursive function.

### Step 3: Obtain the Final Expression for $P_{miss}$

Once we have the  $P_{miss}(cseq(d, \bar{n}))$  for all  $d$ 's,  $P_{miss}$  is computed by:

$$P_{miss} = \frac{\sum_{d=1}^{d_{max}} \left( \sum_{n=d+1}^{n_{max}} N(cseq(d, n)) \times P_{miss}(cseq(d, \bar{n})) \right)}{\sum_{d=1}^{d_{max}} \sum_{n=d+1}^{n_{max}} N(cseq(d, n))} \quad (2)$$

For each  $d$ ,  $P_{miss}(d, \bar{n})$  is an  $\bar{n}^{\text{th}}$ -degree polynomial function of  $P_{miss}$ . Therefore, if  $m$  denotes the maximum value of  $\bar{n}$  for all  $d$ 's, the right hand side of Equation 2 is an  $m^{\text{th}}$ -degree polynomial function. In other words,  $P_{miss} = \mathcal{P}^m(P_{miss})$ .

### Step 4: Compute $P_{miss}$ Using Newton-Raphson Method

$P_{miss}$  can be obtained by solving the equation  $\mathcal{P}^m(P_{miss}) - P_{miss} = 0$  using polynomial root finding techniques, such as the Newton-Raphson algorithm [5]. We start by choosing an initial guess for the polynomial root, say  $root^{(0)}$ . For the equation  $\mathcal{F}(P_{miss}) = \mathcal{P}^m(P_{miss}) - P_{miss}$ , at each iteration  $i$ , we compute the new root estimate by:

$$root^{(i)} = root^{(i-1)} - \frac{\mathcal{F}(root^{(i-1)})}{\mathcal{F}'(root^{(i-1)})} \quad (3)$$

where  $\mathcal{F}'(P_{miss})$  is the first derivative of  $\mathcal{F}(P_{miss})$ . This method converges to the root after only a few ( $< 10$ ) iterations. In all the experiments, we always find only one root in  $[0, 1]$  when we choose the initial root estimate as 0.5. Finally, the root (i.e.  $P_{miss}$ ), is the predicted L2 cache miss rate.

## 4. CIRCULAR SEQUENCE PROFILING

This section describes the profiling input required by our analytical model. The profiling mechanism is not a part of our contributions.

**Table 1: State transition probabilities.**

Transition Event	Probability
1: Dist, Miss, NoRp, NoShift	$P_{dist} \times P_{miss} \times (1 - P_{repl}(p)) \times (1 - P_{shift})$
2: Dist, Miss, NoRp, Shift	$P_{dist} \times P_{miss} \times (1 - P_{repl}(p)) \times P_{shift}$
3: Dist, Miss, Rp	$P_{dist} \times P_{miss} \times P_{repl}(p)$
4: NoDist, Miss, NoRp, NoShift	$(1 - P_{dist}) \times P_{miss} \times (1 - P_{repl}(p)) \times (1 - P_{shift})$
5: NoDist, Miss, NoRp, Shift	$(1 - P_{dist}) \times P_{miss} \times (1 - P_{repl}(p)) \times P_{shift}$
6: NoDist, Miss, Rp	$(1 - P_{dist}) \times P_{miss} \times P_{repl}(p)$
7: Dist, Hit	$P_{dist} \times (1 - P_{miss})$
8: NoDist, Hit	$(1 - P_{dist}) \times (1 - P_{miss})$

$$S(d, n, p) = \begin{cases}
 0 & /* \text{Boundary Condition} */ \\
 & \text{if } n = 0 \\
 P_{dist} \times P_{miss} \times (1 - P_{repl}(p)) \times (1 - P_{shift}) \times S(d-1, n-1, p) + \\
 P_{dist} \times P_{miss} \times (1 - P_{repl}(p)) \times P_{shift} \times S(d-1, n-1, p+1) + \\
 P_{dist} \times (1 - P_{miss}) \times S(d-1, n-1, P+1) + \\
 (1 - P_{dist}) \times S(d, n-1, p) + \\
 P_{dist} \times P_{miss} \times P_{repl}(p) & /* \text{Special Scenario 1 only} */ \\
 & \text{if } n > 0, d' - d = p, P < A \\
 P_{dist} \times (1 - P_{repl}(p)) \times S(d-1, n-1, p) + \\
 (1 - P_{dist}) \times (1 - P_{miss}) \times S(d, n-1, p) + \\
 (1 - P_{dist}) \times P_{miss} \times (1 - P_{repl}(p)) \times S(d, n-1, p) + \\
 P_{dist} \times P_{repl}(p) + (1 - P_{dist}) \times P_{miss} \times P_{repl}(p) & /* \text{Special Scenario 2 only} */ \\
 & \text{if } n > 0, d' - d > p, p = A \\
 P_{dist} \times (1 - P_{repl}(p)) \times S(d-1, n-1, p) + \\
 (1 - P_{dist}) \times S(d, n-1, p) + \\
 P_{dist} \times P_{repl}(p) & /* \text{Special Scenario 1 and 2} */ \\
 & \text{if } n > 0, d' - d = p, p = A \\
 P_{dist} \times P_{miss} \times (1 - P_{repl}(p)) \times (1 - P_{shift}) \times S(d-1, n-1, p) + \\
 P_{dist} \times P_{miss} \times (1 - P_{repl}(p)) \times P_{shift} \times S(d-1, n-1, p+1) + \\
 P_{dist} \times (1 - P_{miss}) \times S(d-1, n-1, P+1) + \\
 (1 - P_{dist}) \times P_{miss} \times (1 - P_{repl}(p)) \times (1 - P_{shift}) \times S(d, n-1, p) + \\
 (1 - P_{dist}) \times P_{miss} \times (1 - P_{repl}(p)) \times P_{shift} \times S(d, n-1, p+1) + \\
 (1 - P_{dist}) \times (1 - P_{miss}) \times S(d, n-1, p) + \\
 P_{miss} \times P_{repl}(p) & /* \text{General Scenario} */ \\
 & \text{otherwise}
 \end{cases}$$

**Figure 7: The recursive function that computes  $P_{miss}(cseq(d', n'))$  as:  $P_{miss}(cseq(d', n')) = S(d' - 1, n' - 1, 1)$ .**

**Stack Distance Profiling** [16]. The input to our models is the *circular sequence profile* of each application. Since circular sequence profiling extends *stack distance profiling*, we will first discuss stack distance profiling. A stack distance profile captures the temporal reuse behavior of an application in a fully or set-associative cache [6, 16, 21], and is sometimes also referred to as marginal gain counters [22, 23]. For an  $A$ -way associative cache with LRU replacement policy, there are  $A + 1$  counters:  $C_1, C_2, \dots, C_A, C_{>A}$ . On each cache access, one of the counters is incremented. If it is a cache access to a block in the  $i^{th}$  position in the LRU stack of the set,  $C_i$  is incremented. If it is a cache miss,  $C_{>A}$  is incremented. Applications with regular temporal reuse behavior usually access more recently used data more frequently than less recently used data. In those applications, larger stack distance counters will have smaller values compared to smaller stack distance counters. A stack distance profile can be obtained statically by the compiler [6], by simulation, or directly in hardware at run-time through simple hardware extensions [23]. Finally, note that the stack distance counter  $C_d$  corresponds to the number of occurrences of a circular sequence with  $d$  distinct address, denoted as  $N(cseq(d, *))$ .

**Circular Sequence Profiling** [8]. A stack distance profiling collects the number of occurrences of a circular sequence with  $d$  distinct addresses, regardless of the number of total accesses  $n$  in a circular sequence. In contrast, a circular sequence profiling collects the number of occurrences of a circular sequence for each  $n$  accesses to  $d$  distinct addresses,

denoted as  $N(cseq(d, n))$ . Performing the profiling with simulation is very simple. Each cache block is augmented with an access counter associated with  $n$ , which is incremented every time another block in the same set is accessed. If the cache block itself is accessed, the access counter represents  $n$  in the circular sequence, while the current stack position of the block represents  $d$  in the circular sequence. Hence,  $N(cseq(d, n))$  is incremented.

Although the purpose of the circular sequence profile is to capture an application's temporal reuse patterns, it is also affected by certain system parameters. For example, since L2 cache accesses are filtered by the L1 instruction and data caches, the L1 cache organization affect the circular sequence profile. Another example is that since prefetches would need to be treated as regular cache misses/accesses in the profile, prefetching mechanisms affect the circular sequence profile. Thus, for the model to be accurate, the circular sequence profile needs to be re-collected when certain system parameters that could affect it change.

## 5. EVALUATION METHODOLOGY

**Simulation Environment.** The evaluation and validation are performed against SESC, a cycle-accurate execution-driven simulator [12]. Note that we cannot validate the model against a real microprocessor system because current systems do not support circular sequence profiling. Table 2 shows the base configuration of the simulated architecture. The processor core is a state of the art out-of-order superscalar processor. To observe the impact of L2 cache param-

ters on the accuracy of the model, we vary the L2 cache sizes from 32 Kbytes to 4096 Kbytes and the L2 cache associativity from 4 to 8 for validation purposes. Note that although only the L2 cache miss rates are needed for validation against the model, we choose a detailed execution-driven timing simulation instead of a trace simulation because we also want to investigate the impact of difference replacement policies on execution time in addition to such impact on miss rates.

**Table 2: Parameters of the simulated architecture. *RT* stands for round-trip time from the processor.**

PROCESSOR
1 core, 4-issue dynamic. 3.2 GHz. Int, fp, ld/st FUs: 3, 2, 2 Branch penalty: 13 cycles. Re-order buffer size: 152
MEMORY
L1 Inst, Data: each WB, 32 KB, 4 way, 64-B block, RT: 2 cycles, LRU replacement L2 unified : WB, 32/64/128/256/1024/2048/4096 KB, 4/8 way, 64-B block, RT: 12 cycles, prime modulo indexed [13]. RT memory latency: 362 cycles Memory bus: split-transaction, 8 B, 800 MHz, 6.4 GB/sec peak

**Applications.** We choose seventeen benchmarks from Spec2000 benchmark suite (mcf, mesa, art, twolf, bzip2, equake, swim, psi, gap, mgrid, parser, and ammp), and from NAS benchmark suite (cg, ft, is, lu, and sp) [17]. Benchmarks written in Fortran90 are not included due to the limitation of our compiler infrastructure. In addition, benchmarks that show less than 5% miss rates when L2 cache size is 1024-KB (gzip, vpr, gcc, eon, crafty, perlbnk, and vortex) are excluded from our reporting because due to the very small miss rates, our model predicts them very accurately. However, despite the accurate prediction, the benchmarks have not really stressed the model since the miss rates are clustered around zero. Hence, we focus on validating the model against benchmarks that have substantial miss rates. For SPEC2000 benchmarks, we use the *ref* input sets, while for the NAS benchmarks, we use the *class A* input sets. For all the applications, we fast forward the first one billion instructions and simulate the next two billion instructions to keep the simulation time reasonable.

## 6. EVALUATION RESULTS

In this section, we will discuss the model validation in Section 6.1 and a case study that analyzes the relationship between cache access patterns and the performance of different replacement policies in Section 6.2.

### 6.1 Model Validation

Throughout the validation, we measure the prediction error as the *absolute value* of the difference between the L2 miss rate obtained through simulation, and the L2 miss rate predicted by our model. The miss rates are local miss rates to the L2 cache (i.e., the number of L2 misses divided by the number of L2 accesses), and are expressed in percentages. Hence, the prediction errors are also expressed in percentages. The reason for using the absolute miss rate difference as the prediction error is because it is a very intuitive measure and that it typically has a linear relationship with the average number of cycles taken to execute one instruction [24], a very common performance metric.

We input each application’s circular sequence profiles under eight different cache sizes and two different cache associativities into the model. For each profile input, the model computes the predicted miss rates for four different replacement policies: NMRU4, NMRU1, Rand-LRU<sub>skw</sub> and Rand-MRU<sub>skw</sub>. LRU is excluded from validation because its miss rates can be computed directly from the circular sequence

profiles without using our model. The predicted miss rates are then compared against the miss rates obtained through the detailed simulations.

We then sort the applications based on their prediction errors, and the full validation results on 8-way associative caches for five benchmarks with the *largest* errors (is, mcf, twolf, bzip2, and sp) are shown in Figure 8. In the figure, the cache size is varied from 32 Kbytes to 4096 Kbytes, while keeping the cache associativity at 8. The figure shows that the predicted and simulated miss rates are very close across different cache sizes. The largest prediction error is 20%, which occurs in mcf when cache size is 2048 Kbytes under Rand-MRU<sub>skw</sub> replacement policy. However, large prediction errors are rare: the errors are larger than 10% for only 14 out of 952 validation points across different cache sizes, associativities, and four replacement policies. Other benchmarks not shown in the figure have even smaller prediction errors.

Table 3 summarizes all validation results for the four replacement policies on 8-way and 4-way associative caches. Each cell in the table (except those in the last two rows) represents the arithmetic mean of the prediction errors across all cache sizes, for a specific benchmark, a replacement policy, and a cache associativity. Hence, each graph in Figure 8 corresponds to a cell in the table. Note that since NMRU4 replacement policy is not applicable on a 4-way associative cache, it is not on the table. The last two rows in the table shows various arithmetic means across applications under a single replacement policy, or the arithmetic mean across all replacement policies under a single cache associativity. The table further confirms that our model is very accurate. In most cases, the mean absolute error is less than 3%. The errors are comparable for the different replacement policies, except for Rand-MRU<sub>skw</sub> which tends to show slightly larger errors.

**Sources of Inaccuracy.** Some of the model’s inaccuracy can be attributed to the variation of accesses across cache sets. Despite using prime modulo indexing to reduce such variation, some benchmarks (bzip2 and twolf) still show a very large variation in cache accesses across different cache sets for large caches. Since such variation tends to increase with the number of cache sets, the model’s accuracy tends to decrease with larger cache sizes. While using set-specific profiles can easily eliminate such inaccuracy, we choose not to do that to keep the profiling very simple, and because such inaccuracy introduce small errors in very few applications. Some inaccuracy may also come from the fact that we assume that the number of accesses in a circular sequence can be represented accurately by its expected value ( $\bar{n}$  in Section 3.5.1). Relaxing this assumption requires treating  $n$  as a random variable, which would increase the complexity of the model. Our assumption that each event type is independent of others (Section 3.5.1) may also introduce slight inaccuracy. Finally, both Figure 8 and Table 3 indicate that the prediction is relatively less accurate for Rand-MRU<sub>skw</sub> replacement policy compared to other replacement policies. We believe that the reason is because the assumption that accesses to distinct addresses are distributed evenly in the circular sequence (Section 3.5.1) is not entirely accurate for applications that tend to access the same block address in a burst (if such burst cannot be filtered by a small L1 cache). For such applications, the target block under the assumption leaves the MRU position slightly earlier than in reality. As a result, under Rand-MRU<sub>skw</sub>, the model slightly underestimates the probability that the target block is replaced and missed, explaining why in Figure 8 the predicted miss rates under the Rand-MRU<sub>skw</sub> tend to be lower than

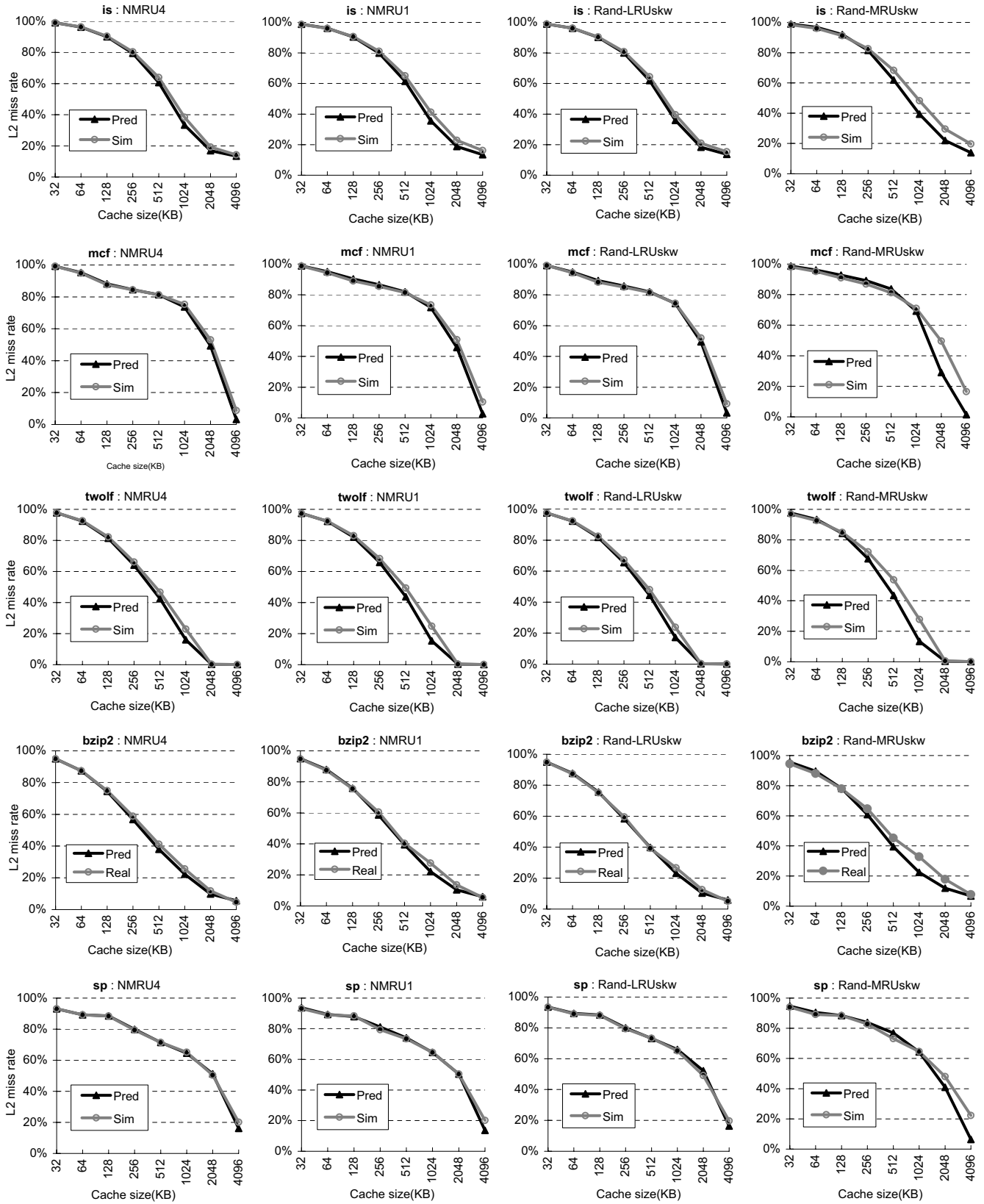


Figure 8: The predicted (Pred) versus simulated (Sim) miss rate for five applications with the largest prediction errors (is, mcf, twolf, bzip2 and sp) for NMRU4, NMRU1, Rand-LRU and Rand-MRU replacement policies. The L2 cache size is varied from 32k to 4096k and L2 cache associativity is kept at 8-way.

**Table 3: Summary of validation results. Each value represents the arithmetic mean of prediction errors under 8 different cache sizes.**

Benchmark	8-Way L2 Cache				4-Way L2 cache		
	NMRU4	NMRU1	Rand-LRUskw	Rand-MRUskw	NMRU1	Rand-LRUskw	Rand-MRUskw
ammp	0.75%	1.71%	1.39%	3.08%	1.25%	1.44%	2.43%
apsi	0.43%	1.09%	1.01%	1.79%	0.45%	0.63%	0.99%
art	0.59%	0.94%	0.68%	1.90%	0.68%	0.66%	2.12%
bzip2	1.50%	1.62%	1.14%	3.73%	2.16%	1.66%	5.02%
cg	0.47%	0.77%	0.64%	2.16%	0.36%	0.47%	1.16%
equake	0.10%	0.11%	0.17%	0.39%	0.11%	0.05%	0.20%
ft	0.94%	1.31%	0.89%	2.42%	1.48%	1.08%	2.98%
gap	0.01%	0.01%	0.01%	0.03%	0.01%	0.01%	0.03%
is	1.58%	2.20%	1.42%	3.93%	2.00%	1.52%	4.06%
lu	0.59%	1.98%	1.12%	1.84%	0.60%	0.64%	1.55%
mcf	1.51%	2.27%	1.40%	5.70%	1.47%	1.08%	3.86%
mesa	0.18%	1.27%	0.78%	3.92%	0.77%	0.83%	2.75%
mgrid	0.10%	0.58%	0.30%	0.80%	0.51%	0.45%	1.24%
parser	0.82%	0.87%	0.66%	2.24%	1.18%	0.92%	3.16%
sp	0.78%	1.31%	1.05%	3.78%	0.63%	0.76%	3.36%
swim	1.01%	1.81%	1.42%	2.73%	0.62%	0.62%	1.29%
twolf	1.76%	2.33%	1.60%	3.92%	2.19%	1.65%	4.86%
Avg	0.77%	1.31%	0.92%	2.61%	0.97%	0.85%	2.42%
	1.40%				1.41%		

the simulated miss rates. Overall, despite the sources of inaccuracy, our model still achieves small prediction errors even for applications that have a large variation of accesses (bzip2 and twolf in Figure 8), and even for Rand-MRUskw policy. Across all benchmarks and all cache configurations, the mean prediction error of the model is only 1.41%.

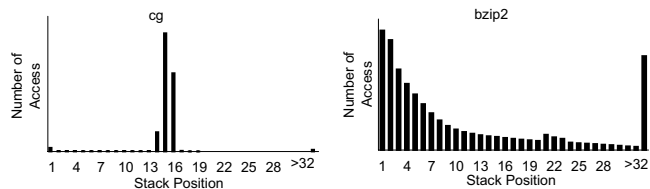
## 6.2 Case Study

To demonstrate one possible practical use of the model, we investigate the relationship between cache access patterns of an application and its performance under different replacement policies. We first motivate the case study with an observation of simulated L2 cache miss rates for various applications and replacement policies in Figure 9. Note that since the applications are quite memory intensive, the L2 cache miss rates translate quite directly into the benchmarks’ execution time (not shown in the figure). Thus, it is important to understand what factors affect the miss rates under different replacement policies.

We group the applications into two: group A consists of benchmarks for which LRU performs worse than other replacement policies, while group B consists of benchmarks for which LRU performs better or the same as other replacement policies. The figure supports several very interesting observations. First, applications in Group A show LRU’s pathological performance cases, in which other replacement policies achieve significantly lower miss rates than LRU (by as much as 67% in ammp, 32% in cg, 18% in art, and 7% in mgrid).<sup>3</sup> However, in Group B, LRU performs slightly better than other replacement policies. Another observation is that the relative performance ranking of the replacement policies in Group A and Group B show a startling contrast. In Group A, Rand-MRUskw performs the best, followed by NMRU1, Rand-LRUskw, NMRU4, and LRU. In Group B, the order is reversed in some cases, with LRU performing

<sup>3</sup>Table 3 shows that the mean prediction errors for applications in group A are very small compared to the difference in miss rates across various replacement policies. Therefore, our model accurately identifies LRU pathological performance cases.

the best, followed by NMRU4, Rand-LRUskw, NMRU1, and finally by Rand-MRUskw.



**Figure 11: Stack distance profiles of cg and bzip2 on a 512-KB 8-way associative L2 cache.**

This “seemingly opposite” behavior of replacement policy performance in Group A versus Group B demand further investigation into what factors trigger such behavior. However, it is not easy to pinpoint the exact causes through simulations alone because the applications differ in many factors (working set size, L2 access frequency, temporal locality, spatial locality, etc.). Fortunately, our model provides a tool for analyzing how the performance of replacement policies is affected by the application’s temporal reuse pattern in isolation from other factors. To perform such an analysis, we look at the stack distance profile shapes of the benchmarks on Group A and Group B. We then generalize the profile into several basic shapes and create three synthetic stack distance profiles corresponding the shapes: unimodal, bimodal, and continuous (Figure 10). Such synthetic stack distance profile shows the frequency of accesses (reuses) to various stack positions (where position 1 is the MRU position, and position A is the LRU position). The continuous stack distance profile is modeled as a geometric progression with the common ratio as its parameter. Figure 11 shows examples of real stack distance profiles from which the generalized shapes are derived. For example, cg shows an approximately unimodal stack distance profile while bzip2 shows an approximately continuous stack distance profile on a 512K cache. We also found some profiles with approximate multi-modal shapes. For such profiles, we choose a bimodal stack distance profile to represent them. Note that the three

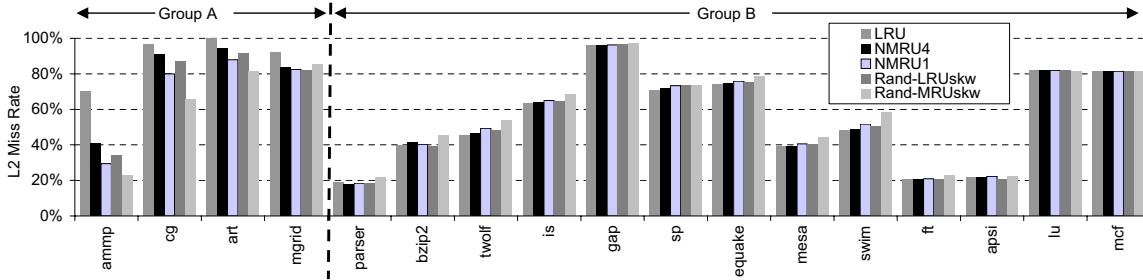


Figure 9: The impact of various replacement policies on applications’ L2 cache miss rates and normalized execution time, when the L2 cache size is 512-KB.

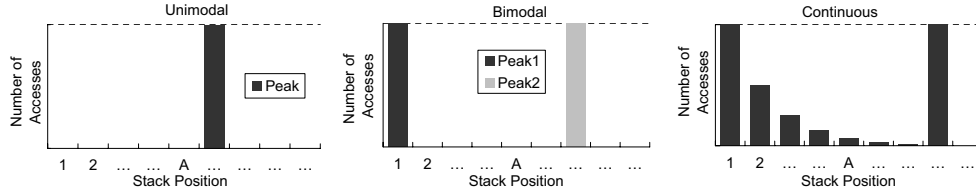


Figure 10: Synthetic stack distance profiles used for analyzing the performance of different replacement policies.

shapes by no means represent all spectrum of cache access patterns. To cover other access patterns, more shapes can be added through the procedure described in this case study. However, it is beyond the scope of this case study to consider an exhaustive list of shapes.

Finally, since the model expects circular sequence profiles as its input, which require the value of  $\bar{\pi}$  for each  $d$ , for simplicity we set and only show results for  $\bar{\pi} = d + 1$ . However, we have also tried setting  $\bar{\pi} = d + 2, d + 3$ , and  $d + 4$  and found that although the miss rates of non-LRU replacement policies are slightly higher, the overall results and trends are similar to that of  $\bar{\pi} = d + 1$ .

Figure 12 shows the miss rates obtained by the model for unimodal profile (a), bimodal profile (b), and continuous profile (c) on an 8-way associative cache. In the x-axis, we vary the peak position of the unimodal profile (with peak = 1, 2, . . .), peak2 position of the bimodal profile (with peak1 = 1 and peak2 = 9, 10, . . .), and the common ratio of the continuous profile (ratio = 0.5, 0.6, . . . , 0.9). The figure shows that the unimodal profile is a pathological performance case for LRU, where the miss rate suffers from a significant jump when peak > 8, while for other replacement policies, the miss rates increase at a slower pace. This is not surprising as it is well known that LRU performs poorly when the working set of an application is slightly larger than the cache size. An interesting observation is that Rand-MRUskw continues to outperform LRU even when the peak’s location (hence the working set size) is four times the cache size. For bimodal profile, LRU also performs worse than other replacement policies, especially compared to Rand-MRUskw. For the continuous profile, LRU performs better than other replacement policies when the common ratio is 0.6 or smaller (indicating a concentrated stack distance profile shape). However, when the stack distance profile is flatter (large common ratios), it is outperformed by other replacement policies.

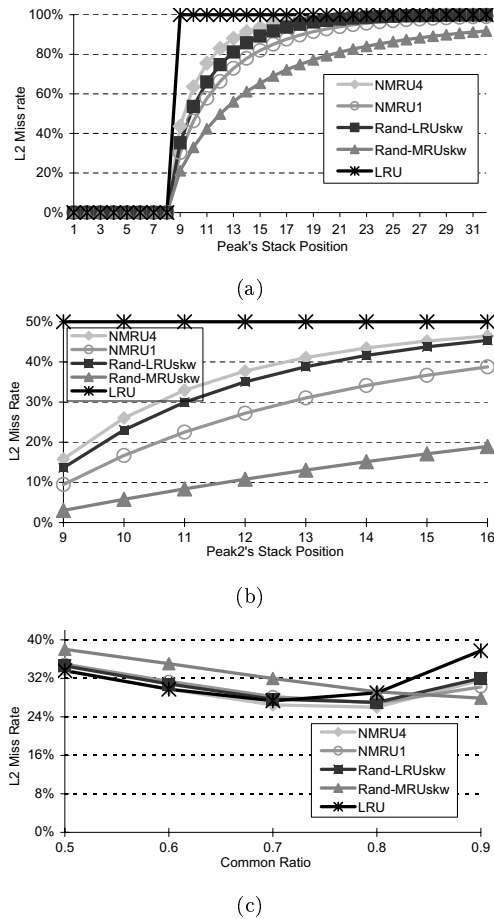
We found that indeed, the four applications in group A which perform poorly on LRU have approximately unimodal stack distance profiles with a peak larger than the cache associativity. The performance ranking of the replacement policies for the unimodal profile obtained by the model ex-

actly matches that in the Group A obtained through simulation: Rand-MRUskw performs the best, followed by NMRU1, Rand-LRUskw, NMRU4, and LRU. This match further validates the accuracy of the model. We also observe that Group B applications’ stack distance profiles are either continuous with small common ratios or weakly continuous. The model shows that for continuous profiles with small common ratios, it is to be expected that LRU outperforms other replacement policies by a *small* margin, which is also confirmed by results for Group B applications in Figure 9. Overall, the model explains that indeed, the temporal reuse patterns of an application reflected through its stack distance profile shape plays a major role in determining how the application would perform under different replacement policies.

**Discussion.** Within the scope of the stack distance profile shapes considered, the case study highlights several implications for chip designers. First, NMRU1 is an attractive replacement policy because it achieves good performance on average across all cases, it is quite robust in the sense that its performance does not degrade so suddenly when the working set of an application slightly exceeds the cache size, and that it can be implemented with low hardware complexity, such as using the XOR-based implementation described in Section 3.3.

Secondly, the case study also motivates chip designers to implement multiple cache replacement policies that can be switched on-demand depending on the situation, especially policies that show the opposite performance behavior such as LRU and Rand-MRUskw (i.e. for almost any situations, one noticeably outperforms the other). If the replacement policy is correctly switched to adapt to a given situation, the potential performance gain can be quite large.

Finally, we note that even in the cases where other replacement policies outperform LRU, there is still a gap between their performance with Belady’s theoretical optimal replacement policy (not shown in this paper). Together with the fact that the classification of synthetic stack distance profiles can be further refined, our model would be very useful for analyzing the performance of replacement policies in future studies.



**Figure 12: The predicted L2 miss rates of various replacement policies for unimodal (a), bimodal (b), and continuous (c) stack distance profiles.**

## 7. CONCLUSIONS

This paper has presented a new analytical model that accurately predicts the miss rates of cache replacement policies for individual applications. The model only requires simple profiling, uses reasonable assumptions without relying on heuristics, and is much faster compared to trace simulations. Validated against a cycle-accurate execution driven simulation on SPEC2000 and NAS benchmarks, the model is accurate across a large number of validation points, achieving an average prediction error of only 1.41%. Finally, our case study reveals that the temporal reuse patterns reflected in the stack distance profile shapes play a major role in determining an application's performance under different replacement policies.

## 8. REFERENCES

- [1] A. Agarwal, J. Hennessy, and M. Horowitz. An analytical cache model. *ACM Trans. on Computer Systems*, 7(2):184–215, 1989.
- [2] H. Al-Zoubi, A. Milenkovic, and M. Milenkovic. Performance Evaluation of Cache Replacement Policies for the SPEC CPU2000 Benchmark Suite. In *Proc. of the 42nd ACM Southeast Conf.*, April 2004.
- [3] E. R. Altman, V. K. Agarwal, and G. R. Gao. A novel methodology using genetic algorithms for the design of caches and cache replacement policy. In *Proc. of the 5th Intl. Conf. on Genetic Algorithms*, pages 392–399, 1993.

- [4] E. Berg and E. Hagersten. StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis. In *Proc. of the IEEE Intl. Symp. on Performance Analysis of Systems and Software*, pages 20–27, 2004.
- [5] L. Bostock and S. Chandler. *Pure Mathematics 2*. Stanley Thornes (Publishers) Ltd., 1979.
- [6] C. Cascaval, L. DeRose, D. A. Padua, and D. Reed. Compile-Time Based Performance Prediction. In *Proc. of the 12th Intl. Workshop on Languages and Compilers for Parallel Computing*, pages 365–379, 1999.
- [7] C. Cascaval and D. A. Padua. Estimating cache misses and locality using stack distances. In *Proc. of the 17th Intl. Conf. on Supercomputing*, pages 150–159, 2003.
- [8] D. Chandra, F. Guo, S. Kim, and Y. Solihin. Predicting Inter-Thread Cache Contention on a Chip Multiprocessor Architecture. In *Proc. of the 11th Intl. Symp. on High Performance Computer Architecture*, pages 340–351, 2005.
- [9] P. G. Emma. Understanding some simple processor-performance limits. *IBM Journal of Research and Development*, 41(3):215–232, 1997.
- [10] S. Ghosh, M. Martonosi, and S. Malik. Cache miss equations: an analytical representation of cache misses. In *Proc. of the 11th Intl. Conf. on Supercomputing*, pages 317–324, 1997.
- [11] IBM. *IBM Power4 System Architecture White Paper*, 2002.
- [12] J. Renaud, et al. SESC. <http://sesc.sourceforge.net>, 2004.
- [13] M. Kharbutli, K. Irwin, Y. Solihin, and J. Lee. Using Prime Numbers for Cache Indexing to Eliminate Conflict Misses. In *Proc. of the Intl. Symp. on High Performance Computer Architecture*, pages 288–299, 2004.
- [14] R. E. Ladner, J. D. Fix, and A. LaMarca. Cache performance analysis of traversals and random accesses. In *Proc. of the ACM-SIAM Symp. on Discrete algorithms*, pages 613–622, 1999.
- [15] W.-F. Lin and S. Reinhardt. Predicting Last-Touch References under Optimal Replacement. *University of Michigan Tech. Rep. CSE-TR-447-02*, 2002.
- [16] R. L. Mattson, J. Gecsei, D. Slutz, and I. Traiger. Evaluation Techniques for Storage Hierarchies. *IBM Systems Journal*, pages 78–117, 9(2), 1970.
- [17] NASA. NAS Parallel Benchmark. <http://www.nas.nasa.gov/Pubs/TechReports/NAReports/NAS-98-009/>, 1998.
- [18] R. W. Quong. Expected i-cache miss rates via the gap model. In *Proc. of Intl. Symp. on Computer Architecture*, pages 372–383, 1994.
- [19] S. Sen, S. Chatterjee, and N. Dumir. Towards a theory of cache-efficient algorithms. *Journal of the ACM*, 49(6):828–858, 2002.
- [20] J. P. Singh, H. S. Stone, and D. F. Thiebaut. A model of workloads and its use in miss-rate prediction for fully associative caches. *IEEE Trans. on Computers*, 41(7):811–825, 1992.
- [21] Y. Solihin, J. Lee, and J. Torrellas. Automatic Code Mapping on an Intelligent Memory Architecture. *IEEE Trans. on Computers: special issue on Advances in High Performance Memory Systems*, 50(11):1248–1266, 2001.
- [22] G. E. Suh, S. Devadas, and L. Rudolph. Analytical cache models with applications to cache partitioning. In *Proc. of Intl. Conf. on Supercomputing*, pages 1–12, 2001.
- [23] G. E. Suh, S. Devadas, and L. Rudolph. A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning. In *Proc. of Intl. Symp. on High Performance Computer Architecture*, pages 117–126, 2002.
- [24] H. J. Wassermann, O. M. Lubeck, Y. Luo, and F. Bassetti. Performance Evaluation of the SGI Origin2000: A Memory-Centric Characterization of LANL ASCII Applications. In *Proc. of Supercomputing*, 1997.
- [25] W. Wong and J.-L. Baer. Modified LRU Policies for Improving Second-Level Cache Behavior. In *Proc. of the Intl. Symp. on High Performance Computer Architecture*, pages 49–60, 2000.