Analytical Derivation of Concurrent Reuse Distance Profile for Multi-Threaded Application Running on Chip Multi-Processor

Jasmine Madonna Sabarimuthu and T.G. Venkatesh

Abstract—Reuse distance has been shown to be a useful metric for performance analysis of caches and programs, locality analysis and compiler optimization. Concurrent reuse distance profile defined as the reuse distance profile of a thread sharing the cache with many other threads varies from its standalone reuse distance profile due to interference from other threads. Measurement of reuse distance profile through simulation, especially for multi-threaded applications, consumes lot of time. Analytical model based reuse distance prediction can reduce drastically the time taken for exploring the cache memory design space. The objective of this work is to propose an analytical model to find the concurrent reuse distance profile of a thread belonging to multi-threaded applications in a shared memory environment. Using the standalone reuse distance profile of each thread as input, we derive three other reuse distance profiles: 1) The concurrent reuse distance profile of a thread sharing the cache with other threads 2) The combined reuse distance profile of all threads sharing the cache and 3) The coherent reuse distance profile of each thread, considering the coherency effect when each thread runs with private cache. We use Markov chain besides combinatorics and basic probability theory as a main analytical tool for the model. We validate our analytical model against simulations, using the multi-core simulator Sniper for the benchmarks of the PARSEC and the SPLASH benchmark suites.

Index Terms—Reuse distance profile, concurrent reuse distance, cache performance, analytical model, multi-threaded applications, multi-core processors, simulation

1 Introduction

Performance evaluation plays a vital role in the design cycle of the next generation processors as it allows the architect to tune the performance of processor using optimal system parameters [1]. Micro-architectural simulation, either application only [2] or full system [3], is widely used for evaluating the performance of the processors. Although simulators give accurate results, simulation time is very high. Analytical models come as a better alternative when we need quick results. It is useful in the early stage of design cycle. The most important advantage of analytical modelling is the insight they provide us in understanding the trade off between different performance metrics. Analytical models have much faster computation times. In practice it makes sense to trade-off a long simulation (which takes a time proportional to the number of instructions executed), for a numerical computation based on analytical model that takes constant time computing.

Though there are a few analytical models available in the literature, much of the complexities of current multi-core processors have not been well captured in the existing models [4], [5]. Modeling the characteristics of multi-threaded applications is one such complexity. Here, the complexity arises due to the presence of shared data among the threads. The need for a system level performance metric of a multi-program workloads has been pointed out by Eyerman and Eeckhout [6]. As mentioned by Yi et al. [7], a healthy research on analytical models for processors and caches is still lacking. We take a small step forward by presenting a new analytical method to find the reuse distance profile of multi-threaded applications.

The performance of the processors can be significantly improved using a well-designed cache memory. Today's Chip multi-processors (CMP) have multiple levels of cache to even boost the performance of memory hungry applications [8], [9]. Sometimes caches may be shared using efficient cache partitioning schemes [10]. This adds to the complexity of modeling the cache. Many researchers [4], [11], [12], [13], [14], [15] have worked on analytical cache models, progressively adding more and more new features to the model. The aspect still lacking in the literature is models for multi-threaded applications with data sharing.

Reuse distance (RD) of data accesses to a cache has been proven to be an useful metric for cache performance analysis [16], [17], [18], driving cache management policy decisions [19] and program locality predictions [20], [21]. Reuse distance of reference to a cache block is the number of unique cache accesses between two consecutive accesses to

• *The Authors are with the Department of Electrical Engineering, Indian Institute of Technology, Madras, Chennai 600036, India.*
E-mail: jasminemadonnas@gmail.com, tgvenky@ee.iitm.ac.in.

Manuscript received 13 Apr. 2018; revised 9 Jan. 2019; accepted 19 Jan. 2019. Date of publication 31 Jan. 2019; date of current version 8 July 2019.
(Corresponding author: T.G. Venkatesh.)
Recommended for acceptance by Alba Cristina Magalhaes Melo.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2019.2896633

the same cache block[1] It is synonymous to LRU stack distance for a fully associative cache [22]. Reuse distance profile—a unique signature of a thread/program—gives the probability of occurrence of each reuse distance for a phase or entire execution of the program. The use of reuse distance metric as a locality measure has been well discussed in the literature [17], [18], [22], [23], [24].

In a chip-multiprocessor, the RD analysis and its measurement becomes complex because of multiple threads of different programs sharing the cache. In case of multi-threaded applications the RD measurement will be even more complicated due to sharing of data among the threads. CRD profile gives the RD profile of a thread running with many other threads sharing the cache. Many of the recent works [24], [25], [26], [27] have stressed the importance of CRD profile and the complexity of its measurement in the multi-core environment. Jiang et al. [24] have shown the potential use of the CRD for multi-threaded applications and its independence to input data sets and the absolute execution speed of individual threads.

When the threads of the multi-threaded applications run with private cache for each thread, coherence misses arise. In this case, the reuse distance profile of a thread with a private cache will get affected destructively by the writes to shared data by threads with different private caches. Thus it is challenging to derive the coherent reuse distance profile of each thread with private cache considering the coherency effect.

Motivation for proposing our model is as follows. There is a need to model multi-threaded applications with arbitrary number of threads running with shared memory in a chip multi-processor. In order to fill this gap we propose a stochastic model for the above mentioned case and derive the (i) concurrent RD profiles of threads sharing a cache, (ii) Combined RD profile of all the threads sharing a cache, and (iii) the coherent RD profile of threads running on private caches, and affected by the coherency effect. Using one-time measurement of the standalone reuse distance profile of the threads along with certain related statistical quantities we envisage to compute the above mentioned RD profiles at higher levels of the cache for various cache configurations. As a result the proposed model can drastically reduce the design cycle time as compared to a purely simulation based approach.

The main contributions of our work are

- Probabilistic method to derive the concurrent reuse distance profile of a thread belonging to multi-threaded applications sharing the cache with many other threads.
- Derivation of combined reuse distance profile of the entire application or of all the threads sharing the cache.
- Derivation of coherent reuse distance profile of each thread considering the coherency effect, when each thread runs with a private cache.

1. Caution : In contrast to our definition some literature, define reuse distance as the number of accesses (not necessarily unique), while reserve the term stack distance as the number of unique accesses between two consecutive accesses to the same data. Please refer Section 3 for more generic definition of RD.

The organization of the paper is as follows. A brief literature survey is given in Section 2. Section 3 explains the background and motivation for our work. The inputs and assumptions for our model along with the terminologies we use is given in Section 4. In Section 5, we derive the concurrent reuse distance profile of the thread sharing the cache with other threads using Markov chain and combinatorics. The derivation of combined reuse distance profile of all threads sharing the cache is also explained. In Section 7, we explain the derivation of coherent reuse distance profile of threads, when each thread runs with a private cache. We present the simulation methodology and validation of our analytical model in Section 8. We conclude our work in Section 9 - providing directions for future work.

2 Related Work

Number of research articles have appeared in the literature that have shown the usefulness of reuse distance in cache performance analysis [16], [17], [18], cache management policies [19] and program locality predictions [20], [21]. Inspired by its versatility, many researchers have proposed different methods of finding the RD profile, especially in multi-core environment wherein the measurement of reuse distance gets complicated in the presence of shared cache. However, only very few have made an attempt to compute the concurrent reuse distance analytically [24].

Jiang et al. [24] carried out an analysis on the complexity of extending the reuse distance measurement in CMP environment. They provided a probabilistic model to find the CRD profile from the locality information of individual threads.

Schuff et al. [25] illustrated how the reuse distance definition could be extended in the multi-core environment and the measurement of the same using simulation. They extended the work to accelerate the reuse distance measurement in multi-core environment based on sampling in [26]. Wu and Yeung [27] explained the application of the RD analysis for the performance analysis of loop based parallel programs and the prediction of CRD profile in such programs. Ding and Zhong [23] used reuse distance analysis for program locality measurement. Ding and Zhong [21] did program locality analysis and pattern recognition of program data through an approximate reuse distance analysis.

Footprint measures the amount of active data usage in an access window. A refined footprint theory has been introduced by Ding et al. [28]. The footprint theory is based on the concept of footprint measurement, footprint distribution analysis, average footprint analysis, and footprint sampling. The theory also deals with the properties of footprint composition, optimal co-scheduling.

Zhong et al. have proposed a parameterized model that predicts the cache miss rate for arbitrary data input set sizes given a cache size and associativity [29]. Ding and Chilimbi [30] presented a composable model to predict the locality information of threads of multi-threaded applications.

One of the early works which showed the effectiveness of reuse distance analysis in program modeling and evaluation of storage systems was done by Mattson et al. [22], which they called *stack processing* algorithms. Beyls et al.

[18] showed how stack distance could be used for performance analysis of caches and programs.

Xu et al. [5] have proposed a cache contention model and predict the instruction throughput for processes running on CMPs. One of the recent works with reuse distance as input was a generic reuse based online model for caches for various replacement policies by Sen et al. [16]. A general framework for the fast measurement of CMP cache performance in both shared and private cache environment based on a single stack simulation of reuse distance was done by Shi et al. in [31]. Cascaval et al. [17] proposed a method to estimate the cache misses at compile time based on stack algorithms.

Steen and Eeckhout have shown the utility of the reuse distance metric for predicting memory level parallelism [32]. Duong et al. [19] have proposed cache replacement policy combined with a bypass policy and cache partitioning policy based on dynamic reuse distances. Analytical calculation of the miss rate of L1 cache for different configurations directly from the RD profile of L1 cache has been reported [33]. Maeda et al. introduce a generalization to the reuse distance, called Hierarchical Reuse Distance (HRD) which can capture locality seen at multiple granularities. They then used HRD to analyze caches with hybrid line sizes, such as sectored caches [34]. Ceballos et al. have proposed StatTask a statistical cache model to predict cache behaviour for a given cache size and task schedule [35]. StatTask can compute how reuse distance profile changes with single and multi-threaded executions of tasks. In contrast to models based on task level parallelism, models based on thread-level parallelism have to tackle the issue of coherency.

Shen et al. [36] proposed a statistical model to estimate reuse distance histogram from easily obtained time distance histogram. Xiang et al. [37] gave a set of methods to derive one locality metric from the other. The locality metrics they use are footprint, inter-miss time, volume fill time, miss ratio and reuse distance. Using their theory of locality metric conversion, they predict the cache miss ratio without special hardware support.

One of the earlier works to model cache contention when two threads of multi-programmed workloads share the cache was done by Chandra et al. [14]. Chen et al. [4] extended the cache contention model based on circular sequence profile when multiple threads share the cache.

Berg and Hagersten in their work on StatCache gave a probabilistic method to estimate the cache miss ratios of fully associative cache for random replacement from reuse distance distributions [38], [39]. Eklov and Hagersten [40] extended the StatCache model—called StatStack—to estimate cache performance of fully associative cache with LRU replacement policy. StatStack work is further extended by Eklov et al. [41]. They modeled cache contention to estimate the cache miss ratio and CPI (Clock Cycles per Instruction) of co-scheduled applications. Prediction of the cache miss ratio of the co-scheduled application based on the StatStack [41] has been restricted to only two threads, while our formulation does not put any restriction on the number of the concurrent threads.

Kaxiras et al. have proposed a model for coherence Communication Prediction in Shared-Memory Multiprocessors using statistical techniques from epidemiological screening and polygraph testing [42]. Steen et al., have proposed analytical model that is independent of micro architectural input parameters such as cache miss rate, branch miss rate etc. [43], [35].

Following are the unique features of our model. Our model takes as input the stand alone RD profiles of the threads along with certain measurable statistics. Our probabilistic model is based on few basic and valid statistical assumptions. We compute reuse distance profiles for threads running concurrently, that lead to constructive as well as destructive interference. Our model is valid for arbitrary number of interfering threads. Our model accounts for shared data among the threads and the effect of coherency.

3 Background

In this section, we present the background and a set of concepts related to the reuse distance, reuse distance profile, reuse interval, time distance, and concurrent reuse distance profile.

Reuse Distance (RD). Reuse distance of a data reference is defined as the number of unique accesses between two consecutive accesses to the same data. When a data is referenced for the first time, its reuse distance is taken to be infinite. The data can be cache blocks, individual words, pages or instructions [25]. In this work we use reuse distance of references to cache blocks. Cache block is the basic unit of transfer between the main memory and the cache.

The probability for a reference to a cache block by a thread to have a particular RD is given by,

$$P(rd = x) = \frac{Number\ of\ references\ with\ rd = x}{Total\ number\ of\ references\ made\ by\ the\ thread}.$$

The plot which gives the probabilities for all reuse distances for a phase or entire execution of a program/thread is called reuse distance profile.

Standalone RD Profile. The RD profile of an isolated thread running with a private cache—without any interference from other threads—is called the standalone RD profile. It is also referred to as the private RD profile (See Ref. [27] of Wu and Yeung).

Reuse Interval (RI). The interval consisting of a sequence of references made between two consecutive accesses to the same data is called reuse interval.

Time Distance (TD): Total number of references between two consecutive references to the same data is called the time distance. In other words, it is total number of references in a reuse interval [36].

The RD profile of the thread when multiple threads run together sharing a cache varies from its standalone RD profile. The threads running together sharing the cache can be of two types. 1) the threads belonging to multi-programmed workloads where there are no shared data and 2) threads belonging to multi-threaded workloads which share data among them. In the case of multi-programmed workloads, the RD of a memory reference will either remain the same or increase. It remains same when no data interferes in that particular reuse interval. It increase when there are interference from other threads. While in multi-threaded workloads, the

$$\textit{Thread 1} \quad \overbrace{a, b, \underbrace{c, c}_{rd=0, td=1}, d, a}^{rd=3, td=5}, ..$$

$$\textit{Thread 2} \quad p, q, p, r, s, ..$$

$$\textit{Thread 3} \quad p, d, p, a, r, ..$$

$$\textit{Threads 1\&2} \quad \overbrace{a, p, b, \underbrace{c, q, c}_{rd=1}, d, p, a}^{rd=5}, r, s..$$

$$\textit{Threads 1\&3} \quad a, b, p, \underbrace{c, d, p, c}_{rd=2}, \underbrace{a, d, a}_{rd=1}, r, ..$$

Fig. 1. An example scenario illustrating the change in RD when threads interleave differently.

threads may bring some shared data to the cache which may then be used by other threads. This results in constructive interference, thereby decreasing the RD. Thus, the RD can remain the same, increase or decrease, compared to the standalone RD. The term concurrent reuse distance profile is used in the literature [24], [27] to refer the reuse distance profile of a thread which runs together with multiple threads sharing the cache. For consistency we use the same term in this paper, which is defined as follows.

Concurrent Reuse Distance (CRD). Concurrent reuse distance (CRD) with respect to a thread is the reuse distance of a data/memory reference when the thread is interfered by references from other threads. This situation occurs when many threads run together sharing a cache. We call the corresponding profile as CRD profile.

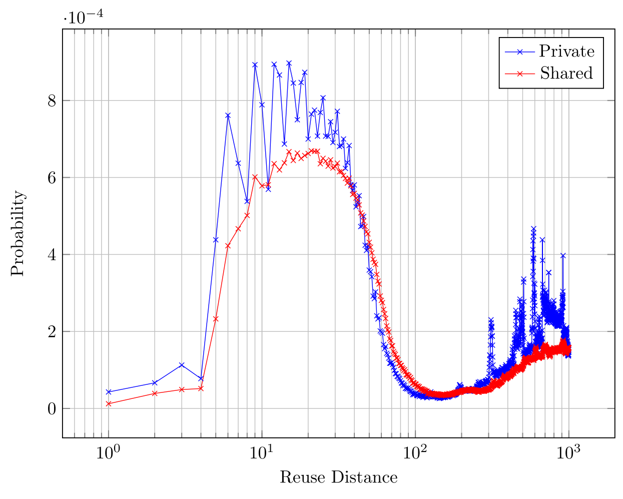
Fig. 1 shows an example of one possible way of interleaving of memory references from three threads and the reuse distance for the references a and c of thread 1. Thread 1 is independent of thread 2, while thread 3 shares data with thread 1. a and d are the shared data between thread 1 and thread 3. For thread 1, the second reference to a has a reuse distance of 3, c has a reuse distance of 0 and other blocks are not re-referenced at all and hence an RD of ∞. One of the probable interleaved traces, for the two cases of i) thread 1 running along with thread 2 and ii) thread 1 running along with thread 3 is also given in Fig. 1. In the first scenario, the RD for a has increased to 5, and for c it has changed to 1. In the second case, the RD of a has become 1, because of the access to a by thread 3 before the second access to a by thread 1. The modified reuse distance of the references in the interleaved trace is the CRD for that particular reference.

Fig. 2 shows the reuse distance profiles of two benchmarks vips and x264 of the PARSEC benchmark suite [44] obtained using the multi-core simulator Sniper [45]. The benchmarks are run with four threads each. The blue curve is the standalone RD profile of a thread when four threads run with a private cache each. The red curve corresponds to the CRD profile of a thread, when all threads share a cache. Two main observations can be made from the plot. First, the reuse signature is unique for an application. Next, the variation in RD profile when multiple threads run together. Finding the CRD profile through simulation is tedious compared to finding the standalone RD profile. Researchers have proposed efficient ways to accelerate CRD measurement like sampling based methods [26]. Also there may arise a need to find the CRD profile for different combination of threads. Repetition of full simulation every time will be a time consuming one. The aim of this paper is to fill this gap by proposing an analytical method to compute the CRD profile of the threads of multi-threaded applications from the standalone RD profile. Please note that standalone RD profile is equivalent to the private RD profile. However our formalism is not only applicable to get CRD profile from RD profiles of threads running on private caches. It can also be used in the case of hyper threads sharing the same cache.

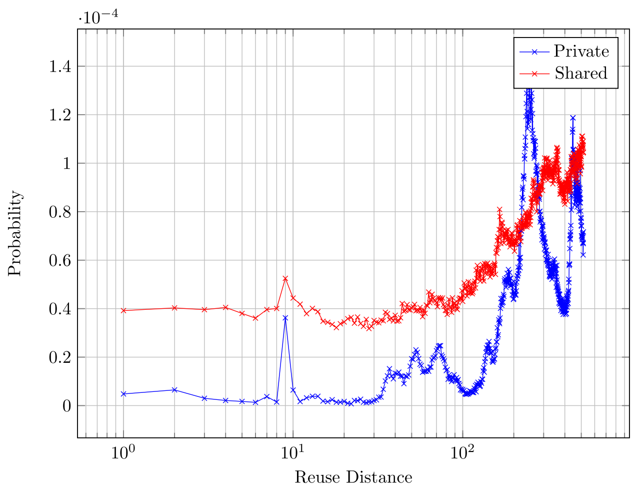
4 Inputs, Assumptions and Terminologies

4.1 Inputs

In order to proceed with the computation of the CRD profile we need to know the number of threads sharing the cache, the standalone RD profiles of all the threads, and the number of distinct blocks shared by the threads. Further we need certain statistics such as total number of cache



(a) PARSEC.vips



(b) PARSEC.x264

Fig. 2. Standalone RD profile of one of the four threads running with a private cache (blue) and CRD profile of that thread when the four threads share a cache (red). a) benchmark vips of PARSEC 2.1, b) benchmark x264 of PARSEC 2.1.

accesses, number of shared accesses, and number of distinct accesses made by a thread. We can then derive the fraction of cache access made by a thread, and the average time distance. Therefore our model takes the following input for constructing the CRD profile from the RD profile.

- T - The number of threads sharing the cache.
- N_i - Number of cache accesses made by thread i where $i \in \{0, 1, ..T-1\}$, during its entire lifetime.
- AF_i - Access Fraction of thread i. It is the fraction of accesses made by thread i over the accesses made by all the threads.

$$AF_i = \frac{N_i}{\sum_{j=0}^{T-1} N_j}.$$

- S_i, D_i - Number of shared accesses and number of distinct cache accesses respectively of each thread i.
- S - The number of distinct blocks that are shared by all the threads. This does not correspond to all the blocks in the shared cache.
- Set of blocks shared by all the threads denoted as $\mathbb{S}$.
- The standalone RD profile of each thread i.
- $\overline{TD}(x)$ is the average of TD obtained by averaging the TD of all reuse interval with an RD of x. The average TD is equal to the inverse of footprint, called fill time in the HTOL theory [37].

4.2 Assumptions

The following are the assumptions for our model.

1) The threads sharing the cache belong to a single multi-threaded application. Hence the threads will have shared data.
2) The RD profile is of cache block size granularity.
3) The number of distinct shared blocks, S is common to all the threads of the multi-threaded application. Luo et al. gives a solution to measure the amount of data (footprint) by a group of threads [46].
4) Each cache block has an equal probability of getting mapped to the cache sets. [47]. Though the mapping of cache blocks to cache sets follow a hash function (mod function or bit-selection) in general, Agarwal et al. [11] have shown the validity of this assumption especially when large data space is mapped to a small number of cache sets. The same assumption is also used in some of the recent analytical models [4], [16].
5) The references made to the cache are independent of each other [48].
6) The data cache access is uniform throughout the execution of the thread. i.e., we are not including the effect of phase change in our model.

   The assumptions 5 and 6 has also been made by previous analytical models and justified to be valid assumptions for cache based analytical models [4], [5], [14]

5 Computation of Concurrent Reuse Distance Profile

The basic idea in the construction of CRD profile is to find the probability with which an RD with value, say d in the standalone RD profile of a thread becomes $d \pm d', d' \geq 0$ when that thread runs together with other threads sharing the cache. We illustrate using an example how the standalone RD corresponding to the reuse interval a given thread t can change when some blocks from other threads interfere in that reuse interval.

$$\textit{Before Interference} \quad \overbrace{A, B, C, C, A}^{rd=2}$$

$$\textit{After interference} \quad \underbrace{A, X_1, B, X_2, X_3, C, X_4, C, A}_{crd=rd\pm d}$$

Fig. 3. Illustration for getting CRD profile from standalone RD profile.

In the following we use the notation rd to denote a reuse distance in the standalone RD profile of a thread, and crd to stand for the reuse distance in the concurrent RD profile of the thread (obtained as a result of interference from other threads).

In order to get intuitive feel on how the crd are affected by the interfering threads we consider the following example. Consider a reuse interval with respect to a target block A of the thread t. Fig. 3 shows two snapshots of the contents of the reuse interval (i) when thread t is running alone, i.e., before interference from other threads and (ii) after interference from other threads. $X_i, i \in 1, ..4$ are the blocks from other threads that interfere in the reuse interval. A given X_i can be,

1) distinct from all blocks originally present within the RI and from other X_i's, or
2) same as one of the blocks in the original RI or
3) Distinct from the blocks in the original RI, but same as one or more of other interfering blocks X_i s, or
4) same as the target block A.

Every X_i falling under category (1) increases the crd by one. Any X_i falling under category (2) will not change crd. In case of category (3), every set of X_is which are all the same block, but distinct from the blocks in the original RI, and other X_is will increase crd by one per set. Every block belonging to category (4) will split the RI. For the example given in Fig. 3, we can deduce the change in the value of crd, for different scenarios as follows.

1) All the X_i's are different from the blocks B, C, A of thread t. In this case,

$$crd = rd + \text{No. of distinct blocks among } X_i$$

2) Some of the X_i's are same as the blocks B or, C, but not A, of thread t. In this case,

$$\begin{aligned} crd = rd + \text{No. of blocks in } X_i \text{ which are distinct} \\ \text{among themselves and also distinct from the} \\ \text{blocks within the reuse interval of thread } t. \end{aligned} \tag{1}$$

   The number of distinct block in X_i has been derived by Xiang et al. in the HOTL theory and is given by the footprint $fp(|Xi|)$ [37].

3) Suppose one of the X_i's is same as the target block A of the reuse interval. In this case the reuse interval

will split. The crd will depend upon where the block which is same as the target block is accessed. In the example of Fig. 3, if X_4 is A, the crd of A will decrease to one. If X_1 is A, then the crd can be higher from the standalone rd depending upon the number of distinct blocks of X_2, X_3, X_4.

In the cases 1 and 2 above, a particular crd value is generated from a given rd without any split in the original reuse interval. While in the third case, the crd is formed due to a split in the RI.

The probability of occurrence of a particular crd for thread t in the interleaved trace, given the reuse interval doesn't split can be written as

$$\begin{aligned}
&P(crd = x|\text{ no split })\\
&= P(rd = 0) \times P(x \text{ distinct blocks interfere } |rd = 0)\\
&\quad + P(rd = 1) \times P(x-1 \text{ distinct blocks interfere } |rd = 1)\\
&\quad + .. + P(rd = x) \times P(\text{no distinct blocks interfere } |rd = x)\\
&P(crd = x|\text{ no split })\\
&= \sum_{i=0}^{x} P(rd = i) \times P(x-i \text{ distinct blocks interfere}|rd = i)
\end{aligned} \tag{2}$$

We derive the term $P(x-i \text{ distinct blocks interfere } |rd = i)$ in the following section.

5.1 Calculation of $P(x-i$ Distinct Blocks Interfere$|rd = i)$

Consider a reuse interval of thread t with the corresponding value of RD as d_t. Consider all the reuse intervals in the reference string whose reuse distance is d_t. The time distance of these reuse intervals can take any value $\geq d_t$. Let the average time distance ($\overline{TD}$) corresponding to the RD of d_t be equal to n_t. For simplicity, we denote $x-i$ as d in the following. Let d denote the number of distinct blocks from threads other than t that interfere in an RI. This can be as a result of k accesses ($k \geq d$) interfering with the RI, out of which d blocks are distinct.

In should be clear from the above discussions that not all interfering accesses will affect the CRD, only accesses that are distinct from earlier accesses matter. Therefore we are particularly interested in tracking distinct accesses. In an RI (or in a sequence of accesses), we refer an access to a block b as *distinct access*, if the block b has not been accessed so far in that RI. We use the term *distinct block* interchangeably with distinct access.

Suppose k accesses interfere in an RI having $\overline{TD} = n_t$, then there will be totally $k + n_t$ accesses in the interleaved trace corresponding to that interval. Independent of other accesses, each access in the interleaved trace can either belong to the thread t with probability AF_t or can belong to any one of the remaining threads with the probability $AF_{rem} = 1 - AF_t$. Hence the probability of having k interferences from other threads in an RI having $\overline{TD}(d_t) = n_t$ can be given by

$$\begin{aligned}
&P(k \text{ interferences from other threads } |RD = d_t)\\
&= \binom{k+n_t}{k} (AF_{rem})^k (AF_t)^{n_t}.
\end{aligned} \tag{3}$$

Out of the k accesses d accesses must be distinct. Let $P(k, d)$ be the probability of having d distinct accesses out of k interfering accesses in an RI. These d blocks are distinct among themselves and also from the d_t blocks of thread t. Hence the probability of d distinct blocks interleaving in an RI with reuse distance d_t can be given by

$$\begin{aligned}
&P(d \text{ distinct blocks interfere } |RD = d_t)\\
&= \sum_{k=d}^{\infty} \left(\binom{k+n_t}{k} (AF_{rem})^k (AF_t)^{n_t} \right) P(k, d).
\end{aligned} \tag{4}$$

Theoretically the upper limit of k in the above equation can take values up to infinite. But for numerically solving the equation it has to be limited to an upper value, which is further discussed in Section 8.3.

5.2 Calculation of $P(k, d)$

In this section, we derive the probability of having d distinct accesses out of k interferences in a reuse interval of thread t with an RD of d_t.

Consider a sequence of k interfering accesses from threads other than t in the reuse interval of the thread t. We traverse the interfering accesses one by one starting from the first access and incrementing the number of accesses (k) and number of distinct blocks (d) encountered so far. At each access, we check whether that access is distinct from the encountered accesses and from the d_t distinct blocks of the thread t.

Consider the example given in Fig. 3, for the case the interfering accesses are $X_1 = D$, $X_2 = B$, $X_3 = E$, and $X_4 = C$. Then the access pattern after interference will be $A, D, B, B, E, C, C, C, A$. The tuple (k, d) will change after each successive accesses X_1, X_2, X_3 and X_4 as follows : (0, 0), (1, 1), (2, 1), (3, 2), and (4, 2). Note that the number of distinct blocks after a given access can go up by 1 or remain the same compared to the previous accesses. Further the probability that the next access is distinct is independent of the number of accesses made and number of distinct accesses already encountered.

This property allows us to define a Markov chain model with (k, d) as the state and compute $P(k, d)$. Hence we define a state of the traversing (Markov)process as follows

Definition 1. [2]*A state X_n at any integer time n denoted as the pair (k, d),is characterized by two state variables k, and d, where k is the number of accesses traversed so far and d is the number distinct blocks out of the k accesses encountered till the time n.*

Here $0 \leq d \leq k$. The number of possible states that can be visited after k accesses is $k + 1$.

As will be shown in the following discussion the next state X_{n+1} visited depends only on the present state X_n and not on how the present state was reached. Also, the probability of transition from one state to another doesn't depends on time n. Hence the process $\{X_n; n \geq 0\}$ defined

2. The tuple (k, d) differs from the footprint definition [37] in that (k, d) is not directly measured from the trace. We just use it as an intermediate value pair, to arrive at the computation probability of d distinct interferences from other threads.

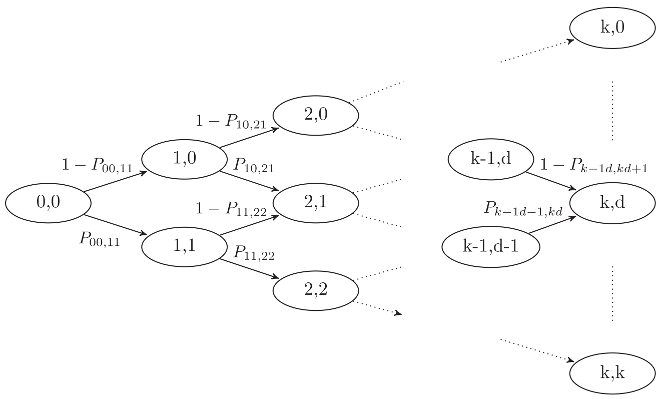


Fig. 4. Markov chain for the determination of number of distinct accesses out of k interfering accesses. State (k, d) indicates that d out of k accesses made so far are distinct. The transition probability $P(k, d|k-1, d-1)$ is denoted as $P_{k-1d-1,k,d}$.

by the random variable $X_n = (k, d)$, is a Markov process. It is a finite state discrete time Markov chain because the state is defined at integer time n and the values X_n take is countably finite. With this definition the steady state occupancy probability of the Markov chain denoted as $P(k, d)$ become the same as the term $P(k, d)$ that occurs in (4).

Fig. 4 shows the state transitions of the Markov chain. We take the initial state as (0, 0) before starting the traversal. The probability of being in the initial state is 1. The state changes to (1, 1), if the block corresponding to the first interfering access is distinct from the d_t blocks of thread t. If it is not distinct, the state changes to (1, 0). Similarly each access is traversed and the corresponding state changes are made as shown in Fig. 4. For ease of representation in Fig. 4, the transition probability $P(k, d|k-1, d-1)$ is denoted as $P_{k-1d-1,k,d}$. We compute the probability of being in a state (k, d), $P(k, d)$, as follows. From the Fig. 4, the state (k, d) can be reached from the states $(k-1, d-1)$ or $(k-1, d)$. Hence the probability $P(k, d)$ can be given as follows

$$\begin{aligned} P(k,d) &= P(k,d|k-1,d-1) \times P(k-1,d-1) \\ &\quad + P(k,d|k-1,d) \times P(k-1,d). \end{aligned} \tag{5}$$

In the above equation $P(k, d|k-1, d-1), d \geq 1$ is the probability that the kth access is distinct when there are $d-1$ distinct accesses out of the $k-1$ accesses happened so far. It is nothing but the state transition probability from the state $(k-1, d-1)$ to the state (k, d). Also note that $P(k, d-1|k-1, d-1) = 1 - P(k, d|k-1, d-1)$ since $P(k, d-1|k-1, d-1)$ is the probability that the kth access is not a distinct access when there are $d-1$ distinct accesses out of the $k-1$ accesses happened so far.

5.2.1 Calculation of $P(k, d|k-1, d-1)$

In this section, we derive $P(k, d|k-1, d-1)$, probability for the kth access to be distinct when there are $d-1$ distinct accesses out of the $k-1$ accesses happened so far. Let the kth access be made by thread i. The kth access belongs to thread i with a probability AF_i, which is nothing but the access fraction of thread i.

Out of the $d-1$ distinct accesses happened so far, let there be j blocks from thread i. The probability for j blocks to belong to thread i among the $d-1$ blocks is

$$\binom{d-1}{j}\Big(\frac{D_i}{\sum_{l,l\neq t} D_l}\Big)^j\Big(1 - \frac{D_i}{\sum_{l,l\neq t} D_l}\Big)^{d-1-j},$$

where $\frac{D_i}{\sum_{l,l\neq t} D_l}$ is the probability for the distinct block to belong to thread i.

If there are j distinct blocks from thread i out of $d-1$ blocks, the kth block will be distinct if its reuse distance is greater than j. The probability for the RD of thread i to take a value greater than or equal to j, $P_i(rd \geq j)$ can be given as

$$\begin{aligned} P_i(rd \geq j) &= 1 - P_i(rd < j) \\ &= 1 - F_{RD_i}(j-1), \end{aligned}$$

where F_{RD_i} denotes the cumulative distribution function of the RD for the thread i.

Since the interfering thread i can be any one of the threads from 0 to $T-1$, $i \neq t$, $P(k, d|k-1, d-1)$ can be written as

$$\begin{aligned} &P(k,d|k-1,d-1) = \\ &\quad \sum_{i=0,i\neq t}^{T-1} AF_i \sum_{j=0}^{d-1} \left(\binom{d-1}{j} \left(\frac{D_i}{\sum_{l,l\neq t} D_l}\right)^j \right. \\ &\quad \left. \left(1 - \frac{D_i}{\sum_{l,l\neq t} D_l}\right)^{d-1-j} \right) \left(1 - F_{RD_i}(j-1)\right). \end{aligned} \tag{6}$$

Equation (6) only gives the probability for the kth access to be distinct from the blocks of same thread occurred as far in the $k-1$ accesses. However, it is possible for the kth access to be shared block. Therefore we should also enforce the condition that the kth access be distinct from the blocks of other threads in the reuse interval. The number of blocks belonging to threads other than i in the interleaved trace of the reuse interval are the d_t blocks of thread t, including the target block, and $d-1-j$ blocks from threads other than i and t. Hence the kth access has to be different from $d_t + d - 1 - j$ blocks. Let $P_{diff}(d_t + d - 1 - j)$ denote the probability that the kth access is different from $d_t + d - 1 - j$ blocks given the access is by thread i,[3] the derivation of which is given in Section 5.2.2

Hence Equation (6) should be modified to write $P(k, d|k-1, d-1)$ as

$$\begin{aligned} P(k,d|k-1,d-1) &= \sum_{i=0,i\neq t}^{T-1} AF_i \sum_{j=0}^{d-1} \left(\binom{d-1}{j} \right. \\ &\quad \left. \left(\frac{D_i}{\sum_{l,l\neq t} D_l}\right)^j \left(1 - \frac{D_i}{\sum_{l,l\neq t} D_l}\right)^{d-1-j} \right) \\ &\quad \times \left(1 - F_{RD_i}(j-1)\right) \times P_{diff}(d_t + d - 1 - j). \end{aligned} \tag{7}$$

5.2.2 Calculation of P_{diff}

Let the kth access belongs to thread i and b be the block corresponding to it. Hence for the block b to be different from $d_t + d - 1 - j$ blocks, it has to be different from the d_t blocks of thread t and $d - 1 - j$ blocks of threads other than t and i.

3. For notation simplicity we are not denoting explicitly that the access is by thread i in $P_{diff}(d_t + d - 1 - j)$.

Let $P^t_{diff}(d_t)$ be the probability that the block b be different from the d_t blocks of thread t and $P^{rem}_{diff}(d-1-j)$ be the probability that it is different from $d-1-j$ blocks belonging to threads other than t. $P_{diff}(d_t+d-1-j)$ is the product of these two probabilities, since these events are independent. $P_{diff}(d_t+d-1-j)$ can therefore be written as

$$P_{diff}(d_t+d-1-j) = P^t_{diff}(d_t) \times P^{rem}_{diff}(d-1-j). \tag{8}$$

To derive $P^t_{diff}(d_t)$, we first derive the probability for one block, say b' in d_t to be same as the block b denoted as P_{same}. For two blocks from two different threads to be same, both the blocks have to shared blocks. Hence, P_{same} can be written as

$$\begin{aligned} P_{same} = & \\ & P(b \in \mathbb{S} | k^{th} \text{ access is by thread } i) \times \\ & P(b' \in \mathbb{S}) \times P(b = b' | b, b' \in \mathbb{S}) \\ & = \frac{S}{D_i} \times \frac{S}{D_t} \times \frac{1}{S} = \frac{S}{D_i} \times \frac{1}{D_t}. \end{aligned} \tag{9}$$

Hence a block in d_t will be different from the block b with probability $1 - P_{same}$. Therefore, the probability for the block b to be different from all the d_t blocks is

$$P^t_{diff}(d_t) = (1-P_{same})^{d_t} = \left(1 - \frac{S}{D_i} \times \frac{1}{D_t}\right)^{d_t}. \tag{10}$$

Similarly for $P^{rem}_{diff}(d-1-j)$ the term $\frac{1}{D_t}$ can be replaced by $\frac{1}{\sum_{j, j \neq i,t} D_j}$, since the $d-1-j$ blocks can belong to threads other than i and t. Thus, $P^{rem}_{diff}(d-1-j)$ can be written as

$$P^{rem}_{diff}(d-1-j) = \left(1 - \frac{S}{D_i} \times \frac{1}{\sum_{j,j\neq i,t} D_j}\right)^{d-1-j}. \tag{11}$$

Substituting (10) and (11) in Equation (8) gives

$$P_{diff}(dt+d-1-j) = \left(1 - \frac{S}{D_i} \times \frac{1}{D_t}\right)^{d_t} \times \left(1 - \frac{S}{D_i} \times \frac{1}{\sum_{j,j\neq i,t} D_j}\right)^{d-1-j}. \tag{12}$$

5.3 Including the Effect of Reuse Interval Split

In this section, we first derive the probability for the occurrence of a particular crd due to a split in the RI of thread t, $P(crd = x|\text{split})$. We then give the total probability for the occurrence of a particular crd, $P(crd = x)$.

Revisiting the example of Fig. 3, a reuse interval will split when any of the blocks (X_i's) interfered from other threads in the RI of thread t is same as the target block. Also, there is a possibility for one or more X_i's being same as the target. In this case, the last access made to the target block by other threads should be considered for the crd. To capture these possibilities, we first find the probability for number of distinct blocks, say d that can interfere in the reuse interval of thread t with an RD of d_t using Equation (4). The reuse interval will split when one of the d distinct blocks interfered is same as the target block.

Let $P_{split}(d)$ gives the probability for the reuse interval to split. The probability for one block among the d blocks to be same as the target block (the derivation is similar to Equation (9)) is

$$P_{same} = \frac{S}{D_t} \times \frac{S}{\sum_{i,i\neq t} D_i} \times \frac{1}{S} = \frac{S}{D_t} \times \frac{1}{\sum_{i,i\neq t} D_i}.$$

All the d blocks will be different from the target block with probability $(1-P_{same})^d$. Hence, the probability for at least one block of d blocks to be same as the target block, which is nothing but $P_{split}(d)$ can be given as follows

$$P_{split}(d) = 1 - (1-P_{same})^d. \tag{13}$$

Note that $P_{split}(d)$ is a function of number of distinct blocks, d, that interfere in the reuse interval. If d is higher, the probability that the reuse interval will split will be higher.

If the reuse interval splits, since any of the d blocks can be same as the target block and it can fall into any position in the interval including the d_t blocks of thread t, the resulting reuse distance can take any value uniformly in the range $\{0, 1, ..d + d_t - 1\}$. Hence the crd will be any value in $\{0, 1, .., d + d_t - 1\}$ with probability $\frac{1}{d+d_t}$, if the reuse interval splits. i.e.,

$$P(crd = x|\text{ split }) = \sum_{i=0}^{rd_{max}} P(rd = i) \times \left(\sum_{j=1}^{i} P(x-i+j \text{ distinct blocks interfere}|rd = i) * \frac{1}{x+j}\right). \tag{14}$$

Combining Equations (2) and (14), the probability for the occurrence of a particular crd, $P(crd = x)$ can be written as

$$\begin{aligned} P(crd = x) & \\ = \sum_{i=0}^{x} & P(rd = i) \times P(x-i \text{ distinct blocks interfere}|rd = i) \\ & * (1 - P_{split}(x-i)) + \sum_{i=0}^{rd_{max}} P(rd = i) \\ & \times \left(\sum_{j=1}^{i} P(x-i+j \text{ distinct blocks interfere}|rd = i) \frac{P_{split}(x-i+j)}{x+j}\right). \end{aligned} \tag{15}$$

The upper bound of i in (14) and (15) denoted as rd_{max} can theoretically take value up to ∞. However for the purpose of carrying out numerical computation rd_{max} can be taken to be the maximum value of RD for which $P(rd = rd_{max})$ is non-zero. Equation (15) is our main contribution.

It is to be noted that Equation (15), is not applicable for calculating $P(crd = \infty)$. All distinct accesses made by thread t for the first time will contribute to $rd = \infty$, and also to $crd = \infty$. However some of the shared blocks accessed by t for the first time may not led to a miss if that shared block is brought into the cache by an interleaving thread (at any position starting from the beginning of the reference string to the position of the target block). The computation of the probability of such events is difficult, but can be assumed to be small. So we can safely assume in our analytical model that $P(crd = \infty) \approx P(rd = \infty)$.

5.4 Limitations of our Model

An approximation invoked by us is in the derivation of $P(crd = \infty)$ as outlined in the previous section. A key assumption made in our formulation as listed in Section 4.2 is that the references made to the cache are independent of each other. This assumption is implicit in the usage of the binomial distribution used at various steps in our derivation (Equations (7), (9), and (12)). The binomial expression assumes that the accesses are independent and the blocks have the same probability of getting mapped onto any given set. On the other hand accesses are correlated.

Our formulation partially accounts for the correlated nature of the cache accesses. Indeed a key ingredient in our derivation is the RD profile that captures the probability with which an access to a block will get repeated after few references. Binomial formula is applicable if the underlying processes is assumed to be a Bernoulli process (IID process). However when threads interleave their memory accesses the probability (that the given access is distinct) varies from one to access to another. Further it depends on the history of accesses made so far. It is not clear what mathematical formulation exists to deal with such cases which is also tractable.

An assumption made by us in our model (Section 4.2) is that all threads share the same set of blocks. But in real scenario there may be some blocks that are shared among a subset of all threads. One such example program has been given by Luo et al. wherein most data is either not shared or shared by no more than two threads, even if the benchmark uses 8 threads [46].

6 Finding Reuse Distance Profile of Application

In this section, we analytically derive the combined RD profile of a group of threads sharing the cache. It can be either all the threads of multi-threaded applications or a few threads of it. This combined RD profile will help us to determine the total cache miss rate directly. Let there be T threads sharing the cache and we are interested in finding the combined RD profile of these T threads. Let N_i be the total number of cache references made by thread $i, i = 0, 1, \ldots T-1$

$$P(combined\ rd = x) = \frac{\text{No. of references with } (rd = x) \text{ in the interfered trace}}{\text{Total number of references}} \quad (16)$$

$$\text{No. of references with } (rd = x) = \sum_{i=0}^{T-1} N_i \times P_i(crd = x), \quad (17)$$

Substituting Equation (17) in Equation (16) will give the combined RD profile of all the threads sharing the cache.

$$\begin{aligned} P(combined\ rd = x) &= \frac{\sum_{i=0}^{T-1} N_i \times P_i(crd = x)}{\text{Total number of references}} \\ &= \sum_{i=0}^{T-1} AF_i \times P_i(crd = x), \end{aligned} \quad (18)$$

where $P_i(crd = x)$ can be obtained from Equation (15).

7 Modeling Private Caches

When the threads of multi-threaded applications run with private caches for each thread, a write to a block by a thread in one cache invalidates the copies of that block in other caches. If an access is made to this invalidated block in other cache, it results in miss independent of that access's reuse distance. This miss is called coherence miss. For reuse distance calculation with respect to the thread incurring coherence miss, this access has to be taken with an RD of infinite. While measuring the standalone RD profile, this effect won't be counted in. Hence we derive the coherent RD profile of each thread taking into account the coherency in this section. The coherent RD profile changes as follows compared to the standalone RD profile

- If a reference with finite RD results in a coherence miss, then its RD will become infinite.
- The above change results in the increase in number of references with infinite RD and decrease in the number of references with finite RD. Since the RD profile is normalized, the shape of the coherent RD profile will be different from the standalone RD profile.

To derive the coherent RD profile, we need one more input apart from the inputs listed in Section 4.1. It is the number of writes to shared blocks of each thread. Let SW_i be the number of shared writes made by each thread. This can be calculated during the one-time profiling of applications that is used to collect the RD profile and other inputs. We first calculate the probability for a reference to result in coherence miss and then we use it to derive the coherent RD profile.

Consider a reuse interval (RI). Let the reuse distance of the RI be d_t. Let n_t be its average time distance. During this interval, there will be on an average

$$n_{rem} = n_t \times \frac{AF_{rem}}{AF_t}, \quad (19)$$

accesses made by other threads to different private caches.

The target access of the RI will result in a coherence miss, if it is a shared access and at least a write has been made to the target block by other threads in a different private cache during the same time interval in which the RI has occurred. We first find the probability for one access among the n_{rem} accesses to be a shared write to the block same as the target block.

The probability for the target access to be shared is

$$P(\text{target access is shared}) = \frac{S_t}{N_t}. \quad (20)$$

The probability for an access among the n_{rem} accesses to be a shared write, denoted as $P(SW)$ is

$$P(SW) = \frac{\text{No. of shared writes from threads other than } t}{\text{Total no. of accesses from threads other than } t} = \frac{\sum_{i,i\neq t} SW_i}{\sum_{i,i\neq t} N_i}. \tag{21}$$

The probability for the block to which shared write is made in other private caches to be same as the target block is $\frac{1}{S}$. Hence, the probability for an access among the n_{rem} accesses to be a shared write to the same block as target is

$$P(\text{shared write to target}) = \frac{S_t}{N_t} \times \frac{\sum_{i,i\neq t} SW_i}{\sum_{i,i\neq t} N_i} \times \frac{1}{S}. \tag{22}$$

Since there are n_{rem} accesses being made by other threads in the same time duration in which RI has occurred, the probability for at least one shared write to the target block in other private caches is

$$P_{coherence}(rd = d_t) = 1 - \left(1 - \frac{S_t}{N_t} \times \frac{\sum_{i,i\neq t} SW_i}{\sum_{i,i\neq t} N_i} \times \frac{1}{S}\right)^{n_{rem}}. \tag{23}$$

The probability for the occurrence of a particular coherent reuse distance denoted as $chrd$ can now be written as

$$\begin{aligned} &P(chrd = d_t) \\ &\quad = P(rd = d_t) - P(rd = d_t) \times P_{coherence}(rd = d_t) \\ &\qquad = P(rd = d_t)(1 - P_{coherence}(rd = d_t)), \end{aligned}$$

and the probability for the occurrence of a reuse distance of infinite is

$$\begin{aligned} &P(chrd = \infty) = \\ &P(rd = \infty) + \sum_{i=0}^{rd_{max}} P(rd = i) \times P_{coherence}(rd = i), \end{aligned} \tag{24}$$

where rd_{max} is the maximum finite value of RD for which $P(rd = rd_{max})$ is non-zero.

8 Model Validation

We validate our analytical model against simulation using the multi-core simulator Sniper [45] for the PARSEC version 2.1 [44] and the SPLASH-2 benchmark suites [49]. We use six applications from the PARSEC and eight applications from the SPLASH-2 benchmark suite respectively for validating our model. We have done the validation with respect to L2 cache. In other words, we assume the L2 cache as shared and the RD profiles are computed for the references (stream of misses from L1 cache) made to L2 cache. Our model can be applied for L3 level too, provided the RD profiles are collected with respect to L3 level.

To validate the model, we first collect the standalone RD profile of each thread of the applications through simulation. Along with the RD profile, we collect the inputs mentioned in Section 4.1. We made appropriate code modifications in the simulator to measure the RD profile and the other inputs.

TABLE 1
Simulator Configuration

No. of cores	4
Logical cores per cpu	1
No. of levels of cache	3
L1-Icache	private, 32 KB, 4-way
L1-Dcache	private, 32 KB, 4-way
L2 cache	variable
coherence protocol	MESI

8.1 Workloads

The PARSEC provides six input sets for its benchmark programs. Out of these input sets small, medium and large input sets are used mainly for micro-architectural simulation. We used the large input set for the both PARSEC and SPLASH-2 benchmark suites respectively. We ran all the benchmarks with four threads for our validation.

8.2 Simulator Configuration

We use the cache-only mode of the simulator Sniper while taking the performance metrics, since we are interested only in the cache related parameters. Table 1 gives the simulator configuration. Sniper provides the configuration files for certain processor architectures. We use the configuration called *gainstown* which internally uses the Intel's Nehalem core model.

We use a 2 MB L2 cache with 8-way associativity, since we are considering the L2 cache as shared. The configuration of L2 cache can be varied, since the RD profile doesn't depend upon the configuration of L2 cache. However, the RD profile with respect to L2 cache depends upon the configuration of L1 cache, since the references made to L2 cache are actually misses from L1 cache. The standalone RD profile is measured, keeping the L2 cache private for each thread. The CRD profile is measured by changing the L2 cache to shared, without changing its other configuration.

8.3 Numerical Computation

The analytical equations can be solved using any numerical solver. We use Python to solve the equations. To aid handling of large values, we use the package numpy in Python. Most of the equations (especially Equations (4) and (15)) if computed directly, will consume more time. To optimize the computation, certain intermediate probability values $(p(k, d|k-1, d-1), p_{dist})$ in these equations can be precomputed and stored in table. The values stored in the table can then be read and used for computation.

The upper limit in Equation (4) can be fixed based on n_t. For particular n_t the average number of accesses that can interfere in an interval can be found using $n_{rem} = n_t * \frac{AF_{rem}}{AF_t}$. The probability for the number of interferences exceeding n_{rem} will be very less. Hence we fix the upper limit to five times n_{rem}. Similarly the upper limit for the variable j in Equation (15) can also be set depending up on the value of d_t (or n_t which is dependent up on d_t) in the numerical computation.

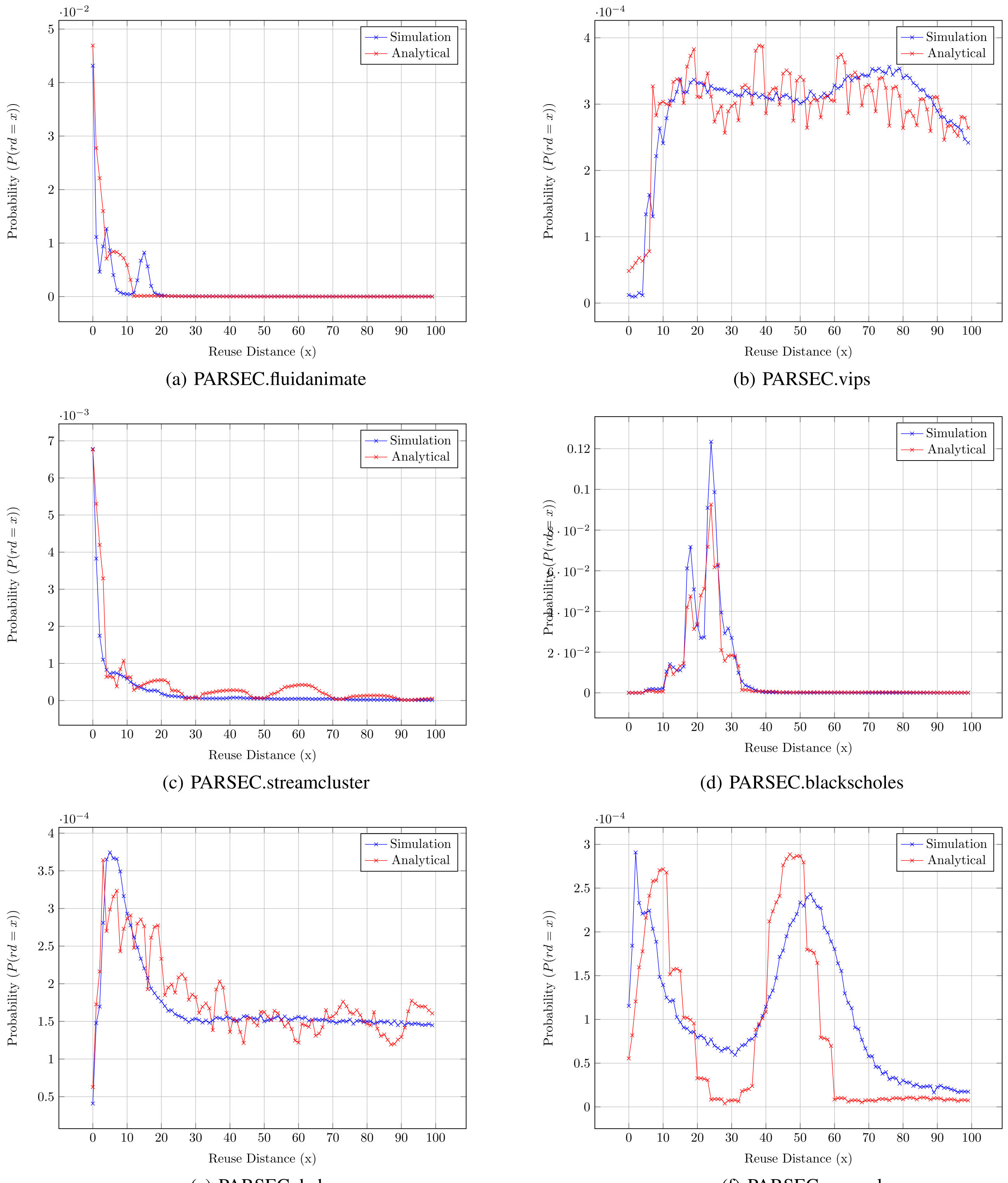


Fig. 5. Comparison of simulation and analytical results of concurrent reuse distance (CRD) profile when 4 threads share the L2 cache for the configuration given in Table 2 for six benchmarks of PARSEC-2.1 benchmark suite.

8.4 Validation Results

8.4.1 CRD Profile of a Single Thread

Fig. 5 shows the CRD profile obtained by simulation and by analysis (Equation (15)) using our model, for six benchmarks namely fluidanimate, vips, streamcluster, blackscholes, dedup and canneal of the PARSEC benchmark suite. Fig. 6 shows the CRD profile of eight benchmarks namely radiosity, cholesky, fft, radix, raytrace, water-nsquared, fmm and lu of the SPLASH-2 benchmark suite. We have shown the RD value up to hundred only in the plots. After the RD value of 100, the values are either very less or closely follows the trend of the CRD profile up to hundred. Fig. 7 shows that there are few exceptions such as SPLASH2.Barnes and PARSEC.X264 for which the deviation between simulation and numerical

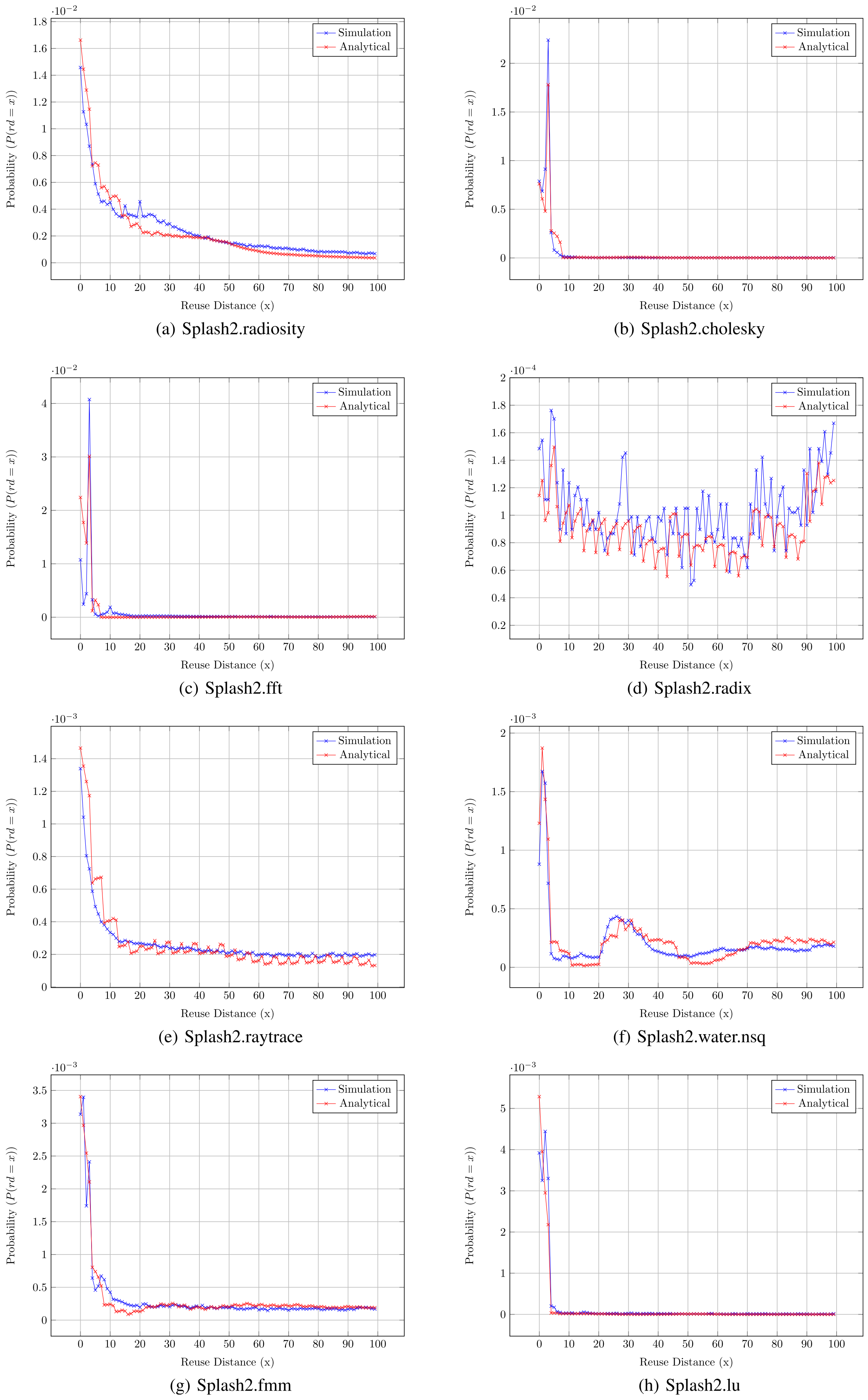
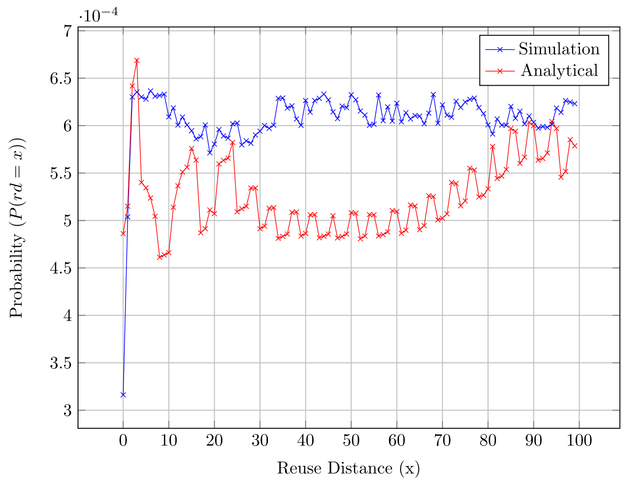
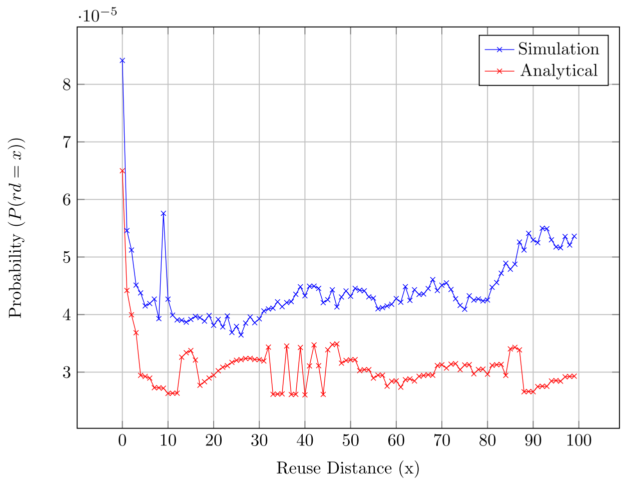


Fig. 6. Comparison of simulation and analytical results of concurrent reuse distance (CRD) profile when 4 threads share the L2 cache for the configuration given in Table 2 for eight benchmarks of SPLASH-2 benchmark suite.



(a) Splash2.Barnes



(b) PARSEC.X264

Fig. 7. Comparison of simulation and analytical results of concurrent reuse distance (CRD) profile when 4 threads share the L2 cache for the configuration given in Table 2 for the Barnes benchmark of SPLASH-2 and X264 benchmark of PARSEC benchmark suites respectively.

results of CRD profile are significant. The value of $P(crd = \infty)$ for the CRD profile obtained through simulation for different benchmarks of SPLASH2 are radiosity - 0.143, cholesky - 0.1076, fft - 0.296, radix - 0.1943, raytrace - 0.0267, waternsq - 0.0033, fmm - 0.0662, lu - 0.04495, and barnes - 0.0077. The value of $P(crd = \infty)$ for the CRD profile obtained through simulation of some of the benchmarks of PARSEC are fluidanimate - 0.4184, vips - 0.0184, streamcluster - 0.00164, and x264 - 0.09971.

8.4.2 *Combined RD Profile of Application*

This section explains the validation results for the prediction of CRD profile of all the threads sharing the cache, as explained in Section 8.4.2. While doing the simulation we observed the following phenomenon. When each thread of the multi-threaded application has a similar profile and metrics, the CRD profile of a single thread running together with other threads matches closely with the combined RD profile of all threads. Fig. 8 shows such a behavior for the benchmarks canneal and fluidanimate of the PARSEC benchmark suite and for raytrace and fmm of the SPLASH benchmark suite respectively. Both the values (blue and red curve) in Fig. 8 are obtained though simulation. The reason behind this behavior can very well be observed from the Equation (18). When the threads have similar profile, $P_i(crd = x)$ can be taken out as common for all threads. The remaining terms sum up to the total number of references, that cancels out with the denominator. This gives $P_i(crd = x)$ as result.

Fig. 9 gives the comparison of predicted results and simulation results for the combined RD profile for the benchmarks streamcluster of the PARSEC benchmark suite and radix of the SPLASH benchmark suite. For many benchmarks the the combined RD profile follows the shape of the CRD profile, as explained in the last paragraph. Also the combined RD profile is a straight forward extension of the CRD profile. Hence, we show the results for only two benchmarks.

8.4.3 *Coherent RD Profile*

Fig. 13 shows the validation results of coherent RD profile. We have done the coherent RD profile computation at L2 level, keeping the L2 cache private. We have shown the results for two representative benchmarks with high sharing pattern namely dedup and bodytrack, hence have higher variation in coherent RD profile compared to standalone RD profile. For the benchmarks with low sharing pattern, there is not much change in the the coherent RD profile compared to the standalone RD profile. Hence we are not giving the results for all benchmarks.

8.5 CRD Profile for Varying Number of Threads

In this subsection we demonstrate that the proposed analytical model can be used to compute CRD profile for arbitrary number of interfering threads. Figs. 10, 11, and 12 show the comparison of the simulation results and analytical results for the CRD profile by varying the number of threads for the benchmark Splash2.fmm. The simulator configuration used is given in Table 2. We find that the shape of the RD profile does not vary as the thread increases, as it is an inherent characteristic of a benchmark. The values of $P(crd = x)$, changes only slightly. Since the CRD profile represent a normalized probability distribution, if the shape of the profile is same the values (probability density) also does not change much. The analytical model captures the trend exhibited by the simulation results.

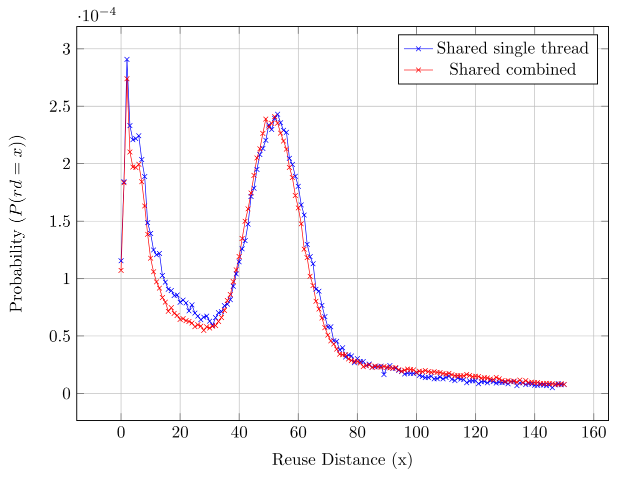
8.6 Error Calculation

In order to measure the effectiveness of the predicting ability of our analytical model, we use two metrics of error calculation. We calculate the Root Mean Squared Error(RMSE) and Normalized RMSE (NRMSE) of the values obtained through simulation and analytical derivation. *RMSE :* The RMSE for a set of actual values, $X_{act,i}$ and estimated values, $X_{est,i}$ is given by

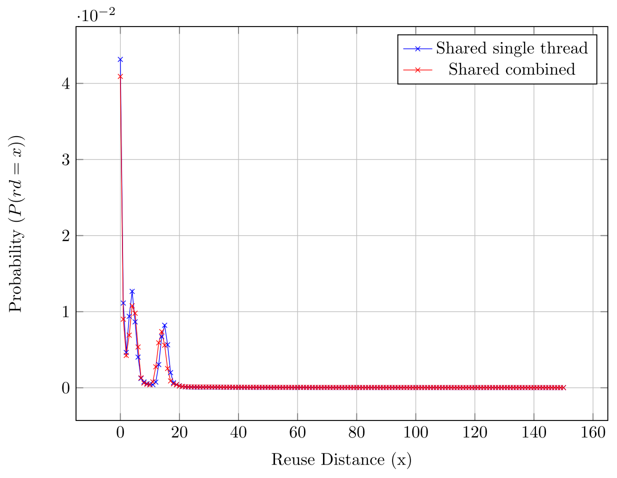
$$RMSE = \sqrt{\frac{\sum_{i=0}^{n}(X_{act,i} - X_{est,i})^2}{n}}.$$

For our validation, X_{act} is the set of results obtained through simulation and X_{est} is the set values got through our analytical model.

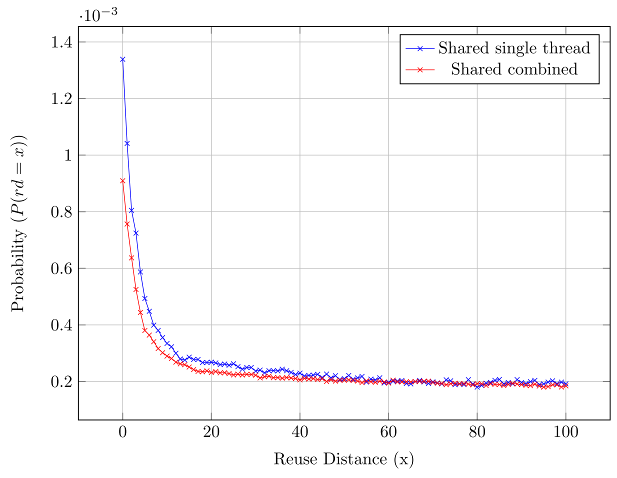
NRMSE : Non-dimensional forms of the RMSE is often used to compare the RMSE with different units. The RMSE



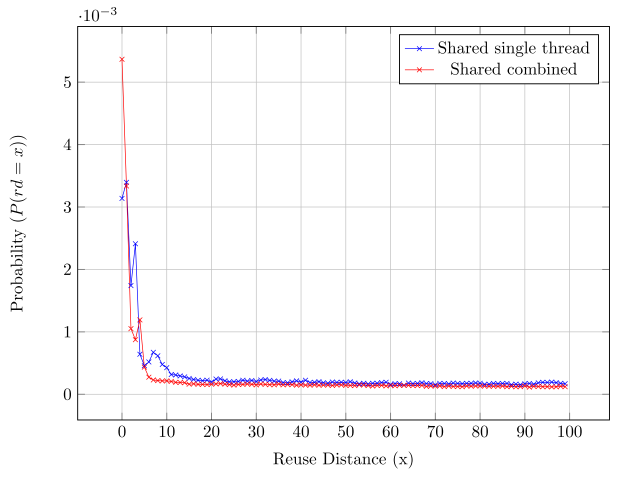
(a) PARSEC.canneal



(b) PARSEC.fluidanimate

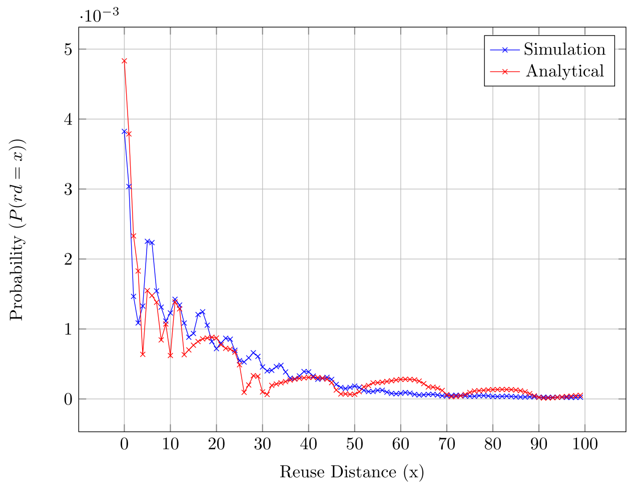


(c) SPLASH.raytrace

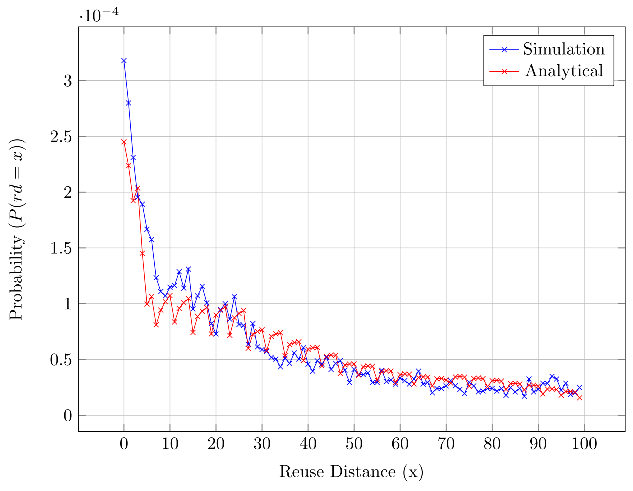


(d) SPLASH.fmm

Fig. 8. Comparison of simulation results of CRD profile of a thread sharing the cache with other threads and combined RD profile of all the threads sharing the cache for two benchmarks each from PARSEC and SPLASH suites.



(a) PARSEC.streamcluster



(b) SPLASH.radix

Fig. 9. Comparison of simulation and analytical results of combined RD profile when four threads share the L2 cache for the configuration given in Table 2 for the benchmarks PARSEC.streamcluster and SPLASH.radix.

can be normalized to the range of estimated values or to the mean of estimated values. We have normalized the RMSE to the range of estimated values. It is defined as

$$NRMSE = \frac{RMSE}{X_{est,max} - X_{est,min}},$$

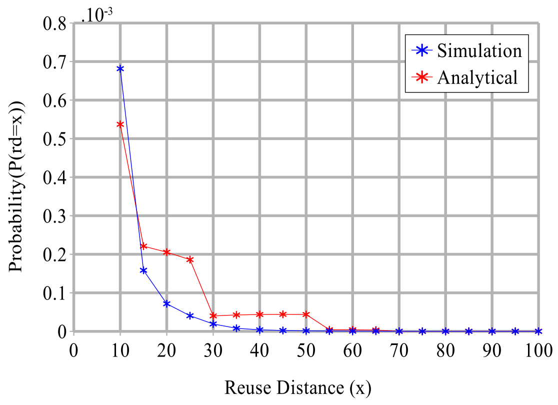


Fig. 10. Simulation Versus analytical results for CRD profile when 8 threads share L2 cache for configuration given in Table 2 for Splash2. fmm.

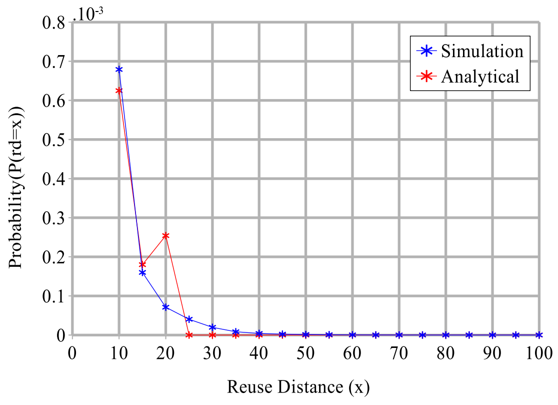


Fig. 11. Simulation Versus analytical results for CRD profile when 10 threads share L2 cache for configuration given in Table 2 for Splash2. fmm.

where $X_{est,max}$ and $X_{est,min}$ are the maximum and minimum value of estimated values respectively.

Table 2 and 3 gives the RMSE and the NRMSE values as a percentage for the values obtained through our analytical model with respect to the values obtained through simulation for concurrent, combined and coherent RD profile. As can be seen from the plots and the RMSE values, our model computes the RD profile to a reasonably good accuracy for most of the benchmarks.

8.7 Discussion on the Results

For most of the benchmarks, our model could predict the CRD profile well. Among the benchmarks of SPLASH-2, the error in CRD prediction for the benchmark barnes is high. Similarly among the benchmarks of PARSEC, the benchmark x264 couldn't follow the CRD profile well as shown in Fig. 7. The reason is mainly due to the large number of shared accesses in these benchmarks. The benchmark Barnes has 53.9 percent shared accesses.[4] Similarly, the benchmark x264 has 16.93 percent shared accesses respectively. Fig. 7 also brings out the limitation of our model.

In Table 4, we make quantitative comparison of the prediction accuracy of the model proposed in [24] with that of our proposed model. The prediction accuracy for the three benchmarks of Jiang et al.'s model has been reported in [24].

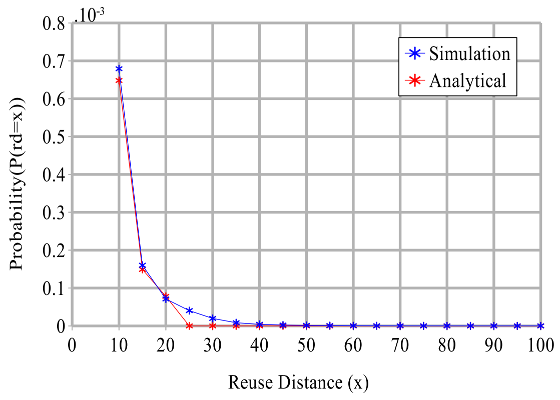


Fig. 12. Simulation Versus analytical results for CRD profile when 12 threads share L2 cache for configuration given in Table 2 for Splash2. fmm.

For our proposed model we have taken the prediction accuracy to be (1 - NRMSE), where NRMSE is as given in Table 2. It can be seen that the prediction accuracy of our model are better.

8.8 Cache Miss Rate Calculation

In this section, as an example of the use of CRD profile, we show the cache miss rate calculation of shared L2 cache with LRU replacement policy using the CRD profile computed with our analytical model. Cache miss rate calculation from RD profile have been explained in many of the previous works [Sen and Wood 2013]. From the RD profile, cache miss rate can be calculated as follows. For a cache with associativity A and number of sets N, those references with reuse distance less than A will surely result in a hit. For an LRU cache, the references with reuse distance greater than or equal to A will be a miss if more than or equal to A references get mapped to the same set. The references with an infinite reuse distance at any level are misses at that level. Assuming a block can go into any sets with equal probability, for a reference with reuse distance d, $d \geq A$, the probability that A or more blocks being mapped to the same set is

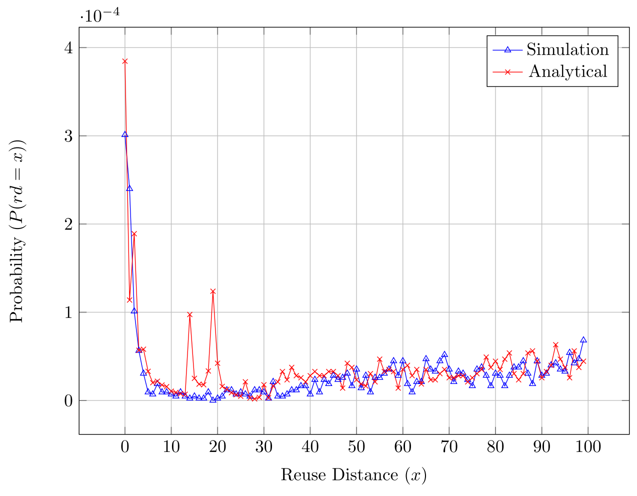
$$P(\text{A or more blocks map to the same set}) = \sum_{i=A}^{d} \binom{d}{i} (\frac{1}{N})^i (1 - \frac{1}{N})^{d-i}. \tag{25}$$

Therefore,

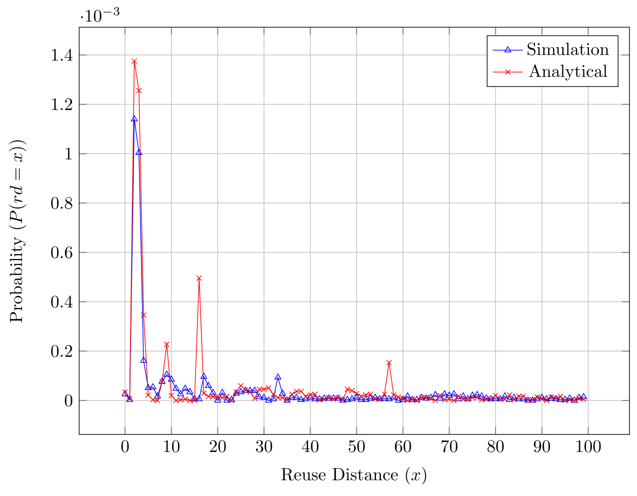
$$\text{Miss rate} = P(rd = \infty) + \sum_{j=A}^{rd_{max}} P(rd = j) \sum_{i=A}^{j} \binom{d}{i} (\frac{1}{N})^i (1 - \frac{1}{N})^{d-i}. \tag{26}$$

Using the above formula, we calculated the miss rate of 2 MB shared L2 cache with an associativity of eight. Since miss rate calculation is not the primary objective of our paper, we limit the calculation to just one cache configuration. For cache miss rate calculation, we use the combined RD profile of the applications. The Fig. 14 shows the miss rate calculation for eight benchmarks of the SPLASH and five benchmarks of the PARSEC benchmark suite respectively. The blue bar represents the actual miss rate given by

4. It includes all accesses. Not just distinct blocks.



(a) PARSEC.dedup



(b) PARSEC.bodytrack

Fig. 13. Comparison of simulation and analytical results of coherent RD profile for the benchmarks dedup and bodytrack of the PARSEC benchmark suite when 4 threads run with private L2 cache each.

TABLE 2
Average Prediction Errors for Concurrent RD Profile Derivation

Benchmark	RMSE	NRMSE (%)	Benchmark	RMSE	NRMSE (%)
PARSEC			SPLASH		
Blackscholes	0.007665	8.28	radiosity	0.000832	5.12
Vips	0.0000414	12.17	cholesky	0.0006924	3.89
Dedup	0.0000336	11.14	lu	0.000243	4.59
Fluidanimate	0.00319	6.82	fft	0.0024437	8.12
Streamcluster	0.000402	5.95	fmm	0.0001285	3.87
Canneal	0.0000686	24.11	radix	0.0000232	24.67
swapation	0.00007565	23.38	raytrace	0.0000906	6.78
			water.nsquared	0.0000907	4.88

the simulator. The red bar gives the miss rate calculated using (26), wherein the values of $P(rd = \infty)$, and $P(rd = j)$ are taken from the Combined RD profile generated by the simulator. Note that there is slight difference between the values of the miss rate calculated using two different approaches based on the simulator's output. The yellow bar are the miss rate calculated using (26), using the combined RD profile computed analytically as given by Section 8.4.2.

8.9 Speedup: Simulation Versus Proposed Model

During cache design, one may evaluate the performance of the cache for different cache configurations by varying its parameters such as size, associativity, or policies for various benchmarks. One of the advantages of the proposed analytical model against the simulation schemes is that it can reduce the time taken to carry out cache performance evaluation for quicker design space exploration. In this subsection we substantiate this claim.

We considered five different benchmarks namely Blackscholes, Canneal, Dedup, Fluidanimate and StramCluster. For each benchmark we carried out simulation for six different configurations by varying the L2 cache size as (512 KB, 1024 KB, and 2048 KB) and L2 associativity to be 4 or 8. The SNIPER simulator and the python code for numerical computation were run in Intel Core i7 system having 4 GB RAM. The number of threads is fixed as four, and the L1 parameters are fixed as given by Table 1 for this experiment. The total time taken to obtain the L2 CRD profiles for six different configurations for each benchmark is found and plotted in Fig. 15. To compute the time taken by our model

TABLE 3
Average Prediction Errors for Combined RD Profile and Coherent RD Profile

Profile	Benchmark	RMSE	NRMSE(%)
Combined RD	Streamcluster	0.000261	5.62
	radix	0.000365	8.025
Coherent RD	Bodytrack	0.0000691	5.02
	Dedup	0.0000275	7.15

TABLE 4
Comparison of Prediction Accuracy of our Proposed Model with that of Jiang et al.'s Model [24] for three Benchmarks of PARSEC Benchmark Suite

	Prediction accuracy	
Benchmark	Model [24]	Our model
Vips	76%	87.83%
Stream Cluster	72%	94.05%
Swapation	74 %	76.62%

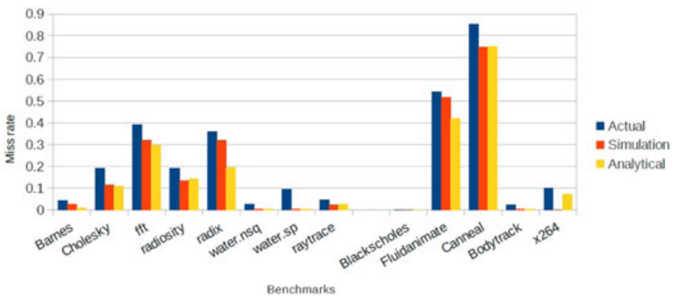


Fig. 14. Miss rate calculation of L2 cache when 4 threads share the cache using the combined RD profile obtained through simulation and the analytical model.

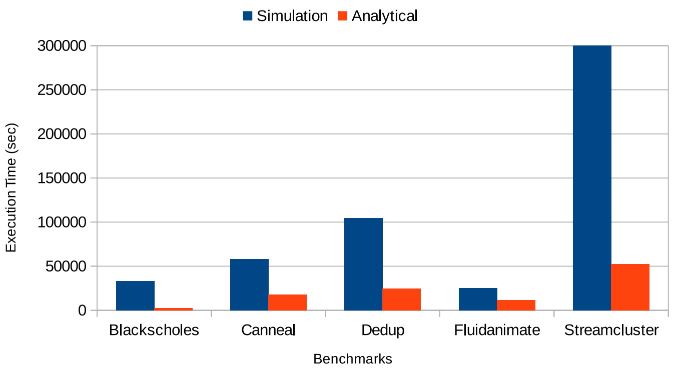


Fig. 15. Comparison of time taken to get CRD profile by simulation against analytical model for six different cache configurations for five benchmarks.

we did as follows.The standalone RD profiles of the threads and other relevant inputs to our model are obtained through simulation once and the corresponding time taken is noted as T1. The total time taken to numerically evaluate the CRD profiles based on our analytical formulae for all six configurations for a given benchmark is found out and noted as T2. The total time taken by the theory (T1 + T2) for the benchmarks are plotted in Fig. 15. It can be seen that the speedup offered by theory varies from 2 to almost 6. The speedup will increase if number of configuration considered increases. The advantage of the speedup will be felt if the number of benchmarks considered is also large.

9 Conclusion and Future Work

In this paper we derived the concurrent reuse distance profile of a thread of multi-threaded applications running in a shared cache environment with other threads in a chip multi-processor by a novel analytical model. We used a Markov chain model along with combinatorics to arrive at our objective. The analytical results were validated with the benchmarks of the PARSEC and the SPLASH benchmark suite using the multi-core simulator Sniper. We then derived the combined RD profile of all the threads sharing the cache from the CRD profile of individual threads. When threads run with private cache, we derive the coherent RD profile of the threads considering the effect of coherency misses.

Our model can be used as tool for CRD measurement in a faster way and to get insights about the behavior of threads in shared cache environment without doing lengthy simulation. The predicted CRD profile can also be used for performance analysis of caches and programs, locality predictions, cache management policies and compiler optimization. As an example use of the CRD profile, we have given the miss rate calculation from the predicted CRD profile.

Though the RD profile is independent of specific cache configuration, when we want to analyze the performance of second or third level cache, we would have to collect the RD profile with respect to that level. The RD profile of the second and the third level cache depends on the configuration of the previous level cache. Computation of the RD profile with respect to a particular cache level, from the RD profile of the application is an important direction for future work. Another aspect still lacking in analytical models, especially in reuse distance calculation, is modeling the spatial locality. Reuse distance is a temporal locality metric. Inclusion of the spatial locality effect in the modeling is a necessity, since the performance of programs and caches depends not only on temporal locality but also on spatial locality.

Acknowledgments

We acknowledge Mr. Vijaya Bhasker and Mrs. Aparna Behara of the EE Deptt., IIT Madras for the help offered by them in preparing the manuscript.

References

[1] L. Eeckhout, *Computer Architecture Performance Evaluation Methods*. San Rafael, CA, USA: Morgan & Claypool, 2010. [Online]. Available: https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6813138
[2] R. Ubal, J. Sahuquillo, S. Petit, and P. López, “Multi2sim: A simulation framework to evaluate multicore-multithread processors,” in *Proc. IEEE 19th Int. Symp. Comput. Archit. High Perform. Comput.*, 2007, pp. 62–68.
[3] A. Patel, F. Afram, S. Chen, and K. Ghose, “MARSS: A full system simulator for multicore x86 CPUs,” in *Proc. 48th Des. Autom. Conf.*, 2011, pp. 1050–1055.
[4] X. Chen and T. Aamodt, “Modeling cache contention and throughput of multiprogrammed manycore processors,” *IEEE Trans. Comput.*, vol. 61, no. 7, pp. 913–927, Jul. 2012.
[5] C. Xu, X. Chen, R. Dick, and Z. Mao, “Cache contention and application performance prediction for multi-core systems,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2010, pp. 76–86.
[6] S. Eyerman and L. Eeckhout, “System-level performance metrics for multiprogram workloads,” *IEEE Micro*, vol. 28, no. 3, pp. 42–53, May 2008.
[7] J. J. Yi, L. Eeckhout, D. J. Lilja, B. Calder, L. K. John, and J. E. Smith, “The future of simulation: A field of dreams,” *Comput.*, vol. 39, no. 11, pp. 22–29, 2006.
[8] B. Sinharoy, R. Kalla, W. Starke, H. Le, R. Cargnoni, J. Van Norstrand, B. Ronchetti, J. Stuecheli, J. Leenstra, G. Guthrie, et al., “IBM POWER7 multicore server processor,” *IBM J. Res. Develop.*, vol. 55, no. 3, pp. 1–1, 2011.
[9] I. Intel, “and IA-32 architectures software developers manual,” *Volume 3A: System Programming Guide, Part*, vol. 1, no. 64, p. 64, 64. [Online]. Available: https://software.intel.com/en-us/articles/intel-sdm
[10] V. Selfa, J. Sahuquillo, S. Petit, and M. E. Gmez, “A hardware approach to fairly balance the inter-thread interference in shared caches,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3021–3032, Nov. 2017.
[11] A. Agarwal, J. Hennessy, and M. Horowitz, “An analytical cache model,” *ACM Trans. Comput. Syst.*, vol. 7, no. 2, pp. 184–215, 1989.
[12] D. Thiebaut and H. S. Stone, “Footprints in the cache,” *Perform. Model. Comput. Architects*, vol. 11, 1996, Art. no. 146.
[13] G. Suh, D. Srinivas, and L. Rudolph, “Analytical cache models with applications to cache partitioning,” in *Proc. 15th Int. Conf. Supercomputing*, 2001, vol. 2001, pp. 1–12.
[14] D. Chandra, F. Guo, S. Kim, and Y. Solihin, “Predicting inter-thread cache contention on a chip multi-processor architecture,” in *Proc. 11th Int. Symp. High-Perform. Comput. Archit.*, 2005, pp. 340–351.

[15] F. Liu, F. Guo, Y. Solihin, S. Kim, and A. Eker, “Characterizing and modeling the behavior of context switch misses,” in *Proc. 17th Int. Conf. Parallel Architectures Compilation Techniques*, 2008, pp. 91–101.
[16] R. Sen and D. A. Wood, “Reuse-based online models for caches,” in *Proc. ACM SIGMETRICS/Int. Conf. Meas. Modeling Comput. Syst.*, 2013, pp. 279–292.
[17] C. Cascaval and D. A. Padua, “Estimating cache misses and locality using stack distances,” in *Proc. 17th Annu. Int. Conf. Supercomputing*, 2003, pp. 150–159.
[18] K. Beyls and E. D’Hollander, “Reuse distance as a metric for cache behavior,” in *Proc. IASTED Conf. Parallel Distrib. Comput. Syst.*, 2001, vol. 14, pp. 350–360.
[19] N. Duong, D. Zhao, T. Kim, R. Cammarota, M. Valero, and A. V. Veidenbaum, “Improving cache management policies using dynamic reuse distances,” in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2012, pp. 389–400.
[20] Y. Zhong, X. Shen, and C. Ding, “Program locality analysis using reuse distance,” *ACM Trans. Program. Languages Syst.*, vol. 31, no. 6, 2009, Art. no. 20.
[21] C. Ding and Y. Zhong, “Predicting whole-program locality through reuse distance analysis,” *ACM SIGPLAN Notices*, vol. 38, no. 5, 2003, pp. 245–257.
[22] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, “Evaluation techniques for storage hierarchies,” *IBM Syst. J.*, vol. 9, no. 2, pp. 78–117, 1970.
[23] C. Ding and Y. Zhong, “Reuse distance analysis,” Tech. Rep. UR-CS-TR-741, University of Rochester, Rochester, NY, Feb. 2001.
[24] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen, “Is reuse distance applicable to data locality analysis on chip multiprocessors,” in *Compiler Construction*. Berlin, Germany: Springer, 2010, pp. 264–282.
[25] D. L. Schuff, B. S. Parsons, and V. S. Pai, “Multicore-aware reuse distance analysis,” in *Proc. IEEE Int. Symp. Parallel Distrib. Process., Workshops Phd Forum*, 2010, pp. 1–8.
[26] D. L. Schuff, M. Kulkarni, and V. S. Pai, “Accelerating multicore reuse distance analysis with sampling and parallelization,” in *Proc. 19th Int. Conf. Parallel Architectures Compilation Techn.*, 2010, pp. 53–64.
[27] M.-J. Wu and D. Yeung, “Coherent profiles: Enabling efficient reuse distance analysis of multicore scaling for loop-based parallel programs,” in *Proc. Int. Conf. Parallel Architectures Compilation Techn.*, 2011, pp. 264–275.
[28] C. Ding, X. Xiang, B. Bao, H. Luo, Y.-W. Luo, and X.-L. Wang, “Performance metrics and models for shared cache,” *J. Comput. Sci. Technol.*, vol. 29, no. 4, pp. 692–712, Jul. 2014. [Online]. Available: http://dx.doi.org/10.1007/s11390-014-1460-7
[29] Y. Zhong, S. G. Dropsho, X. Shen, A. Studer, and C. Ding, “Miss rate prediction across program inputs and cache configurations,” *IEEE Trans. Comput.*, vol. 56, no. 3, pp. 328–343, Mar. 2007.
[30] C. Ding and T. Chilimbi, “A composable model for analyzing locality of multi-threaded programs,” Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-2009–107, 2009.
[31] X. Shi, F. Su, J.-K. Peir, Y. Xia, and Z. Yang, “Modeling and stack simulation of CMP cache capacity and accessibility,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 12, pp. 1752–1763, Dec. 2009.
[32] S. V. den Steen and L. Eeckhout, “Modeling superscalar processor memory-level parallelism,” *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 9–12, Jan. 2018.
[33] J. S. Madonna and T. G. Venkatesh, “Analytical miss rate calculation of L2 cache from the RD profile of L1 cache,” *IEEE Trans. Comput.*, vol. 67, no. 1, pp. 9–15, Jan. 2018.
[34] R. K. V. Maeda, Q. Cai, J. Xu, Z. Wang, and Z. Tian, “Fast and accurate exploration of multi-level caches using hierarchical reuse distance,” in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2017, pp. 145–156.
[35] G. Ceballos, E. Hagersten, and D. Black-Schaffer, “Formalizing data locality in task parallel applications,” in *Algorithms and Architectures for Parallel Processing*, J. Carretero, J. Garcia-Blas, and J. Gracia, Eds. Cham, Switzerland: Springer, 2016, pp. 43–61.
[36] X. Shen, J. Shaw, B. Meeker, and C. Ding, “Locality approximation using time,” *ACM SIGPLAN Notices*, vol. 42, no. 1, 2007, pp. 55–61.
[37] X. Xiang, C. Ding, H. Luo, and B. Bao, “HOTL: A higher order theory of locality,” *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 1, 2013, pp. 343–356.
[38] E. Berg and E. Hagersten, “Statcache: A probabilistic approach to efficient and accurate data locality analysis,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2004, pp. 20–27.
[39] E. Berg, H. Zeffer, and E. Hagersten, “A statistical multiprocessor cache model,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Mar. 2006, pp. 89–99.
[40] D. Eklov and E. Hagersten, “Statstack: Efficient modeling of LRU caches,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2010, pp. 55–65.
[41] D. Eklov, D. Black-Schaffer, and E. Hagersten, “StatCC: A statistical cache contention model,” in *Proc. 19th Int. Conf. Parallel Architectures Compilation Tech.*, 2010, pp. 551–552.
[42] S. Kaxiras and C. Young, “Coherence communication prediction in shared-memory multiprocessors,” in *Proc. 6th Int. Symp. High-Perform. Comput. Archit.*, Jan. 2000, pp. 156–167.
[43] S. V. den Steen, S. D. Pestel, M. Mechri, S. Eyerman, T. Carlson, D. Black-Schaffer, E. Hagersten, and L. Eeckhout, “Micro-architecture independent analytical processor performance and power modeling,” in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Mar. 2015, pp. 32–41.
[44] C. Bienia, “Benchmarking modern multiprocessors,” Ph.D. dissertation, Department of Computer Science, Princeton University, Princeton, NJ, Jan. 2011.
[45] T. E. Carlson, W. Heirman, and L. Eeckhout, “Sniper: exploring the level of abstraction for scalable and accurate parallel multicore simulation,” in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2011, Art. no. 52.
[46] H. Luo, P. Li, and C. Ding, “Thread data sharing in cache: Theory and measurement,” *SIGPLAN Not.*, vol. 52, no. 8, pp. 103–115, Jan. 2017. [Online]. Available: http://doi.acm.org/10.1145/3155284.3018759
[47] A. J. Smith, “On the effectiveness of set associative page mapping and its applications in main memory management,” in *Proc. Int. Conf. Softw. Eng.*, 1976, pp. 286–292.
[48] W. F. King, “Analysis of demand paging algorithms,” in *Proc. IFIP Congr.*, Aug. 1971, pp. 485–490.
[49] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The splash-2 programs: Characterization and methodological considerations,” *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 2, 1995, pp. 24–36.

Jasmine Madonna Sabarimuthu received the BE degree from the Madras Institute of Technology, Chennai, India, in 2010, and the graduate degree from IIT Madras, in 2015. Currently, Jasmine is in her third year of Ph.D. in Electrical and Computer Engineering at North Carolina State University, NC, USA. She then had a short stint as software engineer at NMSWorks Software Pvt Ltd, Chennai before joining MS (by research) at IIT Madras, Chennai in 2012. She works in the broad area of computer architecture.

T. G. Venkatesh received the BE degree in electronics and instrumentation engineering from Annamalai University, India, in 1986, the ME degree in applied electronics from Bharathiar University, India, in 1988, and the PhD degree from the Indian Institute of Science, Bangalore, in 1993. While working as a scientific officer. For a brief duration he served at the Centre for Development of Telematics, and Indian Space Research Organization, Bangalore. He was a faculty at the Indian Institute of Technology, Delhi from 1994 to 1999. Presently he is a faculty at the Electrical Engineering Department of the Indian Institute of Technology, Madras. His areas of research interest include Stochastic Modeling, Computer Networks and Computer Architecture. At present his research group focuses on design and performance evaluation of medium access layer protocol in wireless network, and multicore architecture. He has authored a book on Developing multimedia applications with the Java Media Framework” and co-authored a book on “Computer Systems Design and architecture”, published by Pearson.