2017 IEEE International Symposium on High Performance Computer Architecture

Fast and Accurate Exploration of Multi-Level Caches Using Hierarchical Reuse Distance

Rafael K. V. Maeda[1], Qiong Cai[2], Jiang Xu[1], Zhe Wang[1], and Zhongyuan Tian[1]
[1]Hong Kong University of Science and Technology, [2]HP Labs
rkvivasmaeda@ust.hk, jiang.xu@ust.hk, qiong.cai@hpe.com

***Abstract*—Exploring the design space of the memory hierarchy requires the use of effective methodologies, tools, and models to evaluate different parameter values. Reuse distance is of one of the locality models used in the design exploration and permits analytical cache miss estimation, program characterization, and synthetic trace generation. Unfortunately, the reuse distance is limited to a single locality granularity. Hence, it is not a suitable model for caches with hybrid line sizes, such as sectored caches, an increasingly popular choice for large caches. In this work, we introduce a generalization to the reuse distance, which is able to capture locality seen at multiple granularities. We refer to it as Hierarchical Reuse Distance (*HRD*). The proposed model has same profiling and synthesis complexity as the traditional reuse distance, and our results show that *HRD* reduces the average miss rate error on sectored caches by more than three times. In addition, it has superior characteristics in exploring multi-level caches with conventional single line size. For instance, our method increases the accuracy on L2 and L3 by a factor of 4 and converges three orders of magnitude faster.**

***Keywords*-reuse distance; cache; simulation; statistical**

I. Introduction

The performance gap between processing units and the memory subsystem is of concern to many researchers. Current advances in die-stacked DRAM, as well as hybrid memory (HM) systems, have expanded the design space by providing higher memory bandwidth and allowing the use of large caches [1]–[3]. Finding the optimal cache parameters for potential next generation of memories is complex because it is not bound to hardware configurations alone, it also depends on the application characteristics. Therefore, optimizing the memory hierarchy requires the use of effective methodologies, tools, and models to assist the design exploration.

Reuse distance (*RD*), also referred as least recently used (LRU) stack distance, plays an important role in today's cache and memory system design [4]. It is a microarchitecture-independent model, and one of the dominant metrics in locality analysis. The *RD* is employed for several purposes, such as program characterization, compiler optimization, and analytical cache miss estimation [5]–[8]. Another application of the *RD* is to use it as a model for memory trace synthesis, a technique used to leverage statistical simulation, overcome proprietary code benchmarking, and allow creation of synthetic benchmarks [9], [10].

One of the limitations of the *RD* is that it confines the locality to a single block-granularity. Therefore, it loses locality information seen at any different block size. We demonstrate that if the *RD* is computed at a 64*B*-block granularity, it will lose the information about the reuse of pages, typically 4KB. This limitation makes the *RD* inappropriate to evaluate the performance of units that depend on multiple granularities. In fact, caches with hybrid line organizations, such as sectored caches [11]–[13], footprint caches [1], and unison caches [14], are examples of such units and are becoming increasingly popular as an effective way to implement large caches.

Furthermore, the scope of the *RD* model is limited to locality analysis. Hence, it does not include many useful characteristics for computer design and modeling. For instance, it lacks some detailed instruction-level information, such as timing, instruction mix and dependency, and request type (read or write; *RW*). While these limitations have already been addressed by existing works [10], [15], [16], the request type methods employed in most of them do not capture the real behavior of programs, leading to an inaccurate model for cache traffic.

In this work, we propose three improvements to the traditional reuse distance model. **First**, we generalize the *RD* by introducing the Hierarchical Reuse Distance (*HRD*). It captures locality at multiple granularities using single-profiling single-simulation run, and we show that the flat *RD* model is a particular case of the *HRD*. The *HRD* improves the miss rate predictions on caches with hybrid line size as well as on conventional designs. In addition, it allows reducing the simulation length by three orders of magnitude due to faster statistical convergence. **Second**, we introduce a microarchitecture-independent model for request type, *RW*, that leads to more accurate eviction traffic in the last level cache (LLC). The proposed *RW* mechanism can reduce the traffic error by around 100 times when combined with the *HRD*. Our *RW* method comes virtually for free, introducing little overhead during profiling and trace generation. **Third**, we propose the Bit Markov Chain (*BMC*), a method to incorporate higher order localities into the traditional flat *RD* by approximating the address distribution of the original program. The results show that *BMC* is able to enhance several statistical characteristics of the flat *RD*.

The rest of this paper is organized as follows. In Section II we introduce the conventional *RD* model. Section III details the proposed hierarchical method. The methodology is explained in Section IV, and the results are discussed in

Table I
SIMPLE EXAMPLE OF LOCALITY ANALYSIS.

Access	$a[0]$	$b[0]$	$c[0]$	i	$a[1]$	$b[1]$	$c[1]$	i
Address	0	2	4	6	1	3	5	6
Cache-line	0	1	2	3	0	1	2	3
$2B$-distance	∞	① ∞	∞	∞	3	3	③ 3	3
Page-addr	0	0	0	0	0	0	0	0
$16B$-distance	④ ∞	② 0	0	0	0	0	0	0

Items marked with ⊝ are examples of the *HRD* profiling explanation in Section III-A.

Table II
TRACE SYNTHESIS USING THE FLAT REUSE DISTANCE PROFILE.

$2B$-distance ①	∞	∞	∞	∞	3	3	3	3
Synthesized cache-line	② 32	0	24	8	③ 32	0	24	8
page-addr	4	0	3	1	4	0	3	1
$16B$-distance	∞	∞	∞	∞	3	3	3	3

Items marked with ⊝ are examples of the trace synthesis explanation using the flat reuse distance; see Section II-A.

Section V. Similar works are presented in Section VI, while Section VII concludes this work.

II. BACKGROUND

Reuse distance is a metric of locality present in the sequence of memory accesses, where distance is defined as the number of distinct elements between two references to the same location. The *RD* alone only incorporates temporal locality. To account for spatial locality, it has to be computed based on a block granularity of width W. The common choice for W is the size of a cache line, typically 64*B*, measuring in this case the reuse of cache lines [4], [6]. Table I illustrates an address trace obtained as a simple example. It shows the address for each memory reference, and the respective line and page address, assuming a $2B$ line size and $16B$ page size respectively. The table also depicts the *RD* of $2B$ and $16B$ blocks, where infinity (∞) is used to indicate the first access occurrence to the block.

A. Traditional flat reuse distance

An especial use of the *RD* is to generate a synthetic trace that has same locality profile as the original program. This has been shown useful to leverage statistical simulation, overcome proprietary code benchmarking, and allow creation of synthetic benchmarks. The traditional generation flow using the *RD* model uses the reuse profile obtained for single locality granularity. It receives as input the *RD* distribution and produces a trace that has a similar reuse profile. We explain the synthesis flow using the line reuse profile from the example in Table I. The final synthetic trace is shown in Table II and is explained as follows: ① we randomly generate a sequence of *RD*s following the profile obtained from the original trace, as at the top of Table II. For every *RD*, if infinite, ② a new random address is created, such as the first use of line address 32, and if it is a definite distance, R, ③ the corresponding R^{th} last distinct address is re-used, such as the second use of line address 32.

The middle row of Table II shows the synthesized line address. On one hand, we can observe that even though the absolute line address is not the same as the original program, the $2B$-distance follows the exact locality of the original trace. On the other hand, the locality seen at the page size, $16B$-distance, differs considerably, as shown at the bottom of Table II. This is one of the strongest limitations of the flat *RD*; it only preserves the locality at a single granularity, the one used to profile. In fact, at any other granularity, the reuse has the same profile as the one instrumented.

A considerable difference between the locality of cache lines (64*B* blocks) and locality of pages (4KB blocks) exists in real programs. The locality analysis of some of the SPEC CPU2006 benchmarks exemplify this situation, as shown in Fig. 1. We can note that the page *RD* differs from the cache line by one order of magnitude. Therefore, due to its limitation, the flat *RD* is not suitable to model functional units that depend on locality at different granularities. For instance, an *RD* profiled at 64*B* will lead to accurate cache miss rate estimations, but will not be suitable for units that depend on a locality seen at 4KB, such as translation lookaside buffers (TLB).

This is an already recognized and accepted limitation of the flat *RD*. A common way to overcome this is to realize several independent locality analyses, and several independent simulations of the system, one for each block granularity of interest [9], [10]. It is important to note that it works when evaluating two functional units independently but is not suitable to evaluate functional units that intrinsically depend on locality at multiple granularities. In fact, caches with hybrid line organizations, such as sectored caches [11], footprint caches [1], and unison caches [14], are examples of such units, and are becoming increasingly popular as an effective way to implement large caches, due to their capacity to compromise between tag storage and off-chip traffic.

III. HIERARCHICAL REUSE DISTANCE

In this work, we introduce a generalization to the traditional flat *RD* model, which is able to capture and reproduce locality seen at one or more block granularities. We refer to this method as Hierarchical Reuse Distance (*HRD*).

A. Hierarchical reuse distance

The inspiration to leverage a *HRD* model comes from the hierarchical construction of a computer's functional unit, and the way the program accesses blocks at different granularities. The *HRD* model is composed of two different stages, profiling and synthesis. We introduce these stages here and later we provide implementation details. We use examples for a two layer *HRD*, without loss of generality.

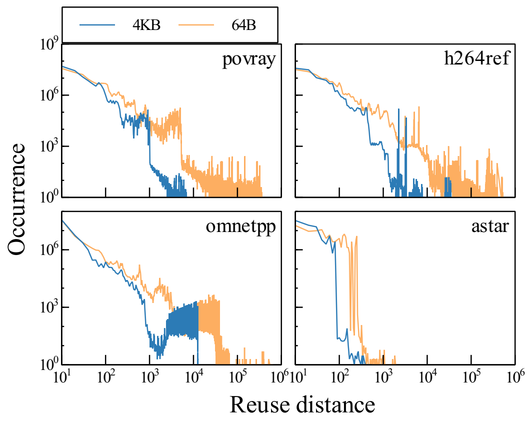

Figure 1. Reuse distance analysis of 64*B* blocks and 4KB blocks.

The notation used for a two-layer *HRD* with the first layer using a block granularity of 64*B* and the second layer using a block of 4KB is *HRD*$(64, 4K)$.

We illustrate the profiling and trace generation procedure of an *HRD*$(2, 16)$ for the example shown in Table I. The profiling is explained as follows. We start analysis at the layer with smaller block granularity, in this case $2B$, recording the block *RD* for each memory reference. Whenever an infinity *RD* is found, we record this infinity in the distribution of the layer, and also compute the *RD* for the next layer. For instance, in the middle part of Table I, for the ① second infinity *RD*, we will also ② compute the *RD* for the next layer. We stop computing the *RD* for a memory reference in two cases: when ③ the *RD* in a layer is not an infinity, or when ④ an infinity is in the last layer. This procedure repeats for all references. Note that only those references that cause an infinity *RD* in one layer can affect the counters in the next layer, such as the first four memory references in the previous example. The final entries that should be accounted in the reuse distribution of each layer in the previous example are underlined in blue (—) in Table I. Note that on every infinity *RD* obtained in one layer, the next layer will capture additional information, which is lost in the flat *RD*.

Address trace synthesis using the *HRD* follows a similar procedure, as shown in Table III. First ① ② we randomly generate a sequence of reuse distances, one for each layer, following the corresponding profile obtained from the original trace. We start the process by analyzing the *RD* at the layers with finer granularities, in this case $2B$. For ③ a non-infinity *RD* value, R, the generation is identical to the flat RD, i.e., re-use of the R^{th} last distinct $2B$ block address previously used. On the other hand, for an infinity distance, we consult which larger-block we should use by checking the *RD* in the next layer. Under this situation, two cases can happen. First, ④ there is a definite *RD* in the second layer, R. In this case, we re-use the corresponding R^{th} last distinct $16B$ block and generate a new $2B$ block address within it.

Table III
SYNTHESIS USING PROFILE INFORMATION FROM AN $HRD(2, 16)$.

2B-distance①	∞	∞	∞	∞	3	③3	3	3
16B-distance②	⑥∞	0	④0	0				
Synthesized cache-line	32	35	33	37	32	⑤35	33	37
Synthesized page-addr	4	4	4	4	4	4	4	4

Items marked with ● are examples of the trace synthesis explanation using the *HRD*; see SectionIII-A.

For instance, ⑤ the address 35 belongs to the reused page address 4. The second situation is ⑥ when there is an infinity in the first layer and also an infinity in the second layer. Since this is the last layer, we have to generate a completely new address on a new page. This situation happens one time in our example, in the first memory reference. As one can observe, the address generation is realized hierarchically, starting from finer granularities and going to coarser, hence the name of the method.

In Table III, we can perceive that the page reuse and the line reuse of the synthesized trace match the original program profile. In addition, other locality metrics are also preserved, such as the footprint. For instance, the original example trace touches a single $16B$ block, which is also preserved in the *HRD* synthesized trace but not in the flat.

B. *Profiling algorithm*

Algorithm 1 shows the detailed profiling procedure. It expects as input the address trace, T, the number of layers, N_l, and the block size for each layer stored in the vector, W. The output of the algorithm is the RD distribution for each layer, expressed as *hist*. The function REUSEDISTANCE should compute the number of distinct elements between two references to the same location. This computation depends on the data structure being used, such as the naive LRU stack implementation [17], or the traditional tree-based [18], [19]. Even though more sophisticated methods, such as approximate reuse distance [4], would work in our proposed model, we leave the evaluation of its accuracy for later studies. For brevity, we abstract its implementation with a data structure *Info* for each layer. Furthermore, we assume the existence of the function UPDATE, that updates the corresponding *Info* data structure for a block being referenced. For instance, in the LRU stack implementation, UPDATE would only bring the current block being referenced to the top of the stack.

The algorithm is detailed as follows. We iterate over every trace $t \in T$, as in line 4. We start at finer granularities, line 7, and compute the *RD* with respect to the corresponding block address, as shown in line 8. The symbol $\lfloor \cdot \rfloor$ represents the floor function. In line 10, we update the counters for the reuse distribution of layer l with the respective distance obtained, *dist*. The loop at line 7 repeats for as long as there is an infinity RD, or until the last layer is reached. The second step of the algorithm is to update the corresponding

Algorithm 1 Hierarchical reuse distance profiling.

```
function PROFILE_HRD($T$, $N_l$, $W$)
    $time \leftarrow 0$
    $hist \leftarrow 0$
    for each $t$ in $T$ do
        $dist \leftarrow \infty$
        $l \leftarrow 0$
        while $dist == \infty$ and $l < N_l$ do
            $block \leftarrow \lfloor t/W[l] \rfloor$
            $dist \leftarrow$ REUSEDISTANCE($Info[l]$, $block$, $time$)
            $hist[l][dist] \leftarrow hist[l][dist] + 1$
            $l \leftarrow l + 1$
        end while
        for $j$ from 0 to $N_l$ do
            $block \leftarrow \lfloor t/W[j] \rfloor$
            UPDATE($Info[j]$, $block$, $time$)
        end for
        $time \leftarrow time + 1$
    end for
    return $hist$
end function
```

data structures for all layers with the respective block address, as in lines 13 to 16.

A few observations are worth mentioning. First, we do **not** compute the *RD* for other layers when we find a non-infinity *RD*. The reason for doing so is that the reuse profile at finer granularities already incorporates the locality profile at coarser granularities, meaning that doing otherwise would incur redundant information in the next reuse distribution.

The second observation is with respect to the asymptotic complexity of time. The complexity of the functions REUSEDISTANCE and UPDATE depends on how many times they are accessed, N, as well as on the number of distinct elements, M. The traditional tree-based implementation has complexity $O(N \log M)$. For a flat RD model, N is the trace length, and M is the number of distinct cache lines. In a two-layer *HRD*$(64, 4K)$, the asymptotic complexity of the first layer is the same as the flat one. Since the second layer is only accessed for distinct elements in the first, M, and assuming P distinct blocks in the second layer, the complexity for the second layer is given by $O(M \log P)$. Therefore, the overall profiling complexity is $O(Layer_1) + O(Layer_2) = O(N \log M) + O(M \log P)$, which still bounded by the complexity of the first layer, $O(N \log M)$, because $P \leq M \leq N$. In summary, the *HRD* has the same asymptotic complexity as the flat RD, as long as the number of layers is kept small. Trace synthesis, explained in the next section, has similar analysis.

C. Trace synthesis

Algorithm 2 shows the address trace synthesis. It expects the number of layers, N_l, and a vector, W, with block sizes for each layer. The reuse distribution profile for each layer is passed in *hist*. The output is the synthetic address trace, *addr*. We assume the existence of a data structure *Info* able to keep an *LRU* history of distinct referenced addresses. The R^{th} last accessed block can be obtained with the function READHISTORY, and UPDATEHISTORY updates the corresponding history data structure. For instance, in a naive stack

Algorithm 2 Address trace synthesis using *HRD*.

```
function SYNTHESIS_HRD($N_l$, $W$, $hist$, $L$)
    for each $t$ in $L$ do
        $dist \leftarrow \infty$
        $l \leftarrow 0$
        $RND \leftarrow U(0, 1GB)$
        while $dist == \infty$ and $l < N_l$ do
            $dist \leftarrow$ RANDOMREUSE($hist[l]$)
            $l \leftarrow l + 1$
        end while
        if $dist == \infty$ then
            $addr[t] \leftarrow RND$
        else
            $addr[t] \leftarrow$ READHISTORY($Info[l-1]$, $dist$)
            $addr[t] \leftarrow$ NEW($addr[t]$, W[l-1])
        end if
        for $j$ from 0 to $N_l$ do
            $block \leftarrow \lfloor addr[t]/W[j] \rfloor$
            UPDATEHISTORY($Info[j]$, $block$)
        end for
    end for
    return $addr$
end function
```

implementation, UPDATEHISTORY brings the corresponding block to the top of the stack. We also assume the existence of a random number generator able to generate reuse distances following the distribution of the corresponding layer, which is expressed by the function RANDOMREUSE. Later we discuss some special cases of this function.

The synthesis algorithm is detailed as follows. We generate L memory references, as in line 2. The generation procedure starts from layers of finer granularities. The first step is to generate a random reuse distance for each layer until we find a definite distance or we reach the last layer, as in the loop in line 6. For an infinity in all layers, a new random address is generated, uniformly distributed in the address space of the program, which is assumed here to be $1GB$, as in line 11. On the other hand, for a definite distance in one of the layers, we reuse the corresponding block with distance *dist*, as in line 13, and generate a new address within this block, as in line 14. The function NEW should generate a unique address. The last step is to update the address history for all layers, as in lines 16 to 19.

The synthesis procedure requires generating random reuse distances that follow the distribution profile for each layer, expressed as the function RANDOMREUSE, line 7. In this work, we use the traditional method in which the RD histogram in its entirety is recorded, and the corresponding cumulative distribution function (*CDF*) is used to generate the random reuse number. Even though we are aware of other methods, studying the storage, performance of generation, and how accurate these models represent the actual *RD* distribution is beyond the scope of this work.

D. Address distribution

The infinity *RD* captures the cold misses of a program. During synthesis, we need to generate a new address for any infinity *RD* obtained. Most of the previous works generate a random number uniformly distributed in the interval

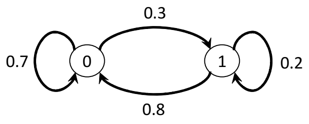

Figure 2. Markov chain of two states with their respective transition probabilities.

$U(0, 1GB)$, as we have shown in Algorithm 2, line 5. While this is effective in generating a unique reference, the address generated is completely independent of the actual address distribution of the program. The main concern is that the original address distribution holds some higher order localities, such as spatial locality and conflicting addresses within different blocks. We show that using the address distribution of the program, in combination with the traditional *RD* model, enhances several properties of the trace generated, such as the accuracy, correlation, and convergence speed.

In this work, we study a low-complexity method to model the address distribution of the original trace. Note, however, that a highly accurate address distribution model is undesired since it would over-represent the locality on top of the *RD*. Hence, we use a relaxed model by treating every address bit as independent of the others, even though we are aware about the high covariance between bits.

We refer to this method as Bit Markov Chain (*BMC*) and it is explained as follows. For every bit of the address width, we create a two-state Markov chain, as shown in Fig. 2. The state transition rate for the i^{th}-bit is given by the conditional probability $P_i(b_i|s_i)$, s_i is the previous bit state, and b_i is the bit value for the current memory reference, with $s_i, b_i \in \{0, 1\}$. This conditional probability is interpreted as the probability of the current bit being b_i given that the previous bit state was s_i. We obtain $P_i(b_i|s_i)$ during profiling, for all combinations of (b_i, s_i), by counting the occurrences of the transitions from s_i to b_i in the i^{th}-bit of the address trace. This count is normalized with respect to the total trace length. As an example, to compute $P_i(0|1)$ we have to count the number of transitions from 1 to 0 that happened in the i^{th}-bit of the address trace.

For synthesis, whenever an infinity *RD* is encountered, we generate a new address by aggregating all bit values obtained from each *BMC*. The i^{th} address bit is set as 1 with probability $P_i(1|s_i)$, and 0 otherwise, where s_i is the previous bit value. This address generation method replaces line 5 in Algorithm 2. The *BMC* is able to preserve the likelihood of a bit and its transition probability.

E. Request type

Another important characteristic from the memory trace is the request type of the memory access: read or write. Nearly all of the previous works on synthetic address generation record a simple proportion of the number of writes in the trace, ρ, and use this information to determine the synthetic memory request type, using binomial trials [9], [20], [21]. This method leads to an equal probability, ρ, of modifying any memory location accessed. Consequently, the number of dirty cache lines is overestimated, causing misleading eviction traffic.

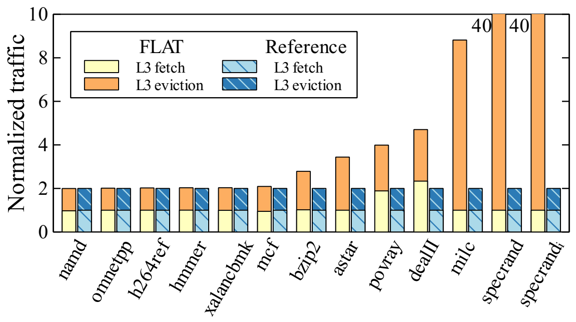

Figure 3. L3 cache traffic in bytes.

We have analyzed the cache traffic for a synthetic trace obtained with the traditional *RD* and binomial trials. Fig. 3 shows the traffic between L3 and memory, in terms of bytes, of the synthetic trace. As we can note, the binomial trials can lead to a remarkable error for the *LLC* if compared to the original program. While the fetching traffic is fairly accurate, the eviction traffic is inaccurate for many applications, reaching more than 9 times for *milc* and 40 times for *specrand*. In order to overcome this, we propose a microarchitecture-independent method obtained with simple modifications to the traditional *RD* algorithm. Our model is able to enhance eviction traffic of higher level caches.

The proposed method is inspired by the way programs access data. We assume that a memory location can be in three different states: new, when it has never been touched; $clean$ when it has been read but never written; and $dirty$, when the memory location has been modified. At the beginning, all states are set to new. When a read (or write) operation is performed in a new memory location, its state is set to $clean$ (or $dirty$). As expected, writing to a $clean$ block causes a state transition to $dirty$. Once the state reaches $dirty$, it will never come back to $clean$ again. Thus, writing or reading to a $dirty$ block will remain $dirty$.

During profiling, we count the occurrences of read (or write), accordingly to the current state of the memory location, and normalize it with respect to the total trace length. This record gives the respective conditional probability $P(RW|s)$, where s is the state, $s \in \{new, clean, dirty\}$ and *RW* is the type of operation being performed, $RW \in \{read, write\}$. During synthesis, we select the request type in accordance with the state of the generated location, s, being $read$ with probability $P(read|s)$, and $write$ otherwise.

Including the proposed enhanced *RW* model is virtually for free in both profiling and generation. The only modifications required in Algorithms 1 and 2 are (1) to include a state variable along with the variable holding the unique address, and (2) to update the counters and state as explained. Since the cost of computing the *RD* is dominated by hashing and searching [22], applying these changes will only have minimal performance degradation.

IV. Methodology

Our methodology consists of two distinct steps. First, we instrument the program to obtain the RD distribution, address distribution using *BMC*, and the request type. Second, we use this information to generate traces that follow same statistical behavior as the original program. We simulate both, the original program and the synthetic traces, and compare the results.

A. Profiling

We build three base models for evaluation. The first, *FLAT*, is the traditional synthesis approach using the flat RD with a cache line of 64-bytes and address generation uniformly distributed in the range $U(0, 1GB)$. The second is the traditional flat *RD* model using *BMC* to generate addresses, referred as *BMC* during discussion. The third is the hierarchical method with two layers, one with a 64*B* block and the second with 4KB block, noted as *HRD*. We choose these granularities because they are the two most common locality granularities in a computer system, the typical cache line size and page size. Note, however, that these values have to be chosen in accordance with the locality granularity of interest and depend on the system being evaluated. For performance reasons, we impose an upper limit on the number of distinct elements for *FLAT* and *HRD* of $128K$ entries.

The GEM5 simulator [23] is used for collecting the traces. Table IV shows the system configuration used. We use a single core ARM processor. Note that the processor microarchitecture, as well as memory alignment, can slightly affect the *RD* profile, but we expect our findings to hold true for other architectures. The simulation is run for $500M$, but we start collecting the traces after $100M$ instructions to avoid the cold start of programs. Traces are collected between the core and the $L1$ data cache.

Table IV
System details used for profiling.

Compiler/OS	arm-gcc v4.5.1 (-O2) / System-call emulation
CPU	ARM-v7, 2-way superscalar, In-order, 5-stages
Caches	I/D-cache (32KB, 4-assoc), L2 (512KB, 8-assoc) 64B-line, LRU replacement policy, Inclusive; write-back/write-allocate

B. Simulation

We use a trace-driven simulator, DineroIV [24], to compare the synthesized traces and the original trace. The baseline cache configuration matches with the configuration used for profiling, as shown in Table V. However, we sweep several parameters, such as cache size, associativity, and line size, to evaluate the prediction performance of the methods.

Table V
Baseline cache configurations (64B-line, LRU).

I/D-cache	L2	L3
32KB, 4-assoc	512KB, 8-assoc	2MB, 16-assoc

C. Evaluation

We use the SPEC CPU 2006 [25] benchmarks to evaluate our models. All benchmarks are used during our evaluations, excluding those in Fortran, due to compilation issues. For brevity, in some of the figures, we only show a representative set of the benchmarks. We evaluate the models based on the cache miss rate, and cache traffic, comparing the synthetic models with the original program trace. We report the absolute miss rate error, and to evaluate how closely the model follows the original trends, we use Pearson's correlation factor. Unless otherwise stated, we obtained a p-value less than 0.05 for all correlation factors computed.

V. Results

This section discusses the results obtained.

A. Locality analysis

Fig. 4 shows an example of the *RD* distribution computed for *povray* using 64*B* and 4KB granularity. As stated before, the traditional flat *RD* model can only capture locality at a single granularity. This is verified by the reuse distribution obtained for *FLAT*. Its 64*B*-reuse matches exactly with the original program, while the 4KB-reuse is misleading. In fact, the distribution for page reuse and for line reuse are the same for *FLAT*, as can be seen by the overlapping curves in the right-top chart, despite the significant difference between them in the original program. As one can observe, *BMC* has same reuse distribution as *FLAT*. This happens because *BMC* uses a relaxed address distribution, which slightly improve some of the statistical properties without considerable changes in the reuse distribution.

The bottom-right chart of Fig. 4 shows the *RD* distribution for the *HRD*. The results demonstrate that the hierarchical implementation is able to capture with more accuracy the 4KB-reuse distribution while preserving accuracy for 64*B*-reuse. We compute the area under the 4KB-reuse curve and use the relative difference between the models and

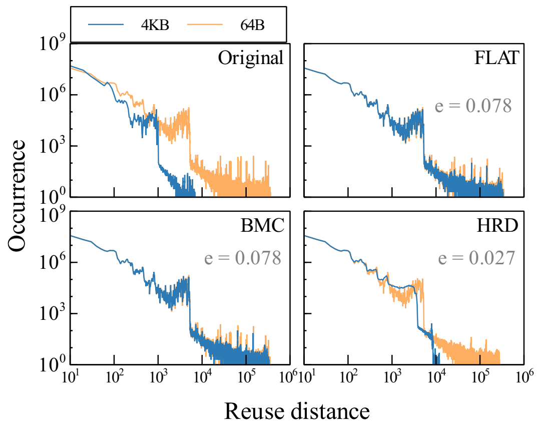

Figure 4. Reuse distance analysis. e is the percentual error for the 4KB-reuse distribution.

the original program as a metric of error, e. The *HRD* can reduce the error by around 3 times compared to flat implementations.

Another metric for locality is the footprint, a measure of how many blocks are touched by a program. In the same way as the *RD*, it has to be computed based on a block size. A typical 64*B* footprint allows for an estimation of whether the program data fits into the cache. On the other hand, 4KB is the common unit handled by memory management units and the 4KB footprint can influence physical page allocation, page swapping, and even policies for hybrid memory systems, such as migration from DRAM to *NVM* [2].

Fig. 5 reports the program footprint at granularities of 64*B* and 4KB. Once again, *FLAT* preserves the 64*B* footprint, but leads to substantial error for 4KB, while the *HRD* is accurate for both granularities. These results restate our arguments that flat *RD* model can only preserve locality at a single granularity, while the *HRD* can preserve for several. For *povray*, we can note that *HRD* is not able to match the footprint of the actual program. This happens when an upper layer block is reused more times than the number of lower layer blocks it can hold. This occurs sparsely and depends on the randomness of the *HRD*.

B. Baseline cache miss rate

The program locality directly affects the cache miss rate. Fig. 6 shows the cumulative miss rate using the baseline cache configuration. The first observation is that all *RD* models can provide close results for L1 cache. In addition, not much difference exists in the accuracy for *FLAT* and *HRD* for L1 cache since it is mostly affected by locality seen at a single block granularity, the cache line (64*B*).

To further assess each of the models, we compute the average absolute error and the correlation factor across all applications. The results are reported in the Fig. 7. The miss rates obtained for all applications present strong correlation with respect to the original program behavior

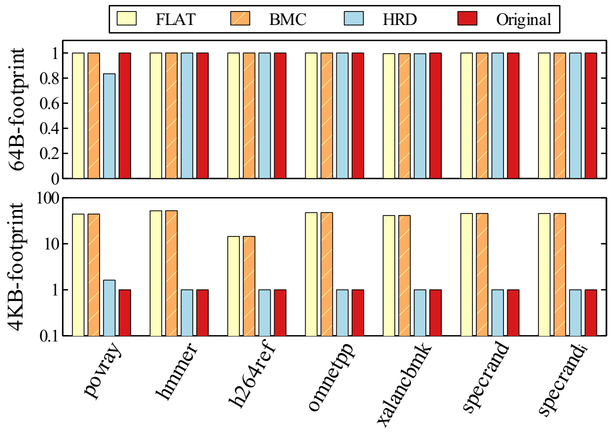

Figure 5. **Footprint** of applications using different block sizes.

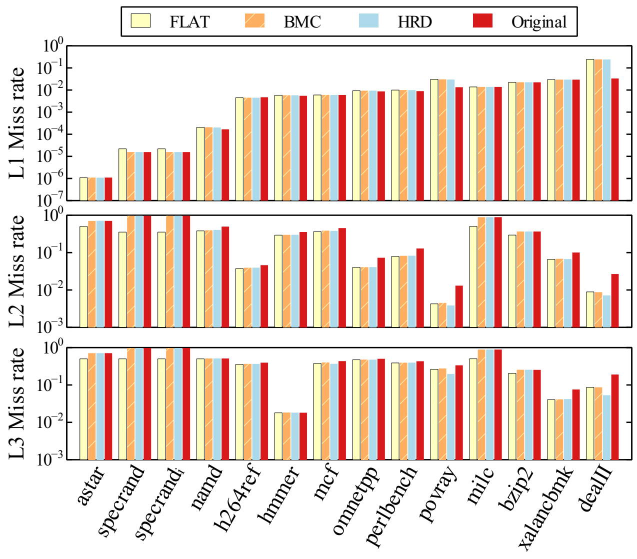

Figure 6. Cumulative cache miss rate for the baseline cache configuration.

for all levels of cache, as shown in the left-hand chart of Fig. 7. Therefore, all models preserve the trends, allowing their use in investigating which applications have higher miss rates under the same cache configuration. This is a key characteristic in design exploration and classification of applications. On the other hand, it can be noted that *FLAT* is substantially less accurate for L2 and L3 than the other models. This comes from the fact that the L1 cache filters the requests issued by the core which removes some locality of the references that reaches L2/L3. *HRD* and *BMC* include additional locality information that minimizes the filtering impact, becoming more accurate.

C. Sensitivity to cache parameters

To support design exploration, the models should be able to predict cache performance for various cache configurations. In order to assess this capability, we conduct experiments with several cache configurations. Around 50 cache configurations are studied by sweeping cache size, associativity, and line size. We investigate, L1 sizes from 2KB to 256KB, L2 from 128KB to 16MB, and L3 from 1MB to 32MB. The associativity is swept from 1 to 32, and also fully associative. The cache line size is changed from

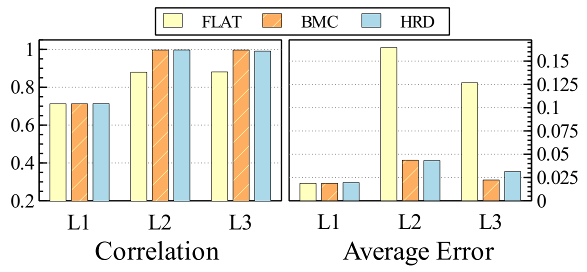

Figure 7. Average miss rate error and correlation factor across all applications, for the baseline cache configuration.

Table VI
AVERAGE MISS RATE ERROR (%) FOR CHANGES IN CACHE CONFIGURATIONS.

		baseline	L1 size	L1 assoc	L2 size	L2 assoc	L3 size	L3 assoc	line
L1	FLAT	1.9	1.3	1.7	1.9	1.9	1.9	1.9	18.8
	BMC	1.9	1.3	1.6	1.9	1.9	1.9	1.9	16.5
	HRD	2.0	1.2	1.7	2.0	2.0	2.0	2.0	11.8
L2	FLAT	16.5	13.1	13.7	18.4	18.5	18.7	18.7	34.7
	BMC	4.4	6.2	5.0	4.5	4.5	4.4	4.4	37.1
	HRD	4.3	6.4	5.0	4.7	4.2	4.3	4.3	17.5
L3	FLAT	12.7	12.7	12.7	17.8	16.4	17.5	17.1	49.5
	BMC	2.2	2.2	2.2	6.3	2.9	6.4	4.7	51.9
	HRD	3.1	3.1	3.1	3.7	3.2	5.3	4.2	18.2

32 to 512 bytes. Only the power of two numbers are used.

Table VI summarizes the average error computed over all benchmarks. Each row represents the average miss rate error in the respective cache level, while each column represents the parameter sweep in the mentioned domain. For instance, in the row *L2* × column *L1 size*, we report the average error in the *L2* cache when *L1 size* is swept from 2KB to 256KB.

We observe that while *FLAT* is accurate for the first level of cache, the accuracies for L2 and L3 caches are degraded, with average errors higher than 12%. One of the reasons, as explained before, is that the requests reaching L2 and L3 have weaker temporal locality and are more affected by other factors that are not captured by the flat *RD* model. The second reason for this is that the miss rate for L2 and L3 is usually higher than that for L1, leading to larger values of error.

BMC substantially increases the accuracy in higher levels of cache. Even though it has a relaxed distribution of the address, it adds higher order locality into the synthesized trace that does not exist in the trace obtained with a uniformly distributed address generation. This subtle locality makes *BMC* outperform the traditional flat *RD* for most of the parameters changes. The average errors obtained are usually below 6.5%, except for the cache line sweep. The extra locality that the *BMC* adds is not enough to improve the ability of *FLAT* in estimating the miss rate for changes in the cache line since it highly depends on the reuse distribution.

The *HRD* method is also able to reduce the average error on L2 and L3, obtaining similar performance as *BMC* for cache size and associativity. For cache line exploration, *HRD* outperforms the flat counterparts, with about 2 times lower error on L2 and 2.8 on L3. We also computed the correlation factor for all parameters being changed, and we found that *FLAT* and *BMC* have a weak correlation, in the interval $[-0.3, 0.3]$, for line sweeping, while *HRD* has moderate correlation, between $[0.5, 0.70]$. Therefore, due to its lower error for line exploration and moderate correlation, the hierarchical model allows early design exploration of cache lines, without having to carry out a new profiling. For associativity and cache size, all models obtained strong correlation, in the interval $[0.70, 1]$.

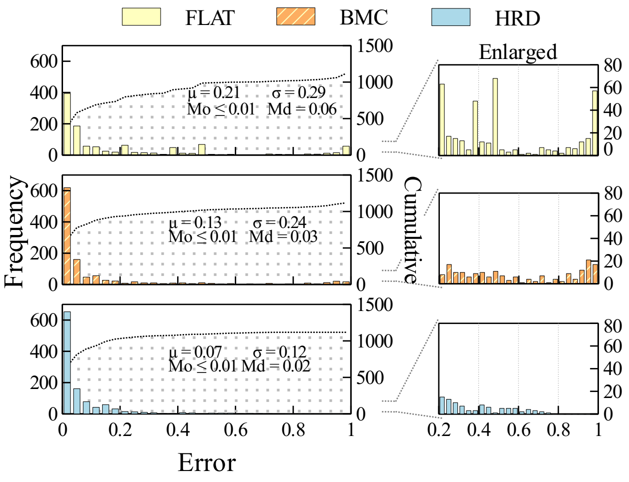

Figure 8. L3 miss rate error distribution.

The L3 miss rate error distribution for all applications and cache configurations is shown in Fig. 8. The cumulative distribution of the error population is also shown. Our results depict that the overall mean error for *FLAT* is high, about 21%, and median of 6%. Using the *BMC* address generator helps to reduce the error. In such case, the mean error is 13% and median is 3%. Despite this reduction, *BMC* still has a considerable concentration of errors above 60%. The hierarchical implementation improves the results even further, achieving mean error of 7%, and median of 2%. In addition, it almost eliminates any error above 60%.

In summary, *BMC* and *HRD* reduced the error in higher level caches by a factor of 4 during exploration of cache size and associativity. For line exploration, *HRD* is substantially superior than other models, with almost two times lower error and moderate correlation.

D. Sectored cache performance

In a conventional cache design, each cache line is associated with one tag. The tag array accounts for a considerable fraction of the total cache area. Using larger line is one of the approaches that can effectively reduce this meta-data, at the cost of moving larger amount of data on/off-chips. Several works try to achieve both, a reduction in the meta-data while keeping the transfer traffic low [1]. Sectored cache is one of the approaches that allows a trade-off of these two aspects. It uses a sector composed of several contiguous cache lines that share a single address tag. Transfers are realized on a line basis, typically 64*B*, which allows efficient use of the memory bandwidth [11]. The literature also refers to the cache line as sub-sector.

Two types of misses can happen in sectored caches. One is sector miss, which happens when there is an access to a sector and none of its lines is valid. The other is line miss, which happens when a line is invalid, but at least one of its siblings (lines that belong to the same sector) is valid. As

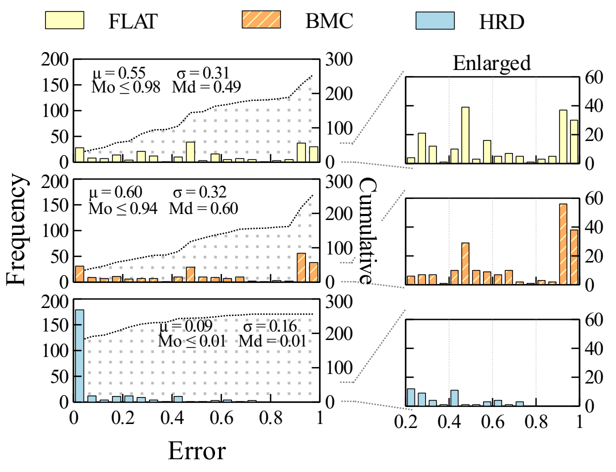

Figure 9. Line (64B) miss rate error distribution.

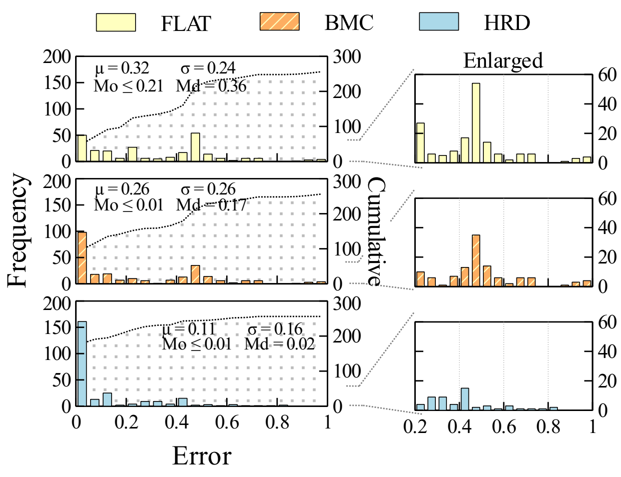

Figure 10. Sector (4KB) miss rate error distribution.

one can expect, the sectored cache performance depends on the locality seen at two distinct granularities, line and sector. We evaluate the performance of the models by simulating 24 configurations of a sectored L3 cache, with fixed sector size of 4KB and line size of 64B. Two normal-sized caches are simulated, 2MB and 8MB, and two bulky caches of 32MB and 128MB. We evaluated the caches using associativities of $\{1, 2, 4, 8, 16, 32\}$. L1 and L2 caches are kept with the same configuration as the baseline.

Fig. 9 shows the error distribution obtained for the line misses. *FLAT* has an average error of 55% and median of 49%. Furthermore, *BMC* does not enhance the accuracy of the conventional model. Observe that even though the flat *RD* was profiled using a block granularity that matches with the line size (64B) it performed poorly in predicting its miss rate. This comes from the fact that *FLAT* does not preserve the 4KB footprint and the reuse of sectors, which leads to a misleading number of sectors, and their lines, to be evicted. These results confirm that a synthetic trace with a single locality granularity is not suitable for functional units that depend on two, or more, locality granularities. On the other hand, *HRD* is substantially more accurate, obtaining half of the error population below 1% for being able to capture both locality granularities.

The sector miss error is reported in Fig. 10. The results show that *FLAT* is also inaccurate for sector miss rate estimations. In this case, *BMC* improves the sector miss prediction, reducing the mean error to 26%. However, *HRD*, once again, provides better results, with average error almost three times lower than the traditional flat *RD*, and median error as low as 2%.

In this section, we restricted the evaluations to sectored cache, but we expect the *HRD* to have superior modeling capabilities than the conventional flat *RD* for other cache designs with hybrid lines.

E. Cache traffic

The conventional method used to model read and write requests is not able to accurately reproduce the cache traffic in higher levels of cache. In this section, we evaluate our proposed *RW* mechanism and the traditional binomial trials using the baseline cache configuration. We use the suffix -*RW* to denote the models using our proposed method.

Table VII shows the relative traffic error of each of the models. The average error obtained for *FLAT*, using the binomial trials, is 36% and 593% for L2 fetch and eviction traffic respectively. The binomial trials method has an equal probability of making any memory location dirty, which does not follow the pattern that programs have in accessing memory locations. Since in write-back caches, dirty cache lines are the main contributors to eviction traffic, the binomial trials model leads to a huge error in the eviction traffic. Our proposed *RW* resembles that pattern and substantially reduces this error. This can be observed for *FLAT-RW*, in which the average eviction error is surprisingly reduced to 68% for L2. Combining *HRD* with *RW* leads to even better results, due to more accurate miss rate across all cache levels. For instance, the average fetch and eviction traffic error for *HRD-RW* is less than 4% for the LLC. Compared to the naive binomial trials method, our proposed *RW* method reduces the eviction traffic error by around 10 times in any of the locality models being used.

Note that fetch traffic is not affected by our *RW* since the

Table VII
AVERAGE CACHE TRAFFIC ERROR ($\%$). Bin IS THE TRADITIONAL BINOMIAL TRIALS.

		FLAT		**BMC**		**HRD**	
		Bin	*RW*	Bin	*RW*	Bin	*RW*
L2	Fetch	36.3	37.0	36.8	36.4	19.1	19.6
	Eviction	593.0	68.4	582.1	69.9	576.1	53.0
L3	Fetch	16.4	16.4	16.4	16.5	4.0	4.3
	Eviction	537.9	16.1	571.4	17.1	565.2	3.2

line miss itself is the trigger for fetching. Furthermore, more sophisticated methods can potentially be more accurate, at the cost of more complex implementation, slowdown during profiling/synthesis and microarchitecture dependency.

F. Convergence

A key characteristic of statistical simulation is that it quickly converges to the average performance behavior of the original application. This allows reduction of the simulation length, enabling quick design space exploration. We evaluate the convergence of the models, by measuring the correlation factor and average absolute error for different simulation lengths. We simulate the memory hierarchy using a fraction of the synthesized trace and compare the final cache miss rate with the original full trace length.

Fig. 11(a) shows the correlation factor and average error computed across all benchmarks. The results demonstrate that, for any of the models, the L1 miss rate achieves strong correlation even with a tiny fraction of the synthetic trace length, such as 10^3 memory references. With this simulation length, the average miss rate error is already low, about 7% for *FLAT* implementations and 5% for hierarchical. As the simulation length increases, the average absolute error can be reduced to less than 2%. For L1 cache, all of the models behave similarly. However, *FLAT* requires simulation lengths of more than 10^6 and 10^7 to reach strong correlation for L2 and L3 respectively. Furthermore, *FLAT* does not reach error below 10% for higher level caches.

The use of additional locality information in the flat *RD* can improve its statistical behavior. For instance, *HRD* and *BMC* have strong correlation for L2 and L3 with short simulation length, 10^3. In addition to this, they have lower error for any of the simulation lengths, which reaches below 10% around 10^6 and 10^7 for L2 and L3 respectively.

To compare each of the models, we define the convergence score as the sum $\sum correlation/error$ for all caches. This is justified by the fact that while high correlation desired, low average error is also important during design exploration. The higher the score the better combination of correlation and accuracy. We plot the score for each of the simulation lengths in Fig. 11(b). As we can note, *FLAT* obtains the worst convergence score among all models, due to its high error in L2 and L3. The peak convergence score for *FLAT* is about 54 for simulation length of 10^7. *HRD* outperforms this value with much a shorter simulation, 10^4, indicating more than 3 orders of magnitude faster convergence.

BMC can also provide substantial convergence improvements to the traditional RD, as observed in the results. However, the *HRD* is still superior, converging around 10 times faster. Moreover, it is worth recalling that *HRD* is also more accurate for simulation of caches with hybrid lines.

G. Hybrid reuse distance model

Throughout this paper, we reported *BMC* combined with the traditional flat *RD*. However, we have also investigated a hybrid approach, in which the hierarchical method and *BMC* are used together, but no significant improvements are obtained for most of the evaluation metrics. In fact, many of the results displayed *HRD* plus *BMC* slightly worse than *HRD* alone. Our initial observations suggest that the reason for this comes from an overexpression of the locality when using the two models together.

VI. Related work

To the best of our knowledge, this is the first attempt to study a hierarchical reuse distance model for locality analysis and trace synthesis.

Generating a synthetic trace has been proved useful as a way to compress the memory trace [26], [27], overcome proprietary benchmark [9], [10], [28], and explore statistical simulation [16], [29], [30]. However, replicating the original program behavior with accuracy comes at the cost of complex data structures of profiling and generation [9]. When targeting a specific class of applications, such as scientific programs, it is possible to leverage static program analysis in combination with run-time instrumentation to reduce complexity and improve accuracy [31], [32]. Locality is an important characteristic of programs and the previously mentioned works use some locality model. Our work concentrates on generalizing the traditional *RD* model while preserving its simplicity. Hence, these works can still benefit from our hierarchical *RD* model. In addition, it can be further enhanced by customizing it to specific needs of the architecture, such as for GPU [33] or multi-core systems [34], [35]

The typical method to obtain new addresses in trace synthesis is to use a uniform random number. In our work, we show that several characteristics of the *RD* can be enhanced by using a simple approximation of the actual program address distribution, inspired by the fact that the address space and its transition probability hold key locality information [36]. Regarding the request type of the memory reference, many previous works have used the binomial trials [9], [15], [21]. However, this method is not able to capture the actual pattern of programs. Balakrishnan identified this issue and proposed a per cache set write fraction approach [10]. Our method, on the other hand, consists of a microarchitecture-independent model that can increase substantially the accuracy of eviction traffic compared to the binomial trials.

The *RD* is not the only model used for locality analysis. A few other locality models have also demonstrated similar capabilities in estimating cache miss rate. Such as models for policies other than LRU [37] and locality approximation using time instead of distance [38]. Other similar locality methods and how they relate to each other have also been previously investigated [39], [40]. Furthermore, many other approaches can complement the design exploration of

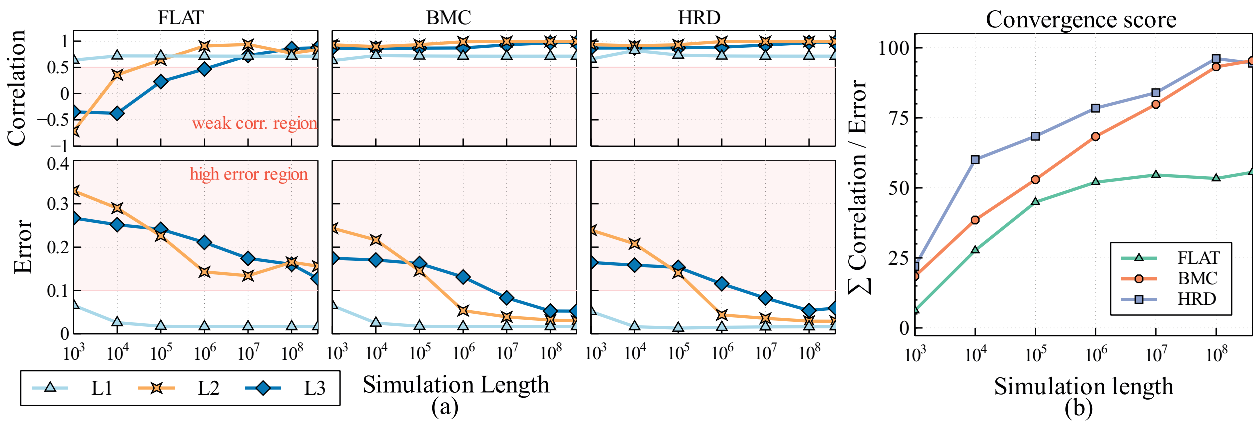

Figure 11. Convergence analysis. (a) L1, L2 and L3 correlation and average error for each of the models. (b) Convergence score defined as $\sum correlation/error$ for all cache levels.

caches, such as trace-driven simulation, analytical/statistical cache models, and miss rate curve prediction [8], [41], [42].

VII. Conclusions

In this work, we have introduced the hierarchical reuse distance model. It allows analysis and trace synthesis preserving multiple locality granularities. We portrayed several situations in which *HRD* can improve the traditional reuse distance model. For instance, *HRD* is more accurate for higher level caches, and presents faster statistical convergence, allowing reduction of the simulation length by more than three orders of magnitude. Furthermore, we found *HRD* more accurate to model cache organizations that depend on hybrid cache lines, such as sectored cache, an increasingly popular design choice for large caches. Despite its superiority, *HRD* has the same asymptotic complexity as the traditional reuse distance analysis and requires only simple modifications to the traditional algorithm. To the best of our knowledge, this is the first in-depth study of a multi-granularity locality analysis and trace synthesis. Two other improvements in the synthetic trace generation have also been studied. One is a relaxed address distribution model, the *BMC*, that also increases the accuracy and statistical convergence of the *RD* for L2 and L3 cache. The second is a microarchitecture-independent model for request type that reduces the eviction traffic error in the last-level cache.

Acknowledgment

We would like to thank the anonymous reviewers for their comments and suggestions. This work is partially supported by GRF16235616, IRS17EG48, and FSGRF15EG04.

References

[1] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked DRAM Caches for Servers: Hit Ratio, Latency, or Bandwidth? Have It All with Footprint Cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, (New York, NY, USA), pp. 404–415, ACM, 2013.

[2] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE*, pp. 664–669, July 2009.

[3] O. Mutlu, "Memory scaling: A systems architecture perspective," in *2013 5th IEEE International Memory Workshop*, pp. 21–25, IEEE, 2013.

[4] Y. Zhong, X. Shen, and C. Ding, "Program Locality Analysis Using Reuse Distance," *ACM Trans. Program. Lang. Syst.*, vol. 31, pp. 20:1–20:39, Aug 2009.

[5] Y. Zhong, M. Orlovich, X. Shen, and C. Ding, "Array Regrouping and Structure Splitting Using Whole-program Reference Affinity," in *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, PLDI '04, (New York, NY, USA), pp. 255–266, ACM, 2004.

[6] K. Beyls and E. H. D'Hollander, "Reuse distance as a metric for cache behavior," in *In Proceedings of the IASTED Conference on Parallel and Distributed Computing and Systems*, pp. 617–662, 2001.

[7] C. CaBcaval and D. A. Padua, "Estimating cache misses and locality using stack distances," in *Proceedings of the 17th annual international conference on Supercomputing*, pp. 150–159, ACM, 2003.

[8] E. Berg and E. Hagersten, "StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis," in *Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '04, (Washington, DC, USA), pp. 20–27, IEEE Computer Society, 2004.

[9] A. Awad and Y. Solihin, "STM: Cloning the spatial and temporal memory access behavior," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 237–247, Feb 2014.

[10] G. Balakrishnan and Y. Solihin, "WEST: Cloning data cache behavior using Stochastic Traces," in *IEEE International Symposium on High-Performance Comp Architecture*, pp. 1–12, Feb 2012.

[11] A. Seznec, "Decoupled sectored caches: conciliating low tag implementation cost and low miss ratio," in *Computer Architecture, 1994., Proceedings the 21st Annual International Symposium on*, pp. 384–393, Apr 1994.

[12] J. B. Rothman and A. J. Smith, "Sector cache design and performance," in *Modeling, Analysis and Simulation of Com-*

puter and Telecommunication Systems, 2000. Proceedings. 8th International Symposium on, pp. 124–133, 2000.

[13] B. Cuesta, Q. Cai, N. Hyuseinova, S. Ozdemir, M. Nicolaides, and F. Zyulkyarov, "Sectored cache with hybrid line granularity," July 3 2014. US Patent App. 13/729,523.

[14] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-47, (Washington, DC, USA), pp. 25–37, IEEE Computer Society, 2014.

[15] K. Ganesan, J. Jo, and L. K. John, "Synthesizing memory-level parallelism aware miniature clones for SPEC CPU2006 and ImplantBench workloads," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, pp. 33–44, March 2010.

[16] L. Eeckhout, K. de Bosschere, and H. Neefs, "Performance analysis through synthetic trace generation," in *Performance Analysis of Systems and Software, 2000. ISPASS. 2000 IEEE International Symposium on*, pp. 1–6, 2000.

[17] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems Journal*, vol. 9, no. 2, pp. 78–117, 1970.

[18] F. Olken, "Efficient methods for calculating the success function of fixed-space replacement policies," tech. rep., Lawrence Berkeley Lab., CA (USA), 1981.

[19] G. Almási, C. Caşcaval, and D. A. Padua, "Calculating Stack Distances Efficiently," in *Proceedings of the 2002 Workshop on Memory System Performance*, MSP '02, (New York, NY, USA), pp. 37–43, ACM, 2002.

[20] A. Joshi, L. Eeckhout, R. H. Bell, and L. John, "Performance Cloning: A Technique for Disseminating Proprietary Applications as Benchmarks," in *2006 IEEE International Symposium on Workload Characterization*, pp. 105–115, Oct 2006.

[21] A. Joshi, L. Eeckhout, and L. John, "The return of synthetic benchmarks," in *2008 SPEC Benchmark Workshop*, pp. 1–11, 2008.

[22] Q. Niu, J. Dinan, Q. Lu, and P. Sadayappan, "PARDA: A Fast Parallel Reuse Distance Analysis Algorithm," in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 1284–1294, May 2012.

[23] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The Gem5 Simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, Aug 2011.

[24] J. Edler and M. D. Hill, "Dinero IV trace-driven uniprocessor cache simulator," 1998.

[25] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *SIGARCH Comput. Archit. News*, vol. 34, pp. 1–17, Sep 2006.

[26] J. L. Wolf, H. S. Stone, and D. Thiébaut, "Synthetic Traces for Trace-Driven Simulation of Cache Memories," *IEEE Trans. Comput.*, vol. 41, pp. 388–410, Apr 1992.

[27] J. Weinberg and A. E. Snavely, "Accurate Memory Signatures and Synthetic Address Traces for HPC Applications," in *Proceedings of the 22Nd Annual International Conference on Supercomputing*, ICS '08, (New York, NY, USA), pp. 36–45, ACM, 2008.

[28] W. S. Wong and R. J. T. Morris, "Benchmark synthesis using the LRU cache hit function," *IEEE Transactions on Computers*, vol. 37, pp. 637–645, Jun 1988.

[29] L. Eeckhout and K. D. Bosschere, "Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces," in *Parallel Architectures and Compilation Techniques, 2001. Proceedings. 2001 International Conference on*, pp. 25–34, 2001.

[30] R. K. V. Maeda, P. Yang, X. Wu, Z. Wang, J. Xu, Z. Wang, H. Li, L. H. K. Duong, and Z. Wang, "JADE: A Heterogeneous Multiprocessor System Simulation Platform Using Recorded and Statistical Application Models," in *Proceedings of the 1st International Workshop on Advanced Interconnect Solutions and Technologies for Emerging Computing Systems*, AISTECS '16, (New York, NY, USA), pp. 8:1–8:6, ACM, 2016.

[31] C. M. Olschanowsky, M. M. Tikir, L. Carrington, and A. Snavely, "PSnAP: Accurate Synthetic Address Streams Through Memory Profiles," in *Proceedings of the 22Nd International Conference on Languages and Compilers for Parallel Computing*, LCPC'09, (Berlin, Heidelberg), pp. 353–367, Springer-Verlag, 2010.

[32] Y. Zhong, C. Ding, and K. Kennedy, "Reuse distance analysis for scientific programs," in *Proceedings of Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, 2002.

[33] C. Nugteren, G. J. van den Braak, H. Corporaal, and H. Bal, "A detailed GPU cache model based on reuse distance theory," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 37–48, Feb 2014.

[34] M. Badr and N. E. Jerger, "SynFull: Synthetic traffic models capturing cache coherent behaviour," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 109–120, June 2014.

[35] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen, *Is Reuse Distance Applicable to Data Locality Analysis on Chip Multiprocessors?*, pp. 264–282. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[36] J. Shao, *Reducing main memory access latency through SDRAM address mapping techniques and access reordering mechanisms*. PhD thesis, Michigan Technological University, 2006.

[37] N. Beckmann and D. Sanchez, "Modeling cache performance beyond LRU," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 225–236, March 2016.

[38] X. Shen, J. Shaw, B. Meeker, and C. Ding, "Locality Approximation Using Time," in *Proceedings of the 34th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '07, (New York, NY, USA), pp. 55–61, ACM, 2007.

[39] X. Xiang, C. Ding, H. Luo, and B. Bao, "HOTL: A Higher Order Theory of Locality," *SIGARCH Comput. Archit. News*, vol. 41, pp. 343–356, Mar 2013.

[40] J. Weinberg, M. O. McCracken, E. Strohmaier, and A. Snavely, "Quantifying Locality In The Memory Access Patterns of HPC Applications," in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, (Washington, DC, USA), pp. 50–, IEEE Computer Society, 2005.

[41] E. Berg and E. Hagersten, "Fast Data-locality Profiling of Native Execution," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, pp. 169–180, June 2005.

[42] S. Ghosh, M. Martonosi, and S. Malik, "Cache Miss Equations: A Compiler Framework for Analyzing and Tuning Memory Behavior," *ACM Trans. Program. Lang. Syst.*, vol. 21, pp. 703–746, July 1999.