ACCORD: an Analytical Cache Contention model using Reuse Distances for modern multiprocessors

Rakhi Hemani*, Subhasis Banerjee** and Apala Guha*

*Indraprastha Institute of Information and Technology, Delhi, India.
**Siemens Corporate Technology, Research and Technology Center, India.
{*rakhih,apala*}*@iiitd.ac.in*
subhasis.banerjee@siemens.ac.in

***Abstract*—Simultaneous execution of multiple threads on multicores is necessary for good resource utilization. However, such utilization calls for accurate models to predict the impact on performance due to contention of shared resources, primarily the last level cache and memory. The major challenges in developing such a model for commercial multicore machines are the unavailability of cache implementation details and the scalability of the performance prediction model for multiple threads. In this paper we propose a cache contention model addressing both these challenges. We leverage observed cache behaviour and reuse distance profile of applications for this purpose. We implement our model on a Xeon Sandy Bridge multicore and observe an RMS error of less than 0.06, for single threaded and multi-threaded workloads. Further we compare the effectiveness of using ACCORD model against the popular LRU Model and find that ACCORD is upto 2.7 times more accurate.**

I. Introduction

The power of multicores, which have become ubiquitous today, can be leveraged only by executing multiple threads simultaneously on the same processor. However, doing so leads to contention for shared resources, such as the last level cache, front side bus and the main memory, resulting in performance degradation. An analysis of performance degradation of applications on commercial machines is critical for guaranteeing QOS of latency sensitive applications [1]. The first step in quantifying performance degradation is last level cache contention modelling. However, there are two major challenges in building cache contention models for commercial multicores: a) the details of the cache implementation policy of commercial multicores are unknown, b) it is difficult to build a scalable model, i.e., a model that is readily applicable to any number of co-running threads, without changing the model. In this paper we propose an analytical cache performance model for multicores addressing both these challenges: an Accurate Cache COntention model using Reuse Distance (ACCORD). We verify our model on a Xeon Sandy Bridge server machine and find that our model has an RMS error of less than 0.06.

A cache contention performance model comprises of two distinct components a) cache modelling b) application (thread) modelling. The major challenge in cache modeling for commercial multicores is that the details about the actual cache implementation policy, (cache replacement policy in particular) are not disclosed by the processor manufacturer. Thus, it is not possible to model the cache using existing analytical approaches as suggested by [2], [3]. To solve this problem we present a novel method to characterise cache behaviour based on observing the relationship between cache miss rate and reuse distance empirically. We define reuse distance between two memory accesses to the same location as the number of distinct memory accesses between them that go to the same last level cache set as these two memory accesses. Note, this definition is slightly different from the standard reuse distance definition. Further, most existing analytical cache contention models assume LRU cache replacement policy [4], [5], [6]. We show that the observed relationship between reuse distance and cache miss rate deviates significantly from LRU. Hence, our model is more accurate in practice than existing models as it is based on observed cache behaviour rather than assuming LRU cache replacement policy.

For thread modelling, the major challenge is to use characteristics that permit scalability. i.e. the model should work for any number of co-running threads without requiring a change in the underlying model. To solve this challenge, we characterise a thread's memory access pattern using two fundamental characteristics: reuse distance profile and cache access rate. The advantage of our approach is that given these two metrics, our model is scalable for any number of threads. Note that many existing approaches for predicting performance degradation on commercial multicores employ learning based models [1], [7], [8]. These approaches have a major drawback, there is no proof on the scalability of these models, i.e., the authors do not provide results on the accuracy of the prediction made by their models with varying number of co-running threads.

In summary, the key contributions of this paper are:

1) ACCORD: an Analytical Cache COntention model based on Reuse Distances. Our performance model is based on the observed cache response rather than (mostly unknown) knowledge of cache implementation policies.
2) Extensions to ACCORD based on reuse distance profile, making ACCORD scalable for a varying number of co-running threads.
3) A demonstration of the effectiveness of the ACCORD on a server class Intel Xeon Sandy Bridge processor, for both single and multi-threaded execution environments.

This paper is organised as follows: In Section II, we motivate for this study by demonstrating the relationship

between reuse distance and cache miss rate. Next we use this relationship termed as cache response model to define our model ACCORD in Section III. Further this section presents the extension to ACCORD for multiple threads. We give the details of the metric used for validating ACCORD, experimental setup and methodology in Section IV. The results for validation of ACCORD for both single threaded and multi-threaded execution environments are given in Section V. The related work is given in section VI. Finally we conclude and give future directions of our work.

II. Motivation

It is well known that commercial caches do not implement the LRU cache replacement policy, however the details of the exact cache replacement policy are unknown. Infact most existing cache contention models [4], [5], [6] assume LRU cache replacement policy. Hence it is important to know the deviation of actual cache behaviour from that predicted by LRU cache replacement policy.

For this purpose we first found the relationship between cache behaviour and reuse distance. Recall that reuse distance has been defined as the number of distinct cache accesses between two accesses to the same memory location such that each intermediary access corresponds to the same cache set. If a pure LRU cache policy is implemented on a processor, then a cache access is a miss if its reuse distance is greater than or equal to the associativity of cache otherwise it is a hit. To demonstrate the inaccuracy of this rule we conducted an experiment on our test machine and found the actual miss rate corresponding to varying reuse distances.

We created threads, such that they posted many cache accesses with a specified reuse distance and then compared the actual miss rate with the LRU model's predicted miss rate. Note that miss rate is defined as the ratio of the number of cache misses to the number of memory accesses.

For conducting this experiment we used a benchmark, Parameterised_Access, (figure 1). The benchmark takes a value rd as input and generates cache accesses with a single reuse distance - rd. The input to the code is a sufficiently large $buffer$, the number of sets in the cache, num_sets, the line size of the cache, $line\_size$ and the desired reuse distance, rd. Assume that the memory addresses accessed (or read) during the first iteration of outer for loop (lines 4-8) are add_1, add_2, .. ,add_N. As all these addresses are consecutive and linesize elements apart, they are mapped to different cache sets. Thus there is a single cache access to every cache set. If U is greater than 1, then the addresses of accesses in the second iteration are add_{N+1}, add_{N+2}, .. , add_{2N+1}. These accesses are also mapped to different cache sets. Thus the number of accesses to every cache set increases to 2. Also, these accesses are distinct as they refer to different memory addresses. At the end of the first iteration of the outermost infinite loop (lines 3-9), the number of distinct accesses to every cache set is U. In subsequent iterations of the outermost loop, the previous access is repeated, i.e first accesses to addresses add_1, add_2 .. add_N are posted. Then accesses to addresses add_{N+1}, add_{N+2}, .. , add_{2N+1} are posted and so on. Each access gets mapped to the same cache set as previously. Thus for each cache set the same U locations are accessed repeatedly. This implies that

```
1: procedure PARAMETERISED_ACCESS( buffer, num_sets, line_size, rd )
2:     U = rd + 1
3:     loop
4:         for i ← 1, U do
5:             for j ← 1, num_sets do
6:                 read buffer[(i * num_sets + j) * line_size]
7:             end for
8:         end for
9:     end loop
10: end procedure
```

Fig. 1. A Benchmark with parameterised Reuse Distance rd

the reuse distance of all cache accesses after the first iteration of the outermost infinite loop is U-1 or rd.

Our test machine has private L1 and L2 cache of sizes 32KB and 256KB respectively, the associativity of both these caches is 8. The L3 cache is a shared cache of size 20MB with cache associativity as 20. Further line size for all cages is 64bytes. To measure cache miss rate we used the perf tool. Further details on our experimentation setup can be found in Section IV. Note that L2 cache hits are eliminated if the number cache accesses in a single iteration of the outermost loop are much larger than the size of L2 cache itself. This is because in this scenario two accesses to the same memory location are separated by a sequence of distinct accesses, with length larger than the size of L2 cache. For our test machine, this is true for rd value greater than 4.

The results of running this benchmark alone on our test machine, are plotted in Figure 2. Blue rhombuses plot the observed (or actual) miss rate, where as the red squares plot the miss rate as predicted by the LRU cache replacement policy. The maximum value of reuse distance considered is 39 as the cache associativity of the last level cache on our machine is 20. It was found that after this value the observed miss rate converges to 1. These results prove that the actual cache behaviour deviates significantly from that predicted by LRU. This relationship forms the basis for our model and is used to model cache behaviour. Note that this experiment can be used to characterise the behaviour of any processor without knowledge of its actual cache implementation policy. Our results conform with the finding in [3]. However, the authors have modeled the cache for a simulated setup where cache implementation details are known.

III. ACCORD Model

A. Notation

For clarity, some terms used frequently in this paper are defined in table I.

A cache access is represented as a and the memory location referred by it is represented as $loc(a)$. For brevity, the memory location corresponding to a is represented by just loc. A sequence of cache accesses, Seq, is defined as an ordered sequence of cache accesses to the same cache set. The starting address of Seq is represented as a_1 and its length by N. The accesses in the sequence are numbered as a_1, a_2, .. ,a_N. The memory locations corresponding to each access are not

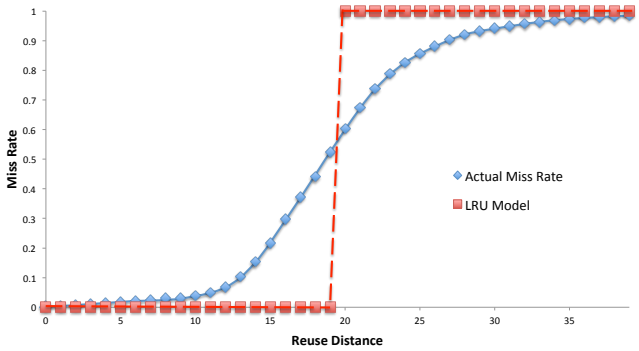

Fig. 2. A comparison between actual miss rate and the miss rate predicted by LRU model.

TABLE I. KEY NOTATIONS

Notation	Description
a	A cache access, the memory location it refers is $loc(a)$ or simply loc
$Seq(a_1, N)$	The ordered sequence of cache accesses, a_1, a_2, .. a_N where all accesses refer to the same cache set.
$Cseq(a_1, N)$	A circular sequence, i.e. a $Seq(a_1, N)$, where a_1 and a_N refer to the same memory location.
$RD(Cseq)$	Reuse Distance corresponding to a circular sequence. It is equal to $Uniq(Cseq) - 1$.
$RD(a_x)$	Reuse distance of a cache access a_x. It is equal to $RD(Cseq)$ such that length of Cseq is minimal and the last access in $Cseq$ is a_x.
PRD_{thread}	Reuse distance profile of a thread. It is a vector where $PRD_{thread}[i]$ is the probability that the reuse distance of any cache access is i.

necessarily distinct. In-fact many accesses in a sequence may refer to the same memory location. A circular sequence is defined as a sequence with the first and last accesses referring to the same memory location. Reuse distance of a circular sequence is defined as the number of unique memory locations accessed in it. Reuse distance of an access, $RD(a_x)$, is defined as the reuse distance of the minimal length circular sequence ending with a_x. Finally reuse distance profile of a thread is defined as a vector PRD_{thread} where $PRD_{thread}[i]$ is the probability that a cache access has reuse distance i. For reasons explained before, the length of PRD vector is limited to 40 for all the models.

B. Basic ACCORD Model

In the previous section we described our methodology for obtaining the relationship between reuse distance and cache miss rate for a commercial processor, without knowledge about cache implementation policy. Our model is based directly on this relationship. The basic idea behind this model is that if the number of cache accesses is large, each entry of data represented by blue line in figure 2 may be interpreted as the estimated miss rate for a given reuse distance (denoted as PMR_{rd}).

The miss rate of a thread, $\hat{MR}_t$, is modelled as the sum of the product of the probability of a particular reuse distance (PRD, refer to Table I) and the estimated miss rate for that reuse distance.

$$\hat{MR}_t = \sum_{rd \in RD_t} PRD_t(rd) * PMR_{rd} \quad (1)$$

The model as formulated above is not directly applicable when multiple applications are co-running. Next, we present extensions to the model that enable its use in a multi-threaded environment.

C. Extension to ACCORD model for multi-threaded environment

In the multi-threaded scenario, the thread whose miss rate is being predicted is termed as the victim and the other co-running threads are termed as aggressors. The miss rate of the victim thread in presence of aggressors is more than its miss rate when run in isolation. This is because accesses made by aggressor threads may evict some data of the victim thread, turning some hits into misses. However, these evictions are not arbitrary. As demonstrated before, probability of a cache access turning into a miss is dependent upon it's reuse distance. In the context of multiple co-running threads, this is restated as, the probability of a cache access turning into a miss is dependent upon it's *effective* reuse distance (or concurrent reuse distance [9]). Effective reuse distance of a cache access by thread t, t_a_x to memory location t_loc_x, is defined as the number of distinct accesses posted by all threads, between t_a_x and the prior access to t_loc_x.

For example, suppose the victim thread posts a sequence of four accesses, and the memory locations accessed are

$$v\_loc_0, v\_loc_1, v\_loc_2, v\_loc_0.$$

Then reuse distance of the last access is two. Further assume that in the multi-threaded scenario the aggressor thread inserts a single access agg_a in the sequence. The sequence of accesses changes and the effective reuse distance of the last access, v_a_4 increases to three.

$$v\_loc_0, v\_loc_1, agg\_loc, v\_loc_2, v\_loc_0$$

Note that the victim thread posts accesses with the same reuse distance profile as before, however from the cache perspective the reuse distances of the accesses is more. Denoting the estimate of effective reuse distance as ξ_{rd}, the miss rate of victim can be computed as

$$\hat{MR}_{victim} = \sum_{rd \in RD_{victim}} PRD_{victim}(rd) * PMR_{\xi_{rd}} \quad (2)$$

Effective reuse distance of a cache access by thread t, t_a_x, was defined as the number of distinct accesses posted by all threads, between t_a_x and a prior access to t_loc_x. The number of distinct accesses posted by thread t itself is the reuse distance of t_a_x. Let, the number of distinct accesses by aggressor thread, i, be represented as $U_{aggi}(t\_a_x)$. Then, for N aggressors, the effective reuse distance for t_a_x is

$$eff\_rd(t\_a_x) = rd(t\_a_x) + \sum_{i=1}^{N} U_{aggi}(t\_a_x) \quad (3)$$

Similarly we define the estimate of effective reuse distance as

$$\xi_{rd} = rd + \sum_{i=1}^{N} \mu_{aggi}(rd) \quad (4)$$

where $\mu_{aggi}(rd)$ is an estimate of unique accesses posted by $aggi$ for a given reuse distance rd of the victim. To estimate $\mu_{aggi}(rd)$, we follow a method similar to that described in [4], [5] and [9].

For all threads we first compute the maximum number of unique addresses that can be posted to a single set as the maximum possible reuse distance plus one. Then we compute a vector $Uniq$, such that $Uniq[i]$ denotes the number of accesses required to post i unique elements. Note that this information can be easily generated from the address trace, for our purpose, we find this information as an average obtained by considering a thousand synthetic traces. Each synthetic trace is generated by randomly generating accesses with the reuse distance profile as PRD.

Now the number of unique accesses posted by aggressor i, $\mu_{aggi}(rd)$, is calculated using the unique vector of the victim, $Uniq_{victim}$, and aggressor i, $Uniq_{aggi}$, cache access rate of victim, AR_{victim} and aggressor i, AR_{aggi} as follows:

First compute the number of cache accesses by the victim required to post $rd + 1$ unique accesses. Note that if an access with reuse distance rd is posted, then the number of unique accesses posted is rd+1. Now compute the time, t_{rd} required by the victim thread to post these many accesses using AR_{victim}. Now using the argument that the aggressor thread also progresses by t_{rd} time, we first find the number of cache accesses posted by the aggressor i using AR_{aggi}. Next, it can be seen that $\mu_{aggi}(rd)$ is simply the number of unique accesses posted by aggressor i, for the above computed number of cache accesses by the aggressor.

IV. Experimental Setup

A. Metrics

The major goals in this paper are to validate the accuracy of ACCORD model as compared to the observed miss rate, and compare the accuracy of the ACCORD model with the LRU model. To facilitate this comparison the following metric is used:

The actual miss rate is defined as the ratio of actual cache misses to the actual cache accesses. Perf tool is used to find the number of L3 cache misses and accesses. Further to guarantee accuracy the actual value is considered as the average of ten observations.

To compare the predicted miss rate, $\hat{MR}$, to the actual miss rate, MR, Root Mean Square (RMS) error is used. RMS error for n observations of MR and the corresponding predictions $\hat{MR}$ is defined as:

$$RMS\ Error = \sqrt{\frac{\sum_{i=1}^{n} \left(MR_i - \hat{MR_i}\right)^2}{n}}$$

In addition to validating ACCORD model, we compare its accuracy with the LRU model.

B. Combine Benchmark

To validate the applicability of ACCORD model over the LRU model in a generic case, it is essential to consider threads with a wide range of reuse distance profiles and then compare the actual miss rate to the predicted miss rate. To generate a thread with a randomly variable reuse distance profile we use a benchmark *Combine*.

The design of the Combine benchmark is presented in figure 3. This is similar to the design of Parameterised_Access benchmark (given in figure 1). Note that the major difference between Parameterized_Access and Combine is that while the former benchmark accepts a scalar value rd as input and posts all accesses having the same reuse distance rd, the Combine benchmark explores the idea of posting variable length sequences, controllable via an input vector $init\_prob$.

Notice that the number of iterations in the outermost for loop (line 10) of the Combine benchmark is given by the variable l, thus l controls the length of the sequence of accesses posted (lines 10 - 14) to each cache set. This rationale has been explained earlier in section II. The value of l may change in different iterations of outermost infinite loop (lines 2 - 15), thus the length of sequence posted is variable. However an access pattern in the accesses is maintained. For example assume that l is equal to 3 for some iteration and memory locations (loc_1, loc_2, loc_3) are accessed in a cache set. In the subsequent iteration if l changes to 2, then the memory locations (loc_1, loc_2) are accessed. Note that if the maximal sequence is represented by $\sigma = (loc_1, ...loc_m)$, then each posted sequence is a subset of σ with the first access always referring to location loc_0. The value of l, posted at a given iteration of the outer infinite loop (lines 2 - 15), changes randomly with the probability density function given by $init\_prob$. Given the $init\_prob$ vector, Lines 3 - 9 indicate one possible way of computing l.

It can be shown that the reuse distance profile of a thread running Combine is dependent upon $init\_prob$ vector. Next we prove that the probability of posting an access of reuse distance j is given as:

$$PRD(j) = \frac{j \ .\ init\_prob[j+1]}{\sum_l \sum_{k \geq l} init\_prob[k]} \quad (5)$$

Consider that the $init\_prob$ input to the Combine benchmark is [0.1, 0.3, 0.6]. Then the reuse distance profile generated by Combine benchmark is given in figure 4.

Figure 5 illustrates the memory locations accessed by all possible sequences. The figure also illustrates the expected occurrences of each sequence considering a total of hundred sequences.

For reuse distance calculation consider an ordering on sequences as Seq_0, Seq_1 and so on. The reuse distance of an

```
1: procedure COMBINE(buffer, num_sets, line_size, init_prob)
2:     loop
3:         X ← RANDOM(0, 1)
4:         sum ← init_prob[1]
5:         l ← 1
6:         while sum < X do
7:             l ← l + 1
8:             sum ← sum + init_prob(l)
9:         end while
10:        for i ← 1, l do
11:            for j ← 1, num_sets do
12:                read buffer[(i * num_sets + j) * line_size]
13:            end for
14:        end for
15:    end loop
16: end procedure
```

Fig. 3. Combine Benchmark

Reuse Distance	PRD
0	0.04
1	0.24
2	0.72

Fig. 4. Reuse distance profile for Combine benchmark with $init\_prob$ vector as [0.1, 0.3, 0.6]

Length of Sequence	Memory Locations of Sequence	Expected Occurences
1	loc1	10
2	loc1, loc2	30
3	loc1, loc2, loc3	60

Fig. 5. Example values for $init\_prob$ of Combine Benchmark

access in Seq_x depends upon its preceding sequences. Given a pair of sequences, Seq_{x-1} and Seq_x, the reuse distance of some accesses in Seq_x is known. For an illustration consider figure 6. All possible pairs of sequences of the current example, are illustrated in the figure. The prior sequence is highlighted in yellow, and some accesses, the accesses for which the reuse distance is known, are highlighed in green. It is observed that the reuse distance of the access to location loc_i is (j-1), if j is greater than or equal to i and length of prior sequence is j. However if the length of prior sequence is less than i, then the reuse distance of access to loc_i is unknown. To find the reuse distance in this case all sequences prior to the access to loc_i need to be considered in order till a sequence of length j greater than i is found. The reuse distance of the access is now

Reuse Distance	Pairwise Sequences
0	loc1 | loc1 ; loc1 | loc1, loc2 ; loc1 | loc1, loc2, loc3
1	loc1, loc2 | loc1 ; loc1, loc2 | loc1, loc2 ; loc1, loc2 | loc1, loc2, loc3
2	loc1, loc2, loc3 | loc1 ; loc1, loc2, loc3 | loc1, loc2 ; loc1, loc2, loc3 | loc1, loc2, loc3

Fig. 6. All possible pairings of access sequences for the example. The reuse distance of the cells marked in green is computable.

Reuse Distance \ loc_i	1	2	3
0	0.1	-	-
1	0.3	0.33	-
2	0.6	0.67	1

Fig. 7. Probabilities as computed by Equation 6

Memory Location	Expected Occurrences	Prob
loc1	100	0.4
loc2	90	0.36
loc3	60	0.24

Fig. 8. Probability of accessing a particular location

computable as j-1. For example, consider the pair of sequences

$$loc_1 \mid loc_1, loc_2$$

The reuse distance of the last access, i.e. the access to loc_2 is unknown. Suppose these sequences are a part of bigger sequence

$$loc_1, loc_2 \mid loc_1, loc_2, loc_3 \mid loc_1 \mid loc_1, loc_2$$

TABLE II. CACHE HIERARCHY CONFIGURATION

Type	Size per core	Associativity	Linesize	Type
L1	32 KB	8	64	Private Data
L2	256 KB	8	64	Private Unified
L3	20 MB	20	64	Shared Unified Distributed

Now the reuse distance of the last access (loc_2) can be estimated as 2. Notice that if the ordering is

$$loc_1, loc_2, loc_3 \mid loc_1, loc_2 \mid loc_1 | loc_1, loc_2$$

then the reuse distance of the last access (loc_2) is 1. Thus for calculation of reuse distance of an access pertaining to a location loc_i, the sequences of length lesser than i are not considered. This is expressed as:

$$prob(rd = j | loc_i) = \frac{init\_prob[j]}{\sum_{k \geq i} init\_prob[k]} \quad (6)$$

For the example figure 7, presents these values for every possible loc_i and reuse distance.

The probability of an access to location loc_i is the simply the ratio of number of expected accesses to loc_i and total number of accesses. For the example under consideration these probabilities are illustrated in figure 8.

$$prob(loc_i) = \frac{\sum_{k \geq i} init\_prob[k]}{\sum_l \sum_{k \geq l} init\_prob[k]} \quad (7)$$

Finally probability of an access with reuse distance (j-1) can be computed as the product as defined below

$$PRD(j-1) = \sum_{i \leq j} (prob(rd = j | loc_i) \; . \; prob(loc_i)) \quad (8)$$

After some simplification this equation get simplified to equation 5. For the example under consideration the reuse distance profile had been specified in figure 4.

C. Test Machine Configuration

All our experiments were run on a Dell R720 Server equipped with a Sandy Bridge Intel Xeon processor. The clock rate of the processor is 2.4 GHz. The cache hierarchy configuration of the processor is given in Table II. The size of the main memory is 64 GB. The BIOS of the processor was set to enable maximum performance, disable prefetching and logical processors. The linux kernel on the machine is 3.11, and all code was compiled using gcc compiler version 4.7.2.

The numactl system call was used to pin threads to physical processors. Perf tool was used to measure performance data, viz. number of last level cache misses and cache accesses.

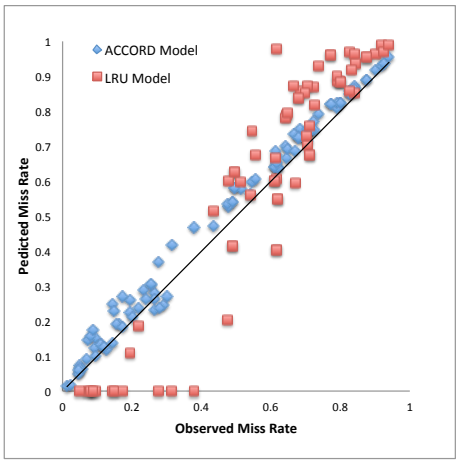

Fig. 9. ACCORD Model versus LRU model for single thread

D. Methodology

For validating the accuracy of the ACCORD model we consider both single threaded and multi-threaded environments. Recall that in a multi-threaded scenario, the thread whose miss rate is being predicted is known as the victim and all the other threads are known as the aggressors. Experiments considering upto three aggressor threads were conducted. Note that most previous studies proposing cache performance models for real machines report results of their model's considering either only a pair of threads [6] and [1] or a fixed number of threads [8].

In our experiments, we considered multithreaded scenarios with the victim thread co-running with upto three aggressors. Thus a total of four scenarios were considered. Each experimental scenario, x, comprises of co-running a victim thread and x-1 aggressor threads. For each scenario, a hundred sub scenarios are considered. Corresponding to each sub scenario, the victim and aggressor threads were assigned random reuse distance profiles using the Combine benchmark. The aggressor threads were run on different cores first and then the victim thread was run. The number of last level cache misses and accesses, made by the victim thread were observed. Further an average of 10 such observations is considered for each sub scenario. The aggressor and victim threads are run in single threaded mode to obtain the access rate. Using the access rate and reuse distance profile information of the victim and aggressor threads, cache miss rate is predicted using ACCORD and LRU model. The result obtained by comparing these predictions with observed values is discussed next.

V. RESULTS

A. Validation for a single thread

The result of running the victim thread alone, i.e. scenario 1, is graphically depicted in figure 9. The figure is a scatter plot comparing the predictions of the LRU model and the ACCORD model. The prediction of the ACCORD model is given by blue rhombuses and the prediction of the LRU model by red squares. The straight line denotes the ideal prediction. It can be seen that the prediction made by the ACCORD model is more accurate than the prediction made by the LRU model. Further it can be observed that for small miss rates the LRU

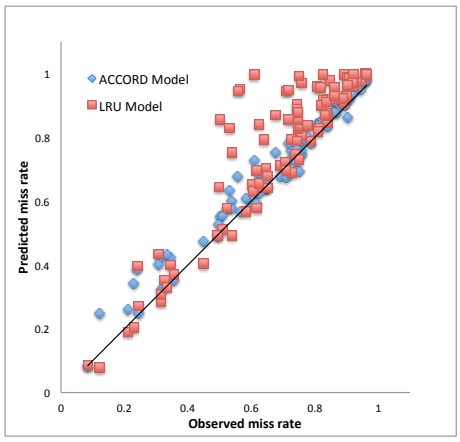

Fig. 10. ACCORD Model versus LRU model for a thread with 1 aggressor

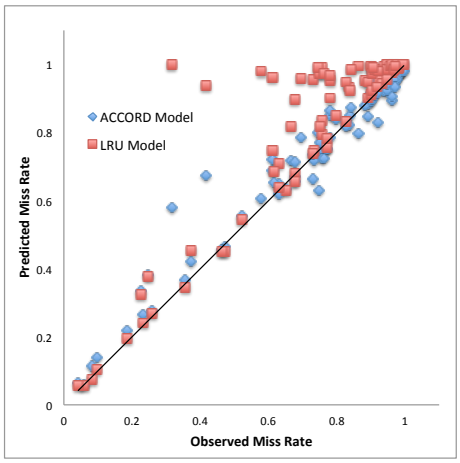

Fig. 11. ACCORD Model versus LRU Assumption for a thread with 2 aggressors

model under predicts and for high miss rates LRU model over predicts. This is in accordance to the result obtained in Figure 2.

B. Validation for multiple threads

The scatter plots for the multithreaded scenarios corresponding to co-running the victim thread and 1, 2 and 3 aggressor threads are depicted in figures 10, 11 and 12 respectively.It can be observed that the predictions made by the ACCORD model are better than the LRU model as they are closer to the straight line. Further it is observed that as the number of threads increase the points get more concentrated towards higher miss rates. This is expected as with an increase in multithreading, the effective reuse distances of the victim thread increase, pushing its miss rate closer to 1.

C. Discussion

The RMS error of the predictions made by the ACCORD and LRU models for both, single and multithreaded scenarios is presented in figure 13. It can be seen that the RMS error of ACCORD is always lesser than 0.06 and is lesser than that of the LRU model. The RMS error of ACCORD model was found to be upto 2.7 times better than the LRU model.

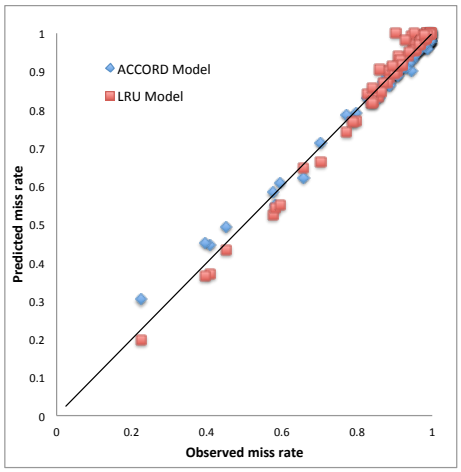

Fig. 12. ACCORD Model versus LRU Assumption for a thread with 3 aggressors

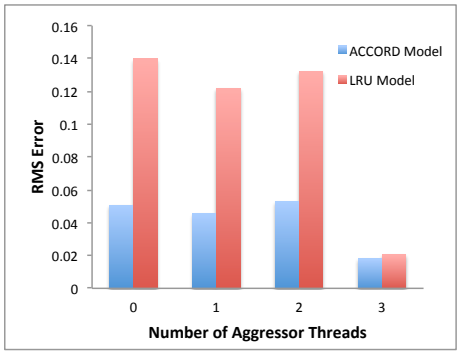

Fig. 13. RMS error of prediction of miss rate by ACCORD model, LRU model and Pairwise Sum

It can be observed that the RMS error decreases significantly when 3 aggressors are co-run. As stated earlier, this is because the effective reuse distance of the victim thread tends to be high, thus most accesses are misses and the predictions of both ACCORD and LRU models are accurate. We repeated our experiment for 4 and 5 aggressor threads also, reducing the maximum possible reuse distance of aggressor threads. The reduction in maximum possible reuse distance was to ensure some variation in miss rate of victim. The results of these experiments are plotted in figure 14 and figure 15. These results further prove the efficiency of ACCORD in the presence of multiple threads. Note that for these cases, the miss rate of the victim thread is limited to a narrow range of [0.8,1], and both LRU and ACCORD model give good predictions for these miss rates.

VI. Related Work

As the number of cores sharing the last level cache continues to scale, it is important to analyse the impact of cache sharing on performance. Many previous studies have proposed analytical cache contention models. Chandra et al. proposed an inductive cache contention performance model in [5]. This model is extended in [4]. However the accuracy of both these models are limited by the LRU assumption. Also the applicability of these models to a commercial machine is

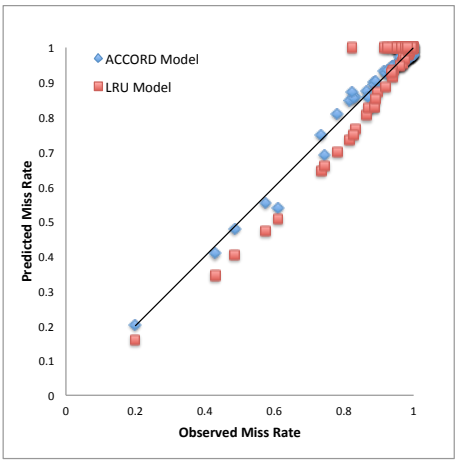

Fig. 14. ACCORD Model versus LRU Assumption for a thread with 4 aggressors

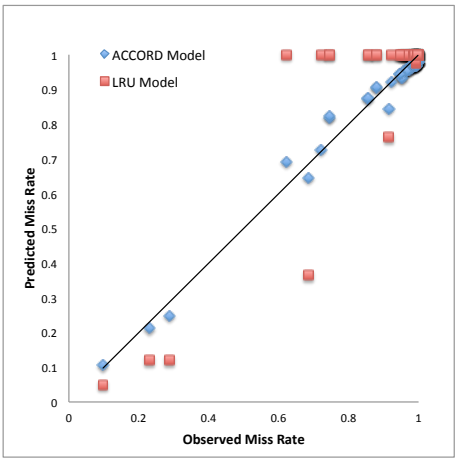

Fig. 15. ACCORD Model versus LRU Assumption for a thread with 5 aggressors

not known. ACCORD is intended to complement these models by relating reuse distances to cache miss rates more accurately.

Xu etal have proposed a detailed cache performance model and validated it on a commercial processor in in [6]. However the accuracy of this model is also limited by the LRU assumption.

Many research efforts targeting performance degradation due to contention use learning based models. Unlike the previous models, these models are not limited by the LRU assumption and are proved to be efficient on commercial processors. The bubble-up model [1], and the empirical model [8] are good examples of such approaches. The major drawback of these models is that they require a lot of experiments to drive their regression models. Further, there is no proof on the scalability of the models. The bubble-up model [1] is applicable for only 2 threads and the empirical model [8] is shown to be applicable for the maximum number of cores on the machine. The basic reason behind the non-scalability of these models is that they are based on the response of applications and do not characterise application sensitivity using fundamental application characteristics. In contrast ACCORD is based on reuse distances and does not suffer from these drawbacks.

Another class of analytical cache models concentrate on modelling the cache replacement policy [2], [3]. The major drawback of these models for commercial machines is that they assume complete knowledge about the cache implementation policy. Further the pseudo-LRU cache model proposed in [3] is applicable only for power of two cache associativity. Recall that our test machine has last level cache associativity of 20. Thus these models are not directly applicable to our setup.

VII. Conclusion and Future Work

To conclude this paper presents an accurate cache performance model ACCORD. The RMS error of the ACCORD model is less than 0.06 and it is upto 2.7 times more accurate than the widely used LRU model. The accuracy of the ACCORD model has been tested for multithreaded environment also. The major advantage of ACCORD lies in its ease of computation. We also propose the design of a benchmark combine, that can be used to quickly create threads with variable reuse distance profile.

We plan to extend the current work by validating it for a wider range of processor architectures and for real applications.

References

[1] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155650

[2] F. Guo and Y. Solihin, "An analytical model for cache replacement policy performance," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 228–239, Jun. 2006. [Online]. Available: http://doi.acm.org/10.1145/1140103.1140304

[3] R. Sen and D. A. Wood, "Reuse-based online models for caches," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '13. New York, NY, USA: ACM, 2013, pp. 279–292. [Online]. Available: http://doi.acm.org/10.1145/2465529.2465756

[4] X. Chen and T. Aamodt, "Modeling cache contention and throughput of multiprogrammed manycore processors," *Computers, IEEE Transactions on*, vol. 61, no. 7, pp. 913–927, July 2012.

[5] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multi-processor architecture," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, Feb 2005, pp. 340–351.

[6] C. Xu, X. Chen, R. Dick, and Z. Mao, "Cache contention and application performance prediction for multi-core systems," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, March 2010, pp. 76–86.

[7] L. Tang, J. Mars, and M. L. Soffa, "Contentiousness vs. sensitivity: Improving contention aware runtime systems on multicore architectures," in *Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era*, ser. EXADAPT '11. New York, NY, USA: ACM, 2011, pp. 12–21. [Online]. Available: http://doi.acm.org/10.1145/2000417.2000419

[8] J. Zhao, H. Cui, J. Xue, X. Feng, Y. Yan, and W. Yang, "An empirical model for predicting cross-core performance interference on multicore processors," in *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 201–212. [Online]. Available: http://dl.acm.org/citation.cfm?id=2523721.2523750

[9] Y. Jiang, E. Z. Zhang, K. Tian, and X. Shen, "Is reuse distance applicable to data locality analysis on chip multiprocessors," in *Proceedings of the 19th Joint European Conference on Theory and Practice of Software, International Conference on Compiler Construction*, ser. CC'10/ETAPS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 264–282.