A Modeling Framework for Reuse Distance-based Estimation of Cache Performance

Xiaoyue Pan Bengt Jonsson
Department of Information Technology
Uppsala University
{xiaoyue.pan, bengt.jonsson}@it.uu.se

***Abstract*—We develop an analytical modeling framework for efficient prediction of cache miss ratios based on reuse distance distributions. The only input needed for our predictions is the reuse distance distribution of a program execution: previous work has shown that they can be obtained with very small overhead by sampling from native executions. This should be contrasted with previous approaches that base predictions on stack distance distributions, whose collection need significantly larger overhead or additional hardware support. The predictions are based on a uniform modeling framework which can be specialized for a variety of cache replacement policies, including Random, LRU, PLRU, and MRU (aka. bit-PLRU), and for arbitrary values of cache size and cache associativity. We evaluate our modeling framework with the SPEC CPU 2006 benchmark suite over a set of cache configurations with varying cache size, associativity and replacement policy. The introduced inaccuracies were generally below 1% for the model of the policy, and additionally around 2% when set-local reuse distances must be estimated from global reuse distance distributions. The inaccuracy introduced by sampling is significantly smaller.**

I. INTRODUCTION

Processor caches are critical components of the memory hierarchy that exploit locality to keep frequently-accessed data on chip. Caches can significantly improve performance and reduce energy usage, but this benefit is highly dependent on both the application and the cache characteristics, including cache size and replacement policy. In today's power and energy constrained computer systems, it is important to understand how an application is affected by cache characteristics, e.g., for performing decisions on resource allocation and scheduling, for program transformation and optimization, for performance debugging, etc. This calls for efficient techniques to predict cache performance of applications with a range of cache configurations that differ from current settings.

The prediction methods can be divided into two categories: simulation and analytical models. While simulation [18] [11] [19] [7] [14] [15] guarantees prediction accuracy, the time it takes to simulate a wide range of configurations may be prohibitively slow, especially for applications with a large number of memory accesses. Analytic models in general has the advantage of efficient calculation (sometimes at the cost of accuracy). Moreover, they provide insights into how the cache performance changes with application characteristics and the cache configurations. In this paper, we focus on analytic models.

Existing analytic models for predicting an application's cache performance have two problems: either they can only predict a very limited set of cache configurations (e.g., [8] only handles fully associative LRU cache), or they require the stack distance distribution as the input [18]. Collecting the stack distance distribution requires memory traces, which incurs a 10-1000X slow-down (using tools like PIN [16] and Valgrind [17]), which is unacceptably expensive for many uses in scheduling, optimization, performance tuning, etc.

An efficient alternative is to base cache performance predictions on reuse distance distributions. The *reuse distance* of a memory access is just the number of accesses made since the previous access to the same cache line [3]: this should be contrasted with the *stack distance*, which is the number of *distinct* cache lines accessed since that previous access. Reuse distance distributions are easy to obtain with small overhead on natively running applications *without* collecting memory traces. Berg and Hagersten [4] showed how they can be collected using hardware counters and watchpoint mechanisms and how sampling can reduce the overhead to only 40% without significant loss in accuracy. The watchpoint mechanism monitors when a sampled cache line is reused; the reuse distance is then recorded and the watchpoint is removed. Reuse distance distributions have been used to predict miss ratios for fully-associative caches of varying size with random and LRU replacement policies with good accuracy [3], [8].

Challenge Reuse distance distributions are a very suitable basis for estimating various cache performance metrics, since they can be collected with small overhead on natively running applications. So far, they have been used only for predicting miss ratios for fully-associative caches under random and LRU replacement policies. No techniques have been presented using reuse distance distributions to predict cache performance for replacement policies found in modern processors, such as PLRU and bit-PLRU, nor for taking set associativity into account. Furthermore, the techniques that have been presented for predicting random and for LRU caches are specific to their respective replacement policy [3], [8], and it is not easy to see how they could be re-targeted for other replacement policies. It would be desirable to have a general framework for cache performance prediction from reuse distance distributions, which can be instantiated for different cache configurations, such as various replacement policies and associativities.

Contribution In this paper, we propose a general modeling framework for predicting cache miss ratios for applications in a cache with different configurations. It overcomes the two

problems with existing methods: 1) expensive-to-collect input and 2) only handling a limited set of cache configurations. Our modeling framework only uses the reuse distance distributions as input; and it can estimate the miss ratio in caches with a wide range of configurations, i.e., cache size, cache associativity and replacement policy. Furthermore, it would not be difficult to instantiate our framework for other conceivable replacement policies. To demonstrate how to use the modeling framework, we show how it can be instantiated for Random, LRU, PLRU, and bit-PLRU replacement policies, and also for taking set-associativity into account by first estimating the set reuse distance distribution from the reuse distance distribution, then predicting the cache miss ratio for each set.

The basic idea of our framework is to model the evolution of the cache between two consecutive accesses to the same cache line. The model has the form of a Markov chain, which can be tailored for each specific cache configuration, such as a combination of replacement policy, cache size, and associativity. The transition probabilities between the states in the Markov chain will in general depend on the reuse distribution of the application. By standard analysis of the Markov chain, we compute the probabilities that the second access results in a miss or hit, respectively, from which a prediction of miss ratio is immediately obtained.

We evaluate our prediction framework by using it to predict miss ratios of a set of benchmarks for caches sized 4kB to 64kB and associativities 2-16 with LRU, PLRU and bit-PLRU replacement policies. For a set-associativitive cache, the framework requires the set reuse distance distribution, which can be obtained either during the reuse distance distribution collection with an extra overhead, or estimated from reuse distance distribution with a model we propose. The estimated miss ratio given the actual set reuse distance distribution for all benchmarks and all cache configurations has an average absolute error below 1%. Estimating the set reuse distance adds another 2% error.

In summary, we present for the first time a *general* framework for predicting an application's cache performance with various cache configurations. The specific contributions include

- The modeling framework fulfills two requirements: 1) it can be used to estimate the miss ratio in caches with different sizes, associativities and replacement policies and 2) its input can be collected with a low overhead while no existing cache models satisfy both requirements.
- Due to the framework's *unified* setting, it can be instantiated to other replacement policies. In contrast, existing approaches for estimating miss ratios apply a conceptually different approach for each replacement policy [3], [8], [18], making it unclear how to re-target the method for a new replacement policy.
- We present the first technique for predicting miss ratios for the widely used bit-PLRU replacement policy.
- Our techniques for predicting miss ratios for Random and LRU from reuse distance distributions improve in accuracy over the existing techniques of StatCache [3] and StatStack [8].
- We also propose a model for estimating set reuse distance distributions from reuse distance distributions, which is slightly more accurate than the model of Hill and Smith [13], which has been used in previous such estimations [18].

The models can be used to guide optimization; for example, if our models predict that a smaller cache with fewer ways suffices to keep a low miss ratio, it may be beneficial to switch off part of the cache (if possible) to save energy. By combining our models with other existing techniques, they can also be used to predict miss ratios on multicore systems with shared caches. For example, using a technique to obtain the combined reuse distance distribution of co-scheduled applications, given their individual reuse distance distributions, we can predict miss ratios in shared caches. This has been done for fully associative LRUs in StatCC [10]. Our models can be directly plugged into StatCC to take other realistic caches into account.

The rest of the paper is organized as follows. Section II introduces the assumptions, terms and notations used throughout the paper. Section III presents the general modeling framework for predicting cache miss ratios. Section IV discusses how to convert the reuse distance distribution to the set reuse distance distribution for set-associative caches. Section V instantiates the general modeling framework to the LRU, PLRU and bit-PLRU caches. Section VI shows how we evaluated our models and the results. Section VII discusses related work and Section VIII concludes the paper.

II. Preliminaries

In this section, we introduce the assumptions, terms, and notations throughout this paper. We use C to denote the overall cache size and A to denote the associativity. In this paper, we will consider the policies Random, LRU, tree-PLRU, and bit-PLRU.

Measures of Temporal Locality We introduce metrics of temporal locality in the trace (i.e., sequence of memory accesses) that is generated by a program execution. A trace can be represented as a mapping from an increasing sequence of nonnegative integers (representing positions in the trace) to *accesses*. We represent each access as a pair $\langle a, m\rangle$, where a is an *address* of a cache line, and m is its repetition number. The first access to an address a is represented as $(a, 0)$, the next as $(a, 1)$, etc.

Consider a memory access $\langle a, m\rangle$. Let $\langle a, m\rangle$ be an access in a trace T, with $m > 0$. The *reuse interval* of $\langle a, m\rangle$ in T, denoted $RI_T\langle a, m\rangle$, is the sequence of memory accesses between the access $\langle a, m-1\rangle$ and $\langle a, m\rangle$. The *reuse distance* of $\langle a, m\rangle$, denoted $rd_T(\langle a, m\rangle)$ is the number of memory accesses in $RI_T\langle a, m\rangle$. If $m = 0$, then $rd_T(\langle a, m\rangle)$ is defined as ∞.

Note on terminology: Many other works base cache performance predictions on stack distances. The *stack distance* of $\langle a, m\rangle$, denoted $sd_T(\langle a, m\rangle)$ is the number of *unique* memory accesses in $RI_T\langle a, m\rangle$. For instance, in the trace $T = b\ a\ b\ b\ c\ d\ b\ a$, where a, b, c and d are the accessed cache lines, we have $rd_T((a, 1)) = 5$ and $sd_T((a, 1)) = 3$. Some authors use slightly different terminology. For instance, Sen and Wood [18] use the term *unique reuse distance* for

stack distance, and *absolute reuse distance* for reuse distance. Some works, including [21], use the term reuse distance to denote stack distance.

Based on the reuse distances in a trace, we define the *reuse distance distribution* (*rdd* for short) of T as a function $\mathbf{rdd}_T$ from natural numbers to probabilities, such that $\mathbf{rdd}_T(k)$ is the fraction of all accesses with reuse distance k. We will sometimes think of this as the probability that the reuse distance is k. We will mostly drop the subscript T (i.e., write $\mathbf{rdd}(k)$ for $\mathbf{rdd}_T(k)$) when it is clear which trace is considered. We introduce the notation $\mathbf{rdd}(\geq k)$ for $\sum_{i=k}^{\infty} \mathbf{rdd}(i)$, i.e., the fraction of accesses whose reuse distance is at least k, and write $\mathbf{rdd}(>k)$ for $\mathbf{rdd}(\geq(k+1))$.

It will be convenient to define the *marginal reuse distance distribution* (*mrdd* for short) of a trace as the function **mrdd** from natural numbers to probabilities, defined by $\mathbf{mrdd}(k) = \frac{\mathbf{rdd}(k)}{\mathbf{rdd}(\geq k)}$. That is, $\mathbf{mrdd}(k)$ is the probability that the reuse distance is k, provided that it is at least k. If sequences of accesses in a reuse interval are considered from the beginning, then if k accesses have been considered without encountering a reuse, then $\mathbf{mrdd}(k)$ is the probability that the next access is the reuse. We can conversely get **rdd** from **mrdd** by $\mathbf{rdd}(k) = \mathbf{mrdd}(k)\prod_{i=0}^{k-1}(1-\mathbf{mrdd}(i))$.

Per-Set Measures of Temporal Locality Since replacement decisions are made on a per-set basis, it is important to develop measures of temporal locality that are local to sets. Assume a cache with S sets. We can then repeat the definitions concerning accesses from Section II, but this time specific to the set to which a is mapped.

The *set reuse distance* of $\langle a, m\rangle$, denoted $rd^s(\langle a, m\rangle)$ is the number of memory accesses in $RI\langle a, m\rangle$ that are mapped to the same set as a.

The *set reuse distance distribution* (*set-rdd* for short) is the function $\mathbf{rdd}^s$ for which $\mathbf{rdd}^s(k)$ is the fraction of all accesses with set reuse distance k. Notations $\mathbf{rdd}^s(\geq k)$ and $\mathbf{rdd}^s(>k)$ are as expected.

III. Modeling replacement policies: general framework

In this section, we introduce our general framework for predicting miss ratios for caches under different combinations of replacement policy, size, and associativity, given only the rdd. In Section V, we instantiate this framework for the policies under consideration: LRU, tree-PLRU, and bit-PLRU. In Section IV, we show how rdds can be used to estimate set-rdds, from which we can predict miss ratios for set-associative caches by applying our model for the relevant replacement policy to the set-rdd.

The overall idea of our general framework is to consider a randomly chosen memory access $\langle a, m\rangle$ (with $m > 0$), and estimate the probability that it is a miss. This is done by modeling the evolution of cache during the reuse interval $RI\langle a, m\rangle$ (i.e., the sequence of memory accesses between the previous access $\langle a, m-1\rangle$ to a and $\langle a, m\rangle$), and how this evolution results in a hit or miss for $\langle a, m\rangle$.

Our model has the form of a Markov chain, whose states represent properties of the current cache contents that influence whether the access $\langle a, m\rangle$ will be a hit or a miss. Typically, one state will represent that a has been evicted; the case when a is still in the cache may require several states in order to properly reflect a specific replacement policy. The Markov chain must also have the two states *hit* and *miss*, which are entered when $\langle a, m\rangle$ occurs. The Markov chain is completed, by assigning an initial probability distribution over its states to model the situation immediately after the previous access $\langle a, m-1\rangle$, and transition probabilities to model how this distribution evolves with each access in the reuse interval.

Given the Markov chain, we analyze how the probability distribution over its states evolves during the reuse interval. After the reusing access $\langle a, m\rangle$, this distribution is nonzero only for *hit* and *miss*, from which we obtain the miss ratio.

Let us explain more precisely how our Markov model is constructed for a specific replacement policy. Let $\langle a, m\rangle$ be the considered access. For $i \geq 0$, let t_i denote the position in the trace, resulting after i accesses after the *previous* access $\langle a, m-1\rangle$ to a. For instance, in a trace $T = b\ a\ b\ b\ c\ d\ b\ a\ e\ \ldots$, if $\langle a, 1\rangle$ (i.e., the second access to a) is the considered access, then t_0 is the position after the first access to a, the position t_3 is after the access to c, and t_7 is the position after the access to e. The Markov chain now consists of

- a set Σ of *states*, which must contain the two states *hit* and *miss*.
- an *initial probability distribution*, denoted $P^{(0)}$, which for each state $s \in \Sigma$ defines the probability $P^{(0)}(s)$ of being in state s at t_0,
- *transition probabilities*, denoted $p^{(i)}_{s,s'}$, which for each $i = 0, 1, \ldots$ and each pair of states $s, s' \in \Sigma$, defines the conditional probability of being in s' at t_{i+1}, given that the Markov chain is in s at t_i.

Note that the transition probabilities $p^{(i)}_{s,s'}$ depend not only on the states s and s', but also on the index i of the position t_i. As a typical case, if the state s represents a situation where the reusing access $\langle a, m\rangle$ has not yet occurred and a is still present in the cache, then $p^{(i)}_{s,\mathit{hit}}$ can be taken as the probability that the reusing access occurs, which is $\mathbf{mrdd}(i)$ as explained in Section II.

Given a Markov chain that models the evolution of the cache as above, we can now for $i = 0, 1, 2, \ldots$ calculate the probability distribution, denoted $P^{(i)}$, over its states at position t_i. For $i = 0$, this distribution is given by the initial probability distribution $P^{(0)}$. For $i = 1, 2, \ldots$, we calculate the probability $P^{(i)}(s)$ of being in s at t_i using the formula

$$P^{(i)}(s) = \sum_{s' \in \Sigma} P^{(i-1)}(s') \cdot p^{(i-1)}_{s',s}$$

i.e., $P^{(i)}(s)$ is obtained by summing the probabilities of being in some state s' at position t_{i-1} and then moving to s at t_i, over all states s' of the Markov chain. If the Markov chain is properly designed, then $P^{(i)}(s)$ will approach 0 as i increases, except for the cases $s = \mathit{hit}$ and $s = \mathit{miss}$. We can therefore get our estimate of the miss ratio as $\lim_{i\to\infty} P^{(i)}(\mathit{miss})$, which

we sometimes denote $P^{(\infty)}(\mathit{miss})$. In practice, we will truncate the calculation when i is so large that $P^{(i)}(\mathit{miss}) + P^{(i)}(\mathit{hit})$ is close to 1.

For some of the replacement policies, our Markov chain model will have transition probabilities that depend on the sought miss ratio. We then introduce an unknown x to represent the sought miss ratio, and let some transition probabilities $p^{(i)}_{s,s'}$ depend on x, so that $P^{(\infty)}(\mathit{miss})$ in general depends on x. The sought miss ratio will then be defined by an implicit equation of form $x = P^{(\infty)}(\mathit{miss})$, which we can solve by standard methods (e.g., fixpoint iteration).

For an A-way set-associative cache, we can estimate miss ratios using the same model, but making it local to an arbitrary set. This is done by replacing the cache size C by the size of a set A, and using the set-rdd instead of the rdd.

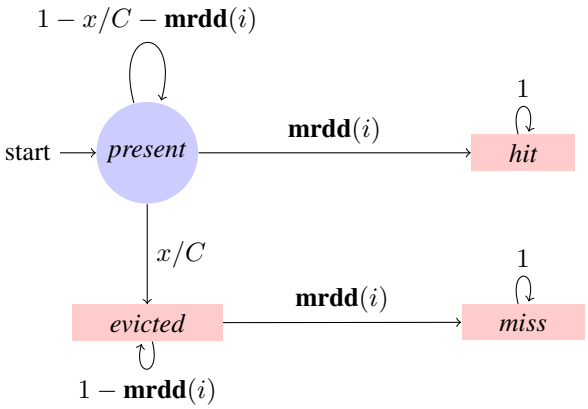

Fig. 1: Markov chain model for random replacement policy

Example (Random replacement policy) Let us illustrate how to construct the Markov chain for the random replacement policy for a fully associative cache of size C.

- Σ contains 4 states: *present*, *evicted*, *hit*, and *miss*, where *present* represents that a has not yet been evicted.
- Initial probabilities are $P^{(0)}_{\mathit{present}} = 1$ and $P^{(0)}_{s} = 0$ for all other states s, since a is present in the cache at the beginning of the reuse interval.
- For $i = 0, 1, \ldots$, the transition probabilities are shown in Figure 1. Both *hit* and *miss* are sink states, which cannot be exited. They are entered whenever $\langle a, m\rangle$ occurs, for which the probability at t_i is $\mathbf{mrdd}(i)$. For the transitions from *present* we have
  - $p^{(i)}_{\mathit{present},\mathit{hit}} = \mathbf{mrdd}(i)$, since this happens when $\langle a, m\rangle$ occurs,
  - $p^{(i)}_{\mathit{present},\mathit{evicted}} = x/C$, since a is evicted precisely if there is a miss (which happens with probability x) which evicts a (which happens with probability $1/C$ by the random replacement policy).
  - $p^{(i)}_{\mathit{present},\mathit{present}} = 1 - x/C - \mathbf{mrdd}(i)$, since this is the only remaining transition from *present*.

Given the above Markov chain model of the cache, let us now calculate the probabilities $P^{(i)}(s)$ for increasing values of i. To simplify the calculations, we here merge the states *evicted* and *miss* into the state *doomed*, since obviously we have $\lim_{i\to\infty} P^{(i)}(\mathit{miss}) = \lim_{i\to\infty} P^{(i)}(\mathit{doomed})$. From $p^{(i)}_{\mathit{present},\mathit{present}} = 1 - x/C - \mathbf{mrdd}(i)$ we derive

$$P^{(i)}(\mathit{present}) = \prod_{j=0}^{i}(1 - x/C - \mathbf{mrdd}(j))$$

which implies

$$P^{(\infty)}(\mathit{doomed}) = \sum_{i=0}^{\infty}\left[x/C \cdot \prod_{j=0}^{i-1}(1 - x/C - \mathbf{mrdd}(j))\right].$$

In total, this means that we can obtain the miss ratio x as the solution to the equation

$$x = \sum_{i=0}^{\infty}\left[x/C \cdot \prod_{j=0}^{i-1}(1 - x/C - \mathbf{mrdd}(j))\right]$$

which can be solved by standard methods (e.g., fixedpoint iteration).

The original StatCache approach [3] also derives an implicit equation for the miss ratio, which is slightly different from ours.

IV. Taking Associativity into Account

In set-associative caches, the estimation of miss probability for a randomly chosen access $\langle a, m\rangle$ should consider only those accesses in the trace that map to the same set as a. This means that miss ratios can be predicted by applying the appropriate replacement policy model from Section V to the set-rdd (instead of the rdd). It is not clear how the low-overhead technique for collecting rdds described in [4] can also obtain set-rdds without additional overhead. For such scenarios, we must therefore first estimate the set-rdd, using the rdd, before applying the model for the actual replacement policy.

For the analogous problem of estimating set stack distance distributions from stack distance distributions, Hill and Smith proposed a simple model based on the assumption that each cache line is mapped randomly to a set, and that the mappings of two different cache lines are independent [13]. They observed that this model usually overestimates the set stack distance distribution (and hence also the miss ratio), since actual set mappings are designed to cause fewer collisions than the model. In our preliminary investigations, we have found that the probability of collisions depend significantly on the used set mapping. We will therefore propose a model which improves in accuracy over that of [13] by taking characteristics of the mapping function into account.

Assume an arbitrary reuse interval $RI\langle a, m\rangle$, and let k be its reuse distance. In order to estimate the set-rdd, we need to estimate the number of accesses in $RI\langle a, m\rangle$ that are mapped to the same set as a. For this estimation, we need the probability that the mapping function f maps an arbitrary access in $RI\langle a, m\rangle$ to the same set as a. Denote this probability by $\alpha^{A,S}_{f}$ to emphasize that it depends also on A and S. Under

the independence assumption, $\alpha_f^{A,S}$ will be $\frac{1}{S}$ [13], but for most actually used mappings, $\alpha_f^{A,S}$ is lower than $\frac{1}{S}$, and also depends on f. We therefore propose to profile the mapping function f on a set of addresses to find out an approximate value for $\alpha_f^{A,S}$. We profile a sequence of accesses to a set of consecutive addresses (not target benchmarks) and apply f to each of the address. During the application of f, we record the number of addresses mapped to each of the S sets.

Let the sets be numbered $1, \ldots, S$, and let $m(i)$ be the number of addresses that are mapped to set i. Letting M be the total number of addresses, $\alpha_f^{A,S}$ can be estimated as

$$\alpha_f^{A,S} = \frac{1}{M} \sum_{i=1}^{S} m(i) \frac{m(i)-1}{M-1} \quad .$$

Having obtained $\alpha_f^{A,S}$, we can estimate the set-rdd as follows. The probability that a reuse interval of size k has i accesses to the same set as a is by standard probability theory $\binom{k}{i}(\alpha_f^{A,S})^i(1-\alpha_f^{A,S})^{k-i}$, hence the set-rdd can be obtained by

$$\mathbf{rdd}^s(i) = \sum_{k=0}^{\infty} \left[\binom{k}{i} (\alpha_f^{A,S})^i (1-\alpha_f^{A,S})^{k-i} \mathbf{rdd}(k) \right] \quad ,$$

where the sum is truncated at a suitable point.

V. Modeling Different Replacement Polices using the General Framework

In this section, we instantiate the generic model framework introduction in section III for three different replacement policies: LRU, Tree-PLRU and Bit-PLRU. For each policy, we first briefly describe how it works, and thereafter present a Markov chain tailored to the policy.

A. LRU On a cache miss, the LRU replacement policy replaces the least recently used (oldest) cache line in the cache. In order to know which cache line is the oldest, the cache lines in the cache are ordered according to the recency of latest access. For an A-way cache, a property of LRU is that an access $\langle a, m\rangle$ is a miss iff the reuse interval $RI\langle a, m\rangle$ contains accesses to at least A different cache lines (i.e., there are accesses to at least A different cache lines between the previous access to a and $\langle a, m\rangle$). For example, in a 4-way cache, given the access sequence $a\ b\ c\ d\ b\ c\ d\ e\ a$, after the first access to a, four different cache lines are accessed ($b\ c\ d\ e$), meaning a has been evicted after these accesses. Reusing a at the end results in a cache miss.

Modeling LRU To study the cache state in a LRU cache, we assign an *age* associated with each cache line. The age of a cache line a is the number of cache lines (in the same set) that have been accessed more recently than a. Thus, right after a is accessed (i.e., at t_0), its age is set to 0. The age of a is increased by an access to a cache line (say b) whose age is bigger than the age of a before this acess to b. For example, in Figure 2, accessing an older cache line d increases the age of younger cache lines a, b and c.

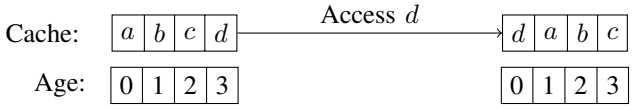

Fig. 2: LRU: increase the ages of cache lines

Our Markov chain for LRU will represent the age of the considered cache line a. It can be obtained from the model for the random replacement policy in Figure 1 by refining the state *present* into one state for each age from 0 to $A-1$ (a cache line is evicted when its age reaches A). The other states (*hit*, *miss*, and *evicted*) are the same as those in Figure 1. The resulting set of states is shown in Figure 3.

Let us motivate the transition probabilities for the transitions from one of the "age" states, say k at position t_i. The age of a is increased from k to $k+1$ when an older cache line is accessed, say by an access $\langle b, n\rangle$. To make b older than a at t_i, the access $\langle b, n-1\rangle$ must be before the last access to a, which is $\langle a, m-1\rangle$. Since there are i accesses between $\langle a, m-1\rangle$ and $\langle b, n\rangle$ at t_i, the reuse distance of b should be bigger than i, i.e., $rd(\langle b, n\rangle) > i$.

Assuming that the reuse distance distribution is independent of the position in the reuse interval, the probability that $rd(\langle b, n\rangle) > i$ can be taken as $\mathbf{rdd}(> i)$. Thus, the transition which increases the age has probability $\mathbf{rdd}(> i)$. The transition that goes to *hit* has probability $\mathbf{mrdd}(i)$, precisely the same as for the random replacement policy. Thus, the transition that keeps the age unchanged and does not encounter the reusing access $\langle a, m\rangle$ has the remaining probability $1 - \mathbf{rdd}(>i) - \mathbf{mrdd}(i)$.

The initial probability is 1 for the state 0, and 0 for the other states.

Based on the Markov chain in Figure 3, we can apply the method introduced in Section III. Since the transition probabilities do not depend on the miss ratio x, we can obtain a closed-form solution. Just as for the random replacement policy, we merge the states *evicted* and *miss* into the state *doomed*. We can calculate the miss ratio from the recurrences

$$\begin{aligned} P^{(i+1)}(k) &= \begin{cases} P^{(i)}(k-1) \cdot \mathbf{rdd}(>i) \\ + \quad P^{(i)}(k)(1 - \mathbf{rdd}(>i) - \mathbf{mrdd}(i)) \end{cases} \\ P^{(i+1)}(\textit{doomed}) &= P^{(i)}(\textit{doomed}) + P^{(i)}(A-1) \cdot \mathbf{rdd}(>i) \end{aligned}$$

and obtain the miss ratio as $\lim_{i\to\infty} P^{(i)}(\textit{doomed})$.

B. PLRU While exploiting the program's temporal locality, LRU is expensive to implement due to the need to keep accesses' recency. In practice, a replacement policy approximating LRU is typically used, e.g., PLRU. The policy PLRU (sometimes called Tree-PLRU) organizes the cache lines in a set into a binary tree with depth $log_2 A$. At the leaves are the cache lines. Each internal node has a bit which indicates which of its subtrees was least recently accessed. By convention, the value 0 denotes that the left subtree was least recently accessed, and 1 that the right subtree was least recently accessed. These bits are used to determine which cache line to evict on a cache miss by traversing the tree from the root and always choosing the least recently accessed subtree. For example, in Figure 4, traversing the tree from the root reaches the cache line d.

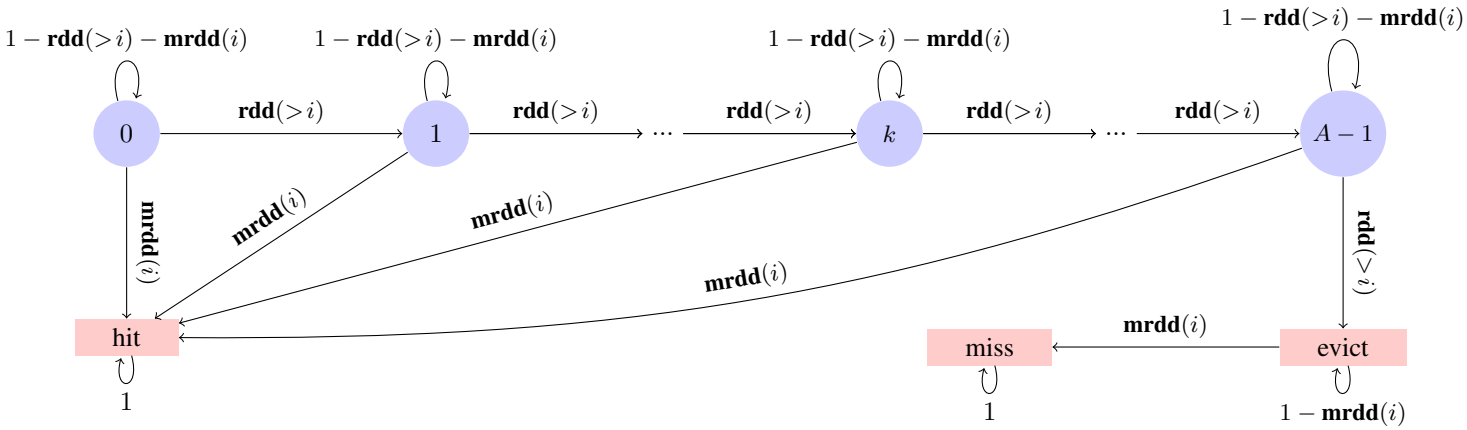

Fig. 3: Modeling LRU

Replacement in PLRU works as follows. Whenever a cache line is accessed, the bits on the path to this cache line are set to be pointing away from this cache line. For example, after c is accessed in Figure 4, the pointers on the path to c (root and left-subtree pointer) are both set to 1 to point away from c. On a cache miss, we follow the directions of the bits and replace the cache line they point to; thereafter we set the pointers on the path to the replaced location to point away from it. In Figure 4, on a cache miss, d will be replaced, the top pointer will be set to 1 and the left to 0. The intuition of the tree-PLRU replacement policy is that when a cache line is being pointed to, it is approximately the oldest cache line.

Modeling PLRU In analogy with the model for LRU, we decompose the *present* state into several states that are relevant for the eventual eviction of a. For PLRU, it is natural to choose the sequence of bits in internal nodes on the path from the root to a, since their values will determine whether or not a is evicted on a miss.

By symmetry, we can without loss of generality assume that cache line a is installed at the rightmost leaf in the tree. Thus, a will be evicted on a miss iff the bits on the rightmost path in the tree at all set to 1. At t_0 (the beginning of the reuse interval), these bits are all set to 0. Thus, our model uses $2^{\log_2 A} = A$ states in which a is still in the cache.

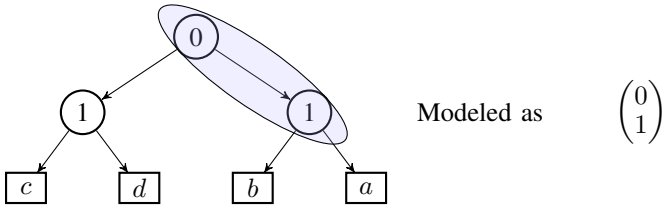

Fig. 4: Abstract the cache state of a 4-way Tree-PLRU

Let us define our model, illustrated on a 4-way PLRU cache (Figure 4). There are four cache lines a, b, c and d in this cache. a is the reused cache line, b is the cache line sharing a subtree with a. c and d are the two cache lines in the other subtree. For a 4-way cache, there are, in addition to the states *hit*, *miss*, and *evicted*, four different states, corresponding to the four possible values of the bits on the rightmost path: $\begin{pmatrix}0\\0\end{pmatrix}$, $\begin{pmatrix}0\\1\end{pmatrix}$, $\begin{pmatrix}1\\0\end{pmatrix}$ and $\begin{pmatrix}1\\1\end{pmatrix}$.

Let us define the transitions. From each of the four states of form $\begin{pmatrix}*\\*\end{pmatrix}$, we consider the following transitions: (1) a miss at t_i, (2) a reuse of a at t_i, (3) $p^{hit}_{b,i}$: meaning that cache line b is accessed at t_i, and (4) $p^{hit}_{c\vee d,i}$: meaning that cache line c or d is accessed at t_i. We make the uniformity assumption that it is equally likely to hit on position b, c and d, i.e., each having $\frac{1}{3}$ the probability that we have neither a miss nor a reuse of a. Under this assumption, the probabilities for the respective cases will become

- a miss has probability x,
- a reuse of a has probability $\mathbf{mrdd}(i)$,
- $p^{hit}_{b,i}$ has probability $\frac{1}{3}(1 - x - \mathbf{mrdd}(i))$, and
- $p^{hit}_{c\vee d,i}$ has probabilitity $\frac{2}{3}(1 - x - \mathbf{mrdd}(i))$.

With these probabilities, the resulting Markov Chain is shown in Figure 5. Let us motivate some of the transitions in Figure 4.

- Consider the transition from $\begin{pmatrix}0\\0\end{pmatrix}$ to $\begin{pmatrix}1\\0\end{pmatrix}$, in which the bit at the root is changed from 0 to 1. This can happen either by accessing the left subtree (with probability $p^{hit}_{c\vee d,i}$) or by a miss (probability x): note that a miss will replace a cache line in the left subtree.
- Consider the transition from $\begin{pmatrix}0\\1\end{pmatrix}$ to itself: the state $\begin{pmatrix}0\\1\end{pmatrix}$ represents the root pointer pointing to the c and d and the second pointer pointing to a as shown in Figure 4. The only way to keep the pointers unchanged is to hit on b, which makes the probability of this self loop p^{hit}_b.

Since the miss ratio x appears in the transitions, calculating the miss rate must be done by solving an implicit equation (we use fixed-point iteration).

C. bit-PLRU Bit-PLRU (sometimes called MRU) is one of the most widely used cache replacement policies, and is employed in mainstream processor architectures like Intel Nehalem (the architecture codename of processors like Intel Xeon, Core i5 and i7) [9].

For each cache set, the bit-PLRU policy maintains an ordering between the available slots. For each cache line that is present in the cache, an extra bit (here called an *MRU-bit* is stored, which approximately represents whether this cache line

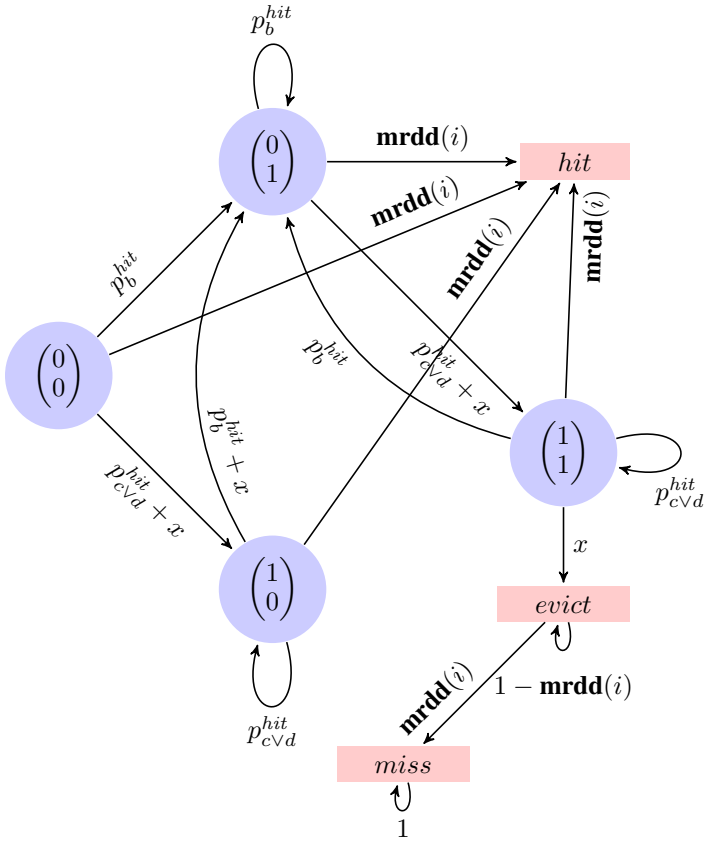

Fig. 5: States and transitions of PLRU cache model

was recently visited. On a cache hit, the bit associated with the accessed cache line is set to 1. On a cache miss, the first cache line (in the set) with a 0 bit is replaced and the bit is set to 1. Eventually there will be only one MRU-bit being 0 in the cache set. When the last 0 bit is set to 1, all other bits are simultaneously flipped to 0; this is called a *global flip*. Thus the global flip guarantees that in each set, there is always at least one MRU-bit which is 0.

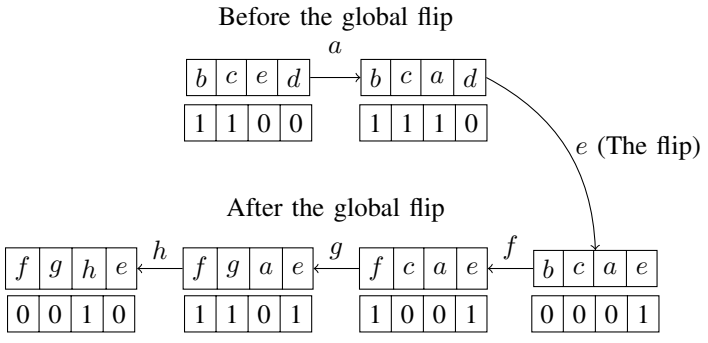

Fig. 6: Stages of the states in bit-PLRU

As an example, in Figure 6 the access to a causes the first cache line with a 0 bit (e in this case) to be replaced by a, and the corresponding bit to be set. The next access (to e) causes the global flip, whereafter all bits, except for that of e, are 0.

Modeling bit-PLRU. Let us make a model of how the cache line a and its set evolves during a reuse interval. We note that at the beginning of the reuse interval, a's bit is set to 1. Later, the global flip sets a's bit to 0, and when a thereafter becomes the first cache line with a 0 bit, it is eligible for eviction at the next cache miss (see Figure 6). Thus, to model the "distance" to eviction, we divide the reuse interval into two stages: before the global flip and after the global flip. To each cache position, we assign an index from 0 to $A-1$, from left to right. For cache lines a and b, we say a *is before* b if the index of a's position is smaller than that of b's.

The states of our model are triples of form $(nz, j, \textit{flipped})$, where nz is the number of zero-bits, *flipped* is a bit which is 1 iff the global flip has occurred, and j is the index of a's position if *flipped* $= 0$, otherwise (i.e., if *flipped* $= 1$) j is number of 0's before a. We motivate the choice of j as follows. Before the global flip, the position of a influences the value of j after the global flip. After the global flip, the number of zero-bits before a measures of how close a is to eviction.

Let us consider the transitions between states. As in the PLRU model, we let x denote the miss ratio, $\mathbf{mrdd}(i)$ the marginal probability of reuse. Then at t_i, the probability of a hit is $p^{hit} = 1 - x - \mathbf{mrdd}(i)$. We define the following probabilities, which depend on the number nz of 0-bits.

- The probability of a hit on a 0-bit cache line is defined as $p_0^{hit}(nz) = p^{hit} \cdot \frac{nz}{A-1}$ before the global flip (assuming equal probability of hit on each bit) and as $p_{0,flipped}^{hit}(nz) = p^{hit} \cdot \frac{nz-1}{A-1}$ after the global flip.
- The probability of a hit on a 1-bit cache line is defined as $p_1^{hit}(nz) = p^{hit} \cdot \frac{A-1-nz}{A-1}$ before the global flip, and as $p_{0,flipped}^{hit}(nz) = p^{hit} \cdot \frac{A-nz}{A-1}$ after the global flip.

We have the following classes of transitions from a state $(nz, j, \textit{flipped})$.

Transitions before the global flip (i.e., flipped $= 0$*):*

- a miss or hit on 0-bit goes to state $(nz-1, j, \textit{flipped})$ with probability $x + p_0^{hit}(nz)$ if $nz > 1$, otherwise (if $nz = 1$) to state $(A-1, j-1, 1)$ if a cache line before j triggers the global flip with probability $(x + p_0^{hit}(1))\frac{j}{A-1}$, and to $(A-1, j, 1)$ if cache line after j triggering the global flip with probability $(x + p_0^{hit}(1))\frac{A-j-1}{A-1}$.
- a hit on a 1-bit goes back to $(nz, j, \textit{flipped})$ with probability $p_1^{hit}(nz)$.

Transitions after the global flip (i.e., flipped $= 1$*)*

1) to state *evicted* on a miss if $j = 0$ with probability x,
2) to state $(nz-1, j-1, 1)$ on a miss or a hit on a 0 cache line before a, where $nz > 1$ and $j > 0$, with probability $\frac{j}{nz-1} p_{0,flipped}^{hit}(nz) + x$,
3) to state $(nz-1, j, 1)$ on a hit on a 0 bit after a, where $nz > 1$ and $nz - j - 1 > 0$, with probability $\frac{nz-j-1}{nz-1} p_{0,flipped}^{hit}(nz)$,
4) to state $(nz, j, 1)$ on a hit on a 1-bit with probability $p_{1,flipped}^{hit}(nz)$.

For the initial distribution, since the nz and j at t_0 is not known, we assume a uniform initial distribution over the values of nz and j. We then use the method introduced in section III to estimate the miss ratio.

VI. Evaluation

In this section, we evaluate the accuracy of our framework for a range of different cache configurations. When using our framework for low-overhead prediction of miss ratios in set-associative caches from rdds obtained by sampling, there are three sources of inaccuracy: the inaccuracy introduced (A) by the policy model (Section V), (B) by the estimation of set-rdds from rdds (Section IV), and (C) by sampling. We evaluate the inaccuracy introduced by each of these sources by comparing actual miss ratios with miss ratios obtained

(1) from actual set-rdds (evaluating (A)),
(2) from set-rdds estimated using
  (i) the Hill/Smith model, or
  (ii) our adapted model, based on either
    (a) actual rdds (evaluating (A)+(B)), or
    (b) rdds obtained by sampling (for (A)+(B)+(C)).

Scenario (1) corresponds to a scenario when the extra overhead of obtaining the actual set-rdds (an order of magnitude overhead) can be tolerated. Scenario (2) is relevant when such overhead can not be tolerated: then set-rdds can be estimated from rdds using either the Hill/Smith model or our adapted model (see Section IV), since rdds can be obtained with much less overhead, in particular if sampling is employed [4] (corresponding to Scenario (2iib).

Experiment setup We evaluated our models with the SPEC2006 benchmark suite [12][1]. Our models are evaluated with cache sizes 4kB, 8kB, 16kB, 32kB and 64kB, with associativities from 2 to 16, and with LRU, PLRU, and bit-PLRU replacement policies. We choose to use small caches because the SPEC2006 benchmarks have too small miss ratios in large caches (19 out of 30 benchmarks would then have a miss ratio smaller than 5% and the average miss ratio is 6.3% in a 64kB cache) to make a thorough evaluation of our models. The profiling and modeling are done on a 32-core system SandyBridge architecute with 2.7GHz Intel Xeon E5-4650 CPUs.

We obtain actual cache miss ratios using Intel's PIN [16] tool to trace and record a sequence of ten million memory accesses for each benchmark. The sequence of memory accesses is then used to simulate the behavior of caches with different sizes, replacement policies, and associativities. This sequence is also used to obtain the actual set-rdds for different cache sizes and associativities. For mapping cache lines to cache sets, we use the Xor mapping from [6], also used in [18].

The actual rdds are collected on the fly while tracing the target application. The sampled rdds are obtained by sampling the reuse distances of randomly selected accesses. We make an unbiased sampling from all accesses by using a sample interval which is geometrically distributed with a certain parameter, corresponding to the density of sampling. Berg and Hagersten [4] have shown that the sampled rdds can be obtained with small overhead. Around 10000 samples for each benchmark are collected.

Results Figures 7, 8 and 9 show the average absolute error of estimated miss ratios over all the benchmarks (for lack of space, we can only present a selection of cache configurations). Using the indexing scheme of the first paragraph of this section, they are estimated (1) from actual set-rdds, (2i) from sett-rdds estimated with Hill/Smith from actual rdds (2iia) from set-rdds estimated with our set-rdd model from actual rdds, and (2iib) from set-rdds estimated with our set-rdd model from sampled rdds. For the LRU replacement policy, Figure 7 also shows average absolute error of miss ratios estimated by the Statstack model [8] from actual set-rdds.

For LRU, our policy model has an average error of 0.72% over all the configurations (not only the ones shown in the figure) when applied on actual set-rdds. The estimated error increases to 3.2% when the set-rdds is estimated with the Hill/Smith model, but only to 2.8% when using our model for estimating set-rdds (i.e., our set-rdd model outperforms the Hill/Smith model by 0.4%). Using Sampling to obtain rdds adds an extra 0.2% to the absolute error. When comparing our LRU model (Section V A.) to the StatStack model [8], we find that StatStack's average absolute error of 0.92% is higher than ours by a difference of 0.2%.

For PLRU (Figure 8) and bit-PLRU (Figure 9), our models give 0.93% and 0.98% of absolute error, respectively, when applied on actual set-rdds. Estimating set-rdds from actual rdds increases the error to 2.9% for both the Hill/Smith model and our model cases. Sampling the rdds adds another percentage of error.

To indicate how the estimation varies over different benchmarks, we show data for a particular configuration, in this case the 8-way 4kB bit-PLRU cache, in Figure 10. The average absolute error of the policy model from actual set-rdds for all 30 benchmarks is 2.4%, with the biggest error 7.38% (429.mcf) and the relative standard deviation 0.22. The estimates obtained using our set-rdd model always have a lower error (2.8% on average) than the Smith model (3.7% error on average). The error increases from 2.8% to 2.96% when the rdds are sampled.

VII. Related Work

The most common metric to characterize a program's locality is that of stack distance [5] [19] [7]. The stack distance is obtained from memory traces, which are typically collected by an order of magnitude slow-down. Algorithms for obtaining stack distances from memory traces in time complexity $\mathcal{O}(n \log m)$ (n being the length of the memory trace and m the number of distinct memory addresses) have been presented by Almási et al [1], Bennet et al [2] and Zhong et al [21]. The reuse distance can be computed in almost linear time.

Several techniques have been presented for estimating miss ratios from stack distances. Guo and Solihin [11] predict cache miss ratios for varying cache replacement policies, cache sizes and associativities, based on a combination of stack distance and reuse distance, which is more expensive to obtain than only the stack distance. Sen and Wood [18] develop an online modeling framework to predict the cache miss ratio in set-associative caches with different replacement policies from stack distance distributions. They also show how stack distance

[1] All benchmarks are included except for 447.dealII, which did not compile on our system

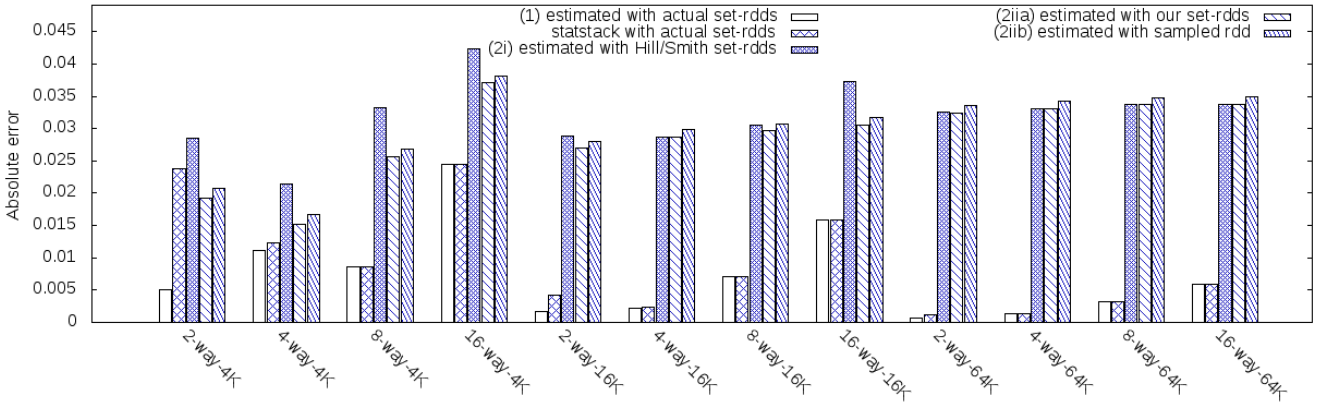

Fig. 7: Comparison of LRU error rates of five estimated cache miss ratios

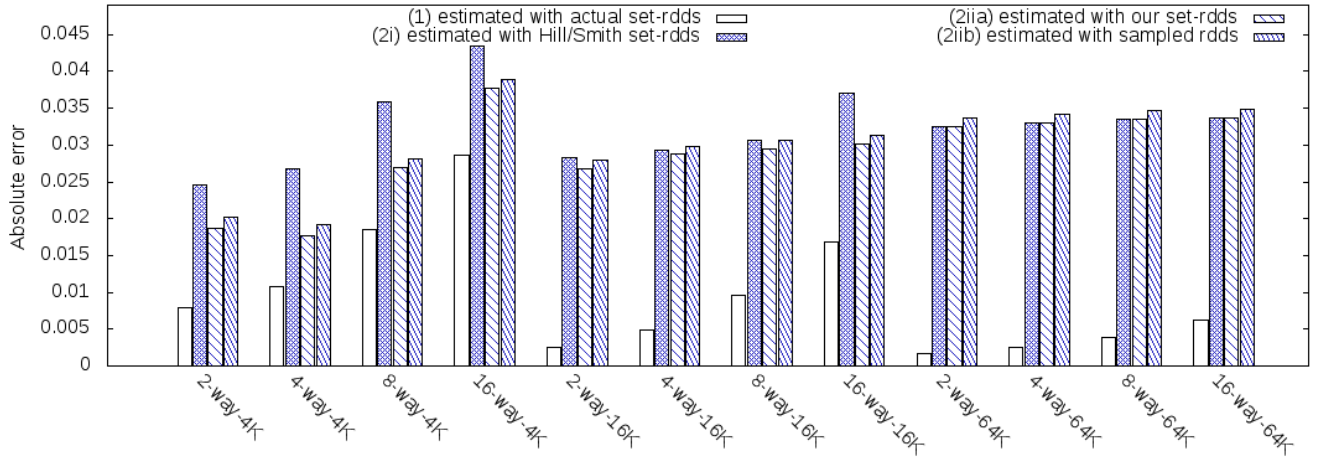

Fig. 8: Comparison of PLRU error rates of four estimated cache miss ratios

distributions can be obtained by special on-chip hardware, which however is not available in today's processors.

Not so many techniques have been presented for estimating miss ratios from data that can be collected with small overhead. Berg and Hagersten show how reuse distance distributions can be collected with small overhead [4] and be used to estimate miss ratios for fully associative random caches [3], in analogy with the technique we present at the end of Section III. Later, Eklov and Hagersten showed how reuse distance distributions could be used to estimate miss ratios in fully associative LRU caches [8]. In the evaluation in sectionVI, we compared the accuracy of our LRU model with that of StatStack, based on actual set-rdds. Our model (error 0.72%) is on average more accurate than StatStack (error 0.92%). Tam et al. obtain miss ratio curves for L2 caches on PowerPC processors by online probing, using features of the PowerPC, which are not available on other processors [20].

For the problem of estimating set stack distance distributions from stack distance distributions, Hill and Smith [13] proposed a probablistic model, which was later used by Sen and Wood [18]. In section VI, we show that our model is on average 0.4% more accurate than the Hill/Smith model.

VIII. Conclusion

We have presented a general modeling framework for estimating cache miss ratios for running applications, using only their reuse distance distributions. The framework proposes a new generic format for cache performance predictions, and can be instantiated for caches with different sizes, associativities and cache replacement policies: we showed how the framework can be instantiated for Random, LRU, PLRU, and bit-PLRU replacement policies, and also for taking set-associativity into account by first estimating set-rdds, for which we proposed an improvement of the previously used Hill/Smith model. The introduced inaccuracies were generally below 1% for the model of the policy, and additionally around 2% for estimating the set-rdd from rdd. The inaccuracy introduced by sampling is significantly smaller.

A way to increase the precision of predictions would be to identify phases in an application and make phase-specific estimations.

Acknowledgment This work is supported by the Swedish Foundation for Strategic Research through CoDeR-MP, and by the Swedish Research Council through UPMARC. We are very

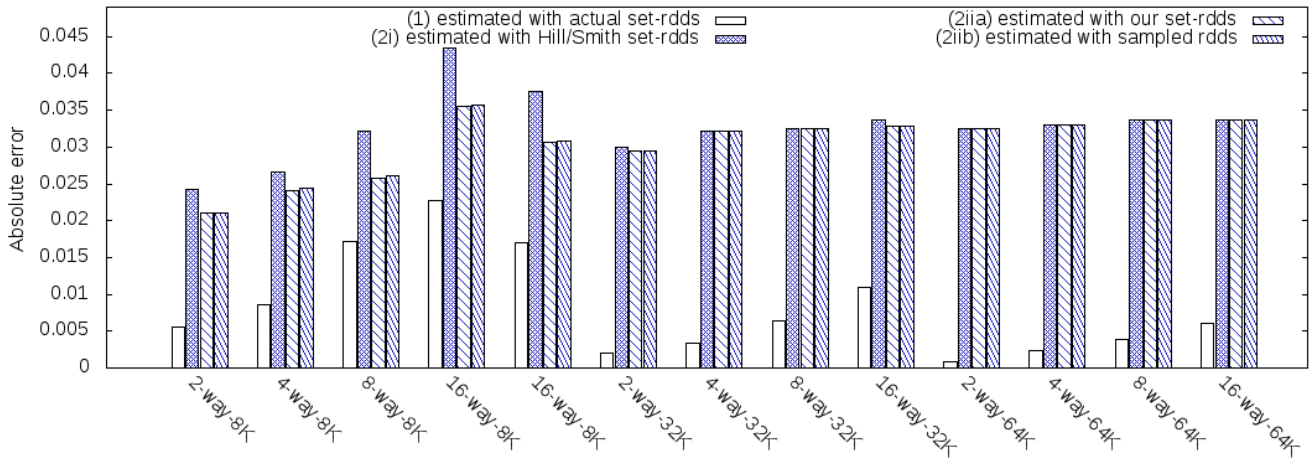

Fig. 9: Comparison of bit-PLRU error rates of four estimated cache miss ratios

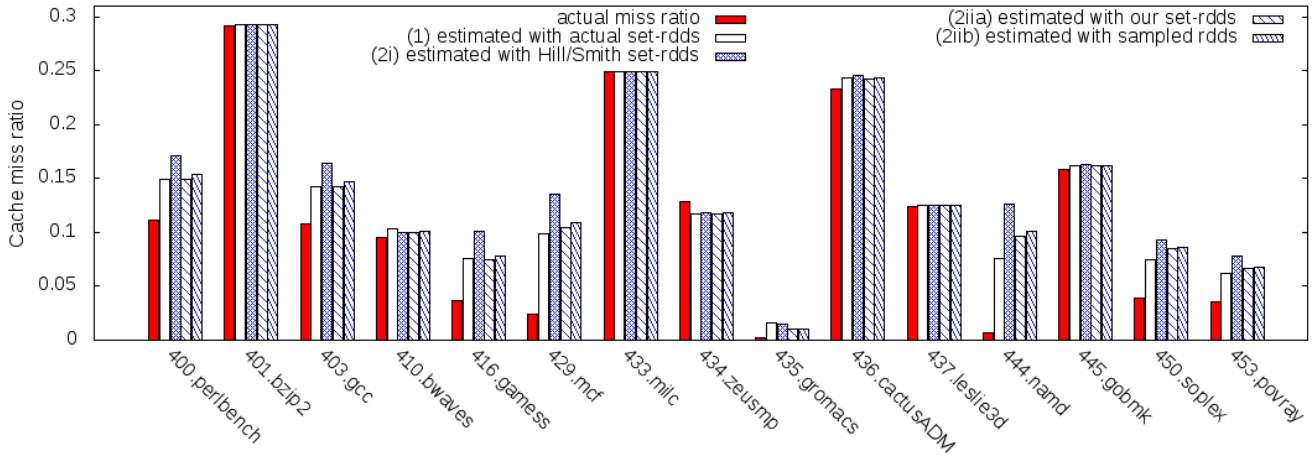

Fig. 10: Actual miss ratio v.s. estimated miss ratios for a 8-way 4K bit-PLRU cache (15 benchmarks)

grateful to Nikos Nikoleris, Erik Hagersten and David Black-Schaffer for the valuable discussions. We would like to thank the anonymous reviewers for their helpful suggestions.

REFERENCES

[1] G. Almási, C. Caşcaval, and D.A. Padua. Calculating stack distances efficiently. In *Workshop on Memory System Performance 2002*. ACM.

[2] B. T. Bennett and V.J. Kruskal. Lru stack processing. *IBM Journal of Research and Development*, 19(4):353–357, July 1975.

[3] E. Berg and E. Hagersten. StatCache: a probabilistic approach to efficient and accurate data locality analysis. In *ISPASS*, 2004.

[4] Erik Berg and Erik Hagersten. Fast data-locality profiling of native execution. In *SIGMETRICS*, pages 169–180. ACM, 2005.

[5] K. Beyls and E.H. DHollander. Reuse distance as a metric for cache behavior. In *IASTED PDCS*, pages 617–662, 2001.

[6] R. Cypher. Mechanism and method for determining stack distance of running software, May 18 2006. US Patent App. 11/281,733.

[7] C. Ding and Y. Zhong. Reuse distance analysis. Technical report, 2001.

[8] D. Eklov and E. Hagersten. StatStack: Efficient modeling of LRU caches. In *ISPASS*, pages 55–65, 2010.

[9] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten. Cache pirating: Measuring the curse of the shared cache. In *ICPP*, 2011.

[10] David Eklov, David Black-Schaffer, and Erik Hagersten. Fast modeling of shared caches in multicore systems. HiPEAC '11. ACM.

[11] F. Guo and Y. Solihin. An analytical model for cache replacement policy performance. In *SIGMETRICS '06/Performance '06*. ACM, 2006.

[12] J.L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[13] M.D. Hill and A.J. Smith. Evaluating associativity in CPU caches. *IEEE Trans. Computers*, 38(12):1612–1630, 1989.

[14] R.E. Kessler, M.D. Hill, and D.A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Trans. Computers*, 43(6):664–675, 1994.

[15] S. Laha, J.H. Patel, and R.K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Trans. Computers*, 37(11):1325–1336, 1988.

[16] C.-K. Luk and R.S. Cohn et al. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI*, 2005.

[17] N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. In *PLDI*, pages 89–100. ACM, 2007.

[18] R. Sen and D.A. Wood. Reuse-based online models for caches. In *SIGMETRICS*, pages 279–292, 2013.

[19] X. Shi, F. Su, J.-K. Peir, Y. Xia, and Z. Yang. Modeling and stack simulation of CMP cache capacity and accessibility. 2009.

[20] D.K. Tam, R. Azimi, L. Soares, and M. Stumm. RapidMRC: approximating L2 miss rate curves on commodity systems for online optimizations. In *ASPLOS*, pages 121–132. ACM, 2009.

[21] Y. Zhong, X. Shen, and C. Ding. Program locality analysis using reuse distance. *ACM Trans. Program. Lang. Syst.*, 31(6):20:1–20:39, 2009.