CMP Cache Performance Projection: Accessibility vs. Capacity

Xudong Shi, Feiqi Su, Jih-kwon Peir, Ye Xia, Zhen Yang
Dept. of Computer and Information Science and Engineering
University of Florida
Gainesville, FL, USA
{xushi, fsu, peir, yx1, zhyang}@cise.ufl.edu

Abstract—Efficient utilizing on-chip storage space on Chip-Multiprocessors (CMPs) has become an important research topic. Tradeoffs between data accessibility and effective on-chip capacity have been studied extensively. It requires costly simulations to understand a wide-spectrum of the design space. In this paper, we first develop an abstract model for understanding the performance impact with respect to data replication. To overcome the lack of real-time interactions among multiple cores in the abstract model, we propose a global stack simulation strategy to study the performance of a variety of cache organizations on CMPs. The global stack logically incorporates a shared stack and per-core private stacks to collect shared/private reuse (stack) distances for every memory reference in a single simulation pass. With the collected reuse distances, performance in terms of hits/misses and average memory access times can be calculated for multiple cache organizations. We verify the stack results against individual execution-driven simulations that consider realistic cache parameters and delays using a set of commercial multithreaded workloads. Our results show that stack simulations can accurately model the performance of various cache organizations with 2-9% error margins. The single-pass stack simulation results demonstrate that the effectiveness of various techniques for optimizing the CMP on-chip storage is closely related to the working sets of the workloads as well as to the total cache sizes.

Keywords: Performance Modeling and Projection, Stack Simulation, CMP Caches, Data replication.

I. INTRODUCTION

Organizing on-chip storage space on Chip-Multiprocessors (CMPs) has become an important research topic. Balancing between data accessibility due to wiring delay and the effective on-chip storage capacity due to data replication has been studied extensively [2, 17, 9, 22, 29, 8, 3, 24, 14, 13, 16]. The goal is to dynamically allocate data blocks closely to the requesting core for fast data access without adversely increasing expensive off-chip memory traffic. A shared cache organization provides the maximum cache capacity that leads to the least off-chip traffic. However, in such a design, data blocks are usually allocated across multiple banks, resulting in an increase in the cache access delay due to a high percentage of remote bank accesses. A private cache organization, on the other hand, reduces the cache access delay by allocating the recently-accessed blocks in the local cache. Multiple copies of a single block may exist in multiple caches to reduce remote accesses. However, multiple copies decreases the effective cache capacity and increases off-chip memory traffic.

A CMP memory hierarchy typically includes small, private instruction/data L1 caches for fast accesses. The interesting issue is the organization of the on-die L2 cache. Recently, there have been several research works [2, 17, 9, 22, 29, 8, 3] proposing various combined private/shared L2-cache organizations following two general directions. The first is to organize the L2 as a shared cache for maximizing the capacity. To shorten the access time, a portion of the L2 can be set aside for replication [29]. The second is to organize the L2 as private caches for minimizing the access time. To achieve higher effective capacity, data replications among multiple L2s are constrained and different private L2s can steal each other’s capacity by block migration [22, 3, 8]. These studies must examine a wide-spectrum of the design space. Due to the lack of an efficient methodology, near-sighted conclusions can potentially be drawn as a result of missing comprehensive views from all essential design parameters.

Analytical models provide quick performance estimations [1, 12]. However, they usually depend on statistical assumptions, and generally cannot accurately model systems with complex real-time interactions among multiple processors [7]. For fast simulations, the stack simulation technique proposed in [20] simulates multiple cache sizes in a single pass under the LRU replacement policy. Several extensions and enhancement have been made to improve the speed of the single-pass stack simulation or the coverage to variable set-associativities [20, 4, 25, 12, 15, 23]. However, these stack simulation methods target only uniprocessor caches where no cache coherence complexity is involved and the memory access delay hardly affects the order of memory requests. Extensions of the stack simulation to multiprocessors are reported in [27, 28]. Those works focus on solving the problem of multiprocessor cache invalidations in stack simulations for general set-associative caches. However, the remote cache hit, an important measure on CMP, is not included. Furthermore, the accuracy of using traces to simulate different multiprocessor cache organizations is not evaluated.

In this paper, we present a general framework for fast projection of CMP cache performance. Four L2 cache organizations, Shared, private, shared with data replication, and private without data replication are studied. We focus on understanding the performance tradeoff between data accessibility and the cache capacity loss due to data replication. The outline and contributions of our approach are as follows.
1) **Modeling Data Replication:** We first develop an analytical model to assess general performance behavior with respect to data replications in CMP caches. The model injects replicas (replicated data blocks) into a generic cache. Based on the block reuse-distance histogram obtained from a real application, a precise equation is derived to evaluate the impact of the replicas. The results demonstrate that whether data replication helps or hurts L2 cache performance is a function of the total L2 size and the working set of the application. Existing CMP cache studies may have overlooked this general replication behavior because they failed to examine the entire design space.

2) **Single-Pass Stack Simulation:** To overcome the limitations of modeling, we developed a single-pass stack simulation technique to handle shared and private cache organizations with the invalidation-based coherence protocol. The stack algorithm can handle complex interactions among multiple private caches. For fast simulations, we adopt fully-associative stacks based on hashing. We also partition the stack into fixed-size groups to avoid scanning the entire blocks for each memory request [15]. This single-pass stack technique can provide local/remote hit ratios and the effective cache size for a range of physical cache capacities.

3) **Performance Projection of Data Replication:** We demonstrate that we can use the basic multiprocessor stack simulation results to estimate the performance of other interesting CMP cache organizations. For example, given different percentages of the L2 cache reserved for data replication, we can derive the average L2 access time under a shared L2 cache organization. Such a cache organization closely resembles the L2 cache with victim replication [29].

4) **Performance Verification:** Finally, we verify the projection accuracy from the stack simulation against the detailed execution-driven simulation for each individual cache configuration using three multithreaded workloads. We observe that both the modeling and the single-pass stack simulation produce consistent performance views for CMP caches with different degrees of data replication. We also show that the single-pass stack simulation produces small error margins ranging 2-9% in comparison with execution-driven simulations for all simulated cache organizations.

The paper is organized as follows. Section 2 describes the abstract analytical model established based on a multithreaded database workload (OLTP). The modeled performance results with respect to data replication are also discussed. Section 3 introduces the CMP stack simulation algorithm that handles both the private and the shared caches. Section 4 describes the simulation and validation methodology. This is followed by performance evaluations of the four target L2 organizations: shared, private, shared with data replication, and private without data replication in section 5. The results obtained from the stack simulations are verified against realistic CMP cache simulations. Section 6 has the related work. We give a brief conclusion in section 7.

II. **MODELING DATA REPLICATION**

In this section, we develop an abstract model independent of private/shared organizations to evaluate the tradeoff between the access time and the miss rate of CMP caches with respect to data replication. The purpose is to provide a uniform understanding on this central issue of caching in CMP that is present in most major cache organizations. This study also highlights the importance of examining a wide enough range of system parameters in the performance evaluation of any cache organization, which can be costly.

In Fig. 1, a generic histogram of block reuse distances is plotted, where the reuse distance is measured by the number of distinct blocks between two adjacent accesses to the same block. A distance of zero indicates a request to the same block as the previous request. The histogram is denoted by \( f(x) \), which represents the number of block references with reuse distance \( x \). For a cache size \( S \), the total cache hits can be measured by \( \int_0^S f(x) \) dx, which is equal to the area under the range of the histogram curve from 0 to \( S \). This well-known, stack distance histogram can provide hits/misses of all cache sizes with a fully-associative organization and the LRU replacement policy.

To model the performance impact of data replication, we inject replicas into the cache. Note that regardless of cache organization, replicas help to improve the local hit rate since replicas are created and moved close to the requesting cores. On the other hand, having replicas reduces the effective capacity of the cache, and hence, increases cache misses. We need to compare effect from the increase of local hits against that from the increase of cache misses.

Suppose we take a snapshot of the L2 cache and find a total of \( R \) replicas. As a result, only \( S-R \) cache blocks are distinct, effectively reducing the capacity of the cache. Note that the model does not make reference to any specific cache organization and management. For instance, it does not say where the replicas are stored, which may depend on factors such as shared or private organization. We will compare this scenario with the baseline case where all \( S \) blocks are distinct.

First, the cache misses are increased by \( \int_0^S f(x) dx \), since the total number of hits is now \( \int_0^S f(x) dx \). On the other hand, the replicas help to improve the local hits. Among the \( \int_0^S f(x) dx \) hits, a fraction \( R \alpha \) hits are targeting the replicas. Depending on the specific cache organization, not all accesses to the replicas result in local hits. A requesting core may find a replica in the local cache of another remote core, resulting in a remote hit. We assume that a fraction \( L \) accesses to replicas are actually local hits. Therefore, compared with the baseline case, the total change of memory cycles due to the creation of \( R \) replicas can be calculated by:

\[
P_m \times \int_{S-R}^S f(x) dx - G_l \times \frac{R}{S} \times L \times \int_{S-R}^S f(x) dx
\]

where \( P_m \) is the penalty cycles of a cache miss; and \( G_l \) is the cycle gain from a local hit. With the total number of memory
Figure 1. Cache performance impact with replica accesses, \( \int_{0}^{\infty} f(x)dx \), the average change of memory access cycles is equal to:

\[
\left( P_e \times \int_{0}^{\infty} f(x)dx - G_s \times \frac{R}{S} \times L \times \int_{0}^{\infty} f(x)dx \right) / \int_{0}^{\infty} f(x)dx \tag{2}
\]

Now the key is to obtain the reuse distance histogram \( f(x) \). We conduct experiment using an OLTP workload [21] and collect its reuse distance histogram. With the curve-fitting tool of Matlab [19], we obtain the equation \( f(x) = A \exp(-Bx) \), where \( A = 6.084 \times 10^6 \) and \( B = 2.658 \times 10^{-3} \). This is shown in Fig. 2, where the cross marks represent the actual reuse frequencies from OLTP and the solid line is the fitted curve. We can now substitute \( f(x) \) into equation (2) to obtain the average change in memory cycles as:

\[
P_e \times \left( \exp(-B(S-R)) - \exp(-BS) \right) - G_s \times \frac{R}{S} \times L \times \left( 1 - \exp(-B(S-R)) \right) \tag{3}
\]

Equation (3) provides the change in L2 access time as a function of the cache area being occupied by the replicas. In Fig. 3, we plot the change of the memory access time for three cache sizes, 2, 4, and 8 MB, as we vary the replicas’ occupancy from none to the entire cache. In this figure, we assume \( G_s = 15 \), \( P_e = 400 \), and \( L = 0.5 \). Note that negative values mean performance gain. We can observe that the performance of allocating L2 space for replicas for the OLTP workload varies with different cache sizes. For a 2MB L2, the results indicate no replication provides the shortest average memory access time, while for a larger 8MB L2 cache, keeping only 35% of distinct cache blocks and allocating the other 65% of cache for the replicas has the best memory performance. With a medium-sized 4MB L2 cache, allocating 40% of the cache for the replicas has the smallest access time. These results are consistent with the reuse histogram curve shown in Fig. 2. The reuse count approaches zero when the reuse distance is equal to or greater than 2MB. It increases significantly when the reuse distance is shorter than 2MB. Therefore, it is not wise to allocate space for the replicas when the cache size is 2MB or less.

The general behavior due to data replication is consistent with the detailed simulation result as will be given in Section 5. Further discrepancies can be explained as follows. Note that the fraction of replicas cannot reach 100% unless the entire cache is occupied by a single block. For a CMP with eight cores, the fraction of the cache actually occupied by the replicas may approach three-fourth assuming the average degree of sharing is four. This is true even if the entire L2 is permissible for replication. Therefore, in Fig. 3, the average memory time increase is not very meaningful when the fraction of replicas is approaching the cache size.

In addition to cache replication, we expect the cache sharing behavior also varies with different workloads and cache sizes. It is essential to study a set of representative workloads with a spectrum of cache sizes to understand the tradeoff of accessibility vs. capacity on CMP caches. A fixed replication policy may not work well for a wide-variety of workloads on different CMP caches. Although mathematical modeling can provide understanding of the general performance trend, its inability to model sufficiently detailed interactions among multiple cores makes it less useful for making accurate performance prediction. To remedy this problem, in the following section, we will describe a global-stack based simulation for studying CMP caches.

III. ORGANIZATION OF GLOBAL STACK

Fig. 4 sketches the organization of the global stack that records the memory reference history. In the CMP context, a block address and its core-id uniquely identify a reference, where the core-id indicates from which core the request is issued. Several independent linked lists are established in the
global stack for simulating a shared and several per-core private stacks. Each stack entry appears exactly in one of the private stacks determined by the core-id, and may or may not reside in the shared stack depending on the recency of the reference. For fast search, an address-based hash list is also established in the global stack.

Since only a set of discrete cache sizes are of interest for cache studies, both the shared and the private stacks are organized as groups [15]. Each group consists of multiple entries for fast search during the stack simulation and for easy calculations of cache hits under various interesting cache sizes after the simulation. For example, assuming the cache sizes of interest are 16KB, 32KB, and 64KB. The groups can then be organized according to the stack sequence starting from the MRU entry with 256, 256, 512 entries for the first three groups, respectively, assuming the block size is 64B. Based on the stack inclusion property, the hits to a particular cache size are equal to the sum of the hits to all the groups accumulated up to that cache size. Each group maintains a reuse counter, denoted by G1, G2, and G3. After the simulation, the cache hits for the three cache sizes can be computed as G1, G1 +G2, and G1+G2+G3 respectively.

Separate shared and private group tables are maintained to record the reuse frequency count and other information for each group in the shared and private caches. A shared and a private group-id are kept in each global stack entry as a pointer to the corresponding group information in the shared and the private group table. The group bound in each entry of the group table links to the last block of the respective group in the global stack. These group bounds provide fast links for adjusting entries between adjacent groups. The associated counters are accumulated on each memory request, and will be used to deduce cache hit/miss ratios for various cache sizes after the simulation. The following subsections provide detailed stack operations.

A. Shared Caches

Each memory block can be recorded multiple times in the global stack, one from each core according to the order of the requests. Intuitively, only the first-access of a block in the global stack should be in the shared list since there is no replication in a shared cache. A first-access block is the one that is most recently used in the global stack among all blocks with the same address. The shared stack is formed by linking all the first-access blocks from MRU to LRU. Fig. 5 illustrates an example of a memory request sequence and the operations to the shared stack. Each memory request is denoted as a block address, A, B, C, ..., etc., followed by a core-id. The detailed stack operations when B1 is requested are described as follows.

- Address B is searched by the hash list of the shared stack. B2 is found with the matching address. In this case, the reuse counter for the shared group where B2 resides, group 3, is incremented.
- B2 is removed from the shared list, and B1 is inserted at the top of the shared list.
- The shared group-id for B1 is set to 1. Meanwhile, the block located on the boundary of the first group, E1, is pushed to the second group. The boundary adjustment continues up to the group where B2 was previously located.
- If a requested block cannot be located through the hash list, (i.e. the very first access of the address among any cores), the stack is updated as above without incrementing any reuse counters.
- After the simulation, the total number of cache hits for a shared cache that is large enough to include exactly the first m groups is the sum of all shared reuse counters from group 1 up to group m.

B. Private Caches

The construction and update of the private lists are essentially the same as those of the shared list, except that we link accesses from the same core together. We collect crucial information such as the local hits, remote hits, and number of replicas, with the help of the local, remote, and replica counters in the private group table. For simplicity, we assume these counters are shared by all the cores, although per-core counters may provide more information. Fig. 6 draws the
contents of the four private lists and the private group table, when we extend the previous memory sequence (Fig. 5) with three additional requests.

1) **Local/Remote Reuse Counters**

The local counter of a group is incremented when a request falls into the respective group in the local private stack. In this example, only the last request, A1, encounters a local hit, and in this case, the local counter of the second group is incremented. After the simulation, the sum of all local counters from group 1 to group m represents the total number of local hits for private caches with exactly m groups.

Counting the remote hits is a little tricky, since a remote hit may only happen when a reference is a local miss. For example, assume that a request is in the third group of the local stack; meanwhile, the minimum group id of all the remote groups where this address appears is the second. When the private cache size is only large enough to contain the first group, neither a local nor a remote hit happens. If the cache contains exactly two groups, the request is a remote hit. Finally, if the cache is extended to the third group or larger, it is a local hit. Formally, if an address is present in the local stack; meanwhile, the minimum group id of all the remote groups where this address appears is the second. When the cache contains exactly two groups, the request is a remote hit. Finally, if the cache is extended to the third group or larger, it is a local hit. Formally, if an address is present in the local group L and the minimum id of the remote groups that contains it is R, the access can be a remote hit only if the cache size is within the range from group R to L-1. We increment the remote counters for groups R to L-1 (R ≤ L-1). Note that after the simulation, the remote counter m is the number of remote hits that a cache with exactly m groups encounters. To differentiate them from the local counters, we call them accumulated remote counters.

In the example, the first highlighted request, B1, encounters a local miss, but a remote hit to B2 in the first group. We accumulate the remote counters for all the groups. The second request, A2, is also a local miss, but a remote hit to A1 in the second group. The remote counter of the first group remains unchanged, while the counters are incremented for all the remaining groups. Similar to B1, all the remote counters are incremented for C1. Finally, the last request, A1, is a local hit in the second group and is also a remote hit to A2 in the first group. In this case, only the remote counter of the first group is incremented since A1 is considered as a local hit if the cache size extends to more than the first group.

2) **Measuring Replica**

The effective cache size is an important factor for shared and private cache comparisons [2, 9, 29, 8]. The single-pass stack simulation counts each block replication as a replica for calculating the effective cache size along the simulation. Similar to the remote hit case, we use accumulated replica counters. As shown in Fig. 6, the first highlighted request, B1, creates a replica in the first group, as well as any larger groups, because of the presence of B2. The second highlighted request, A2, does not create a new replica in the first group. But it does create a new replica in the second group because of A1. Meanwhile, A2 pushes B2 out of the first group, thus reduces a replica in the first group. This new replica applies to all the larger groups too. Note that the addition of B2 in the second group does not alter the replica counter for group 2, since the replica was already counted when B2 was first referenced. Similar to B1, the third highlighted request, C1, creates a replica to all the groups. Lastly, the reference, A1, extends a replica of A into the first group because of A2. The counters for the remaining groups stay the same.

3) **Handling Memory Writes**

In private caches, memory writes may cause invalidations to all the replicas. During the stack simulation, write invalidations create holes in the private stacks where the replicas are located. These holes will be filled later when the adjacent block is pushed down from a more-recently-used position by a new request. No block will be pushed out of a group when a hole exists in the group. To accurately maintain the reuse counters in the private group table, each group records the total number of holes for each core. The number of holes is initialized to the respective group size, and is decremented whenever a valid block joins the group. Maintaining the hole-count for each group can avoid searching for the existing holes.

IV. **SIMULATION METHODOLOGY**

We use the full-system Virtutech Simics 2.2 simulator [18] to simulate an 8-core CMP system with Linux 9.0 and x86 ISA. The processor module is based on the Simics Microarchitecture Interface (MAI) and models timing-directed processors in detail. Each core has its own instruction and data L1 cache. The global stack runs behind the L1 caches and simulates every L1 misses, essentially replacing the role of L2 caches. During simulations, stack distances and other related statistics are collected as described in Section 3. The results of the single-pass stack simulation are used to derive the performance of shared or private caches with various cache sizes and the sharing mechanisms for understanding the accessibility-vs.-capacity tradeoff in CMP caches.

The results from the stack simulation are verified against execution-driven timing simulations, where detailed cache models with proper access latencies are inserted. In the
detailed timing simulation, we assume the shared L2 has eight banks, with one local and seven remote determined by the least-significant three bits of the block address. For the private L2, we model both local and remote accesses. The MOESI coherence protocol is implemented to maintain data coherence among the private L2s. For comparison, we use the hit/miss information and average memory access times to approximate the execution time behavior because the single-pass stack simulation cannot provide IPCs. Table 1 summarizes important simulation parameters.

We use three multithreaded commercial workloads, OLTP (Online Transaction Processing), Apache (Static web server), and SPECjbb (Java server), as our benchmarks. We consider the variability of these multithreaded workloads by running multiple simulations for each configuration of each workload and inserting small random noises (perturbations) in the memory system timing for each run.

V. EVALUATION AND VALIDATION

The accuracy of the CMP memory performance projection can be assessed from two different angles, the accuracy of predicting individual performance metrics, and the accuracy of predicting general cache behavior. By verifying the results against the timing simulation, we demonstrate that the stack simulation can accurately predict cache hits and misses for the four targeted L2 cache organizations, and more importantly, it can precisely project the sharing and replication behavior of the CMP caches.

One inherent weakness of stack simulation is its inability to insert accurate timing delays for variable L2 cache sizes. Since the fluctuation in memory delays may alter the sequence of memory accesses among multiple processors, we try a simple approach to insert memory delays into the stack simulation for several discrete cache sizes. In our simulation, these interesting cache sizes are 1MB, 2MB, 4MB, 8MB, to 16MB, and the corresponding simulation results are denoted as stack-1, stack-2, stack-4, stack-8, and stack-16, respectively. An off-chip cache miss latency is charged if the reuse distance is longer than each discrete cache size.

A. Hits/Misses for Shared and Private L2 Caches

Fig. 7 shows the projected and real miss rates for shared caches, where “real” represents the results from the timing simulations. In general, the stack results follow the timing results closely. For OLTP, stack-2 matches the timing results with about 5-6% average error. For Apache and SPECjbb, the difference among different delay insertions is less apparent. The stack results predict the miss ratios with about 2-6% error, except for Apache with a small 1MB cache.

There are two major factors that affect the accuracy of the stack results. The first one is cache associativity. Since we use a fully-associative stack to simulate a 16-way cache, the stack simulation usually underestimates the real miss rates. This effect is apparent when the cache size is small, due to more conflict misses. This associativity issue can be solved with more complicated set-associative stack simulations [20, 12].
For simplicity, we keep the stack fully-associative. More sensitivity studies also need to be done to evaluate L2 with smaller set associativity. The second effect is due to inaccurate delay insertions. For example, the OLTP stack-1 simulation of a larger cache inserts cache miss latency for accesses with reuse distance falling between 1M and the actual cache size, where cache hit latency should have been applied. Such wrongly inserted extra delays for larger caches cause more OS interference and context switches that may lead to more cache misses. At 4MB cache size, the overestimate of cache misses due to the extra delay insertion exceeds the underestimate due to the full associativity. The gap becomes wider with larger caches. On the other hand, the stack-16 simulation for a smaller cache mistakenly inserts hit latency, instead of miss latency, for accesses with reuse distance from the cache boundary to 16M, causing less OS interferences, thus less misses. In this case, both the full associativity and the delay insertion lead to underestimate of the real misses, which makes the stack-16 simulation the most inaccurate.

Fig. 8 shows the private cache results. The overall misses, the remote hits are compared. Note that the horizontal axis shows the size of a single core. With eight cores, the total sizes of the private caches are comparable to the shared cache sizes in Fig. 7. We can make two important observations. First, in comparison with the shared cache, the timing simulation results show that the overall L2 miss ratios are increased by 14.7%, 9.9%, 4.3%, 1.1%, and 0.5% for OLTP for the respective private cache sizes from 1MB to 16MB. For Apache and SPECjbb, the L2 miss ratios are increased by 11.8%, 4.4%, 1.1%, 1.0%, 2.2%, and 7.3%, 3.1%, 2.9%, 0.6%, 0.5%, respectively. Fig. 9 further plots the average L2 access time ratio between the private caches and the shared caches with equal capacity. When the total capacity is small, although the private-cache cases have more local hits, they also encounter more L2 misses. The private cache may have up to 50% longer average L2 access time. However, when the total capacity is sufficiently large, the private cache becomes better. With bigger caches, the difference in L2 misses diminishes, but the private L2s have much more local hits, which makes the average L2 access time shorter.

Second, the estimated miss rates and remote hit rates from the stack simulation match closely to the results from the timing simulations, with less than 10% error margin. We also simulate the effective capacity for the private-cache cases. The effective cache size is the average over the entire simulation period. In general, the private cache does reduce the cache capacity due to replicated and invalid cache entries. The effective capacity is reduced to 45-75% for the three workloads with various cache sizes. The estimated capacity based on the stack simulations is almost identical to the result from the timing simulations. Due to its higher accuracy, we use the stack-2 simulation in the following discussion.

\[ B. \text{ Shared Caches with Replication} \]

To balance accessibility and capacity, victim-replication [29] creates a dynamic L1 victim cache for each core in the
The average memory access time includes the following components. First, since the L1 and the victim cache are exclusive, the total hits to the victim cache can be estimated from the private stacks with the size of the L1 plus the size of the victim: $C_{L1} + (r*C_{L2})/n$. Note that this estimation may not be precise due to the lack of the L1 hit information that alters the sequence in the stack. Second, the hit to the shared portion of the L2 and the L2 misses can be calculated from the shared stack with the size $(1-r)*C_{L2}$. Finally, the hit to the remote cache can be calculated by subtracting the hits to the victim, the hits to the shared L2, and the L2 misses from the total L2 accesses.

Fig. 10 demonstrates the average L2 access time with static victim replication. Generally, large caches favor more replications. For a small 2MB L2, except that Apache has a slight performance gain at low replication levels, the average L2 access times increase with more replications. The optimal replication levels for OLTP are 12.5%, and 37.5% respectively for 4MB and 8MB L2. Such general performance behavior with respect to data replication for OLTP is similar to what we have observed from the analytical model in section 2. For SPECjbb, 12.5% replication shows the best for both 4MB and 8MB L2. The figure for Apache shows that the performance is better with replications as large as 50% for 4MB, but 37.5% for 8MB L2s. The seemly contradiction comes from the fact that L2 misses start to reduce drastically around 8M caches, as demonstrated in Fig. 7. We can also observe that the optimal replication levels match perfectly between the stack simulations and the real timing simulations. With respect to the average L2 access time, the stack results are within 2%-8% error margins.

C. **Private Caches without Replication**

Private caches sacrifice capacity for fast access time. It may be desirable to limit replications in the private caches. To understand the performance of the private L2 without replication, we run a stack simulation in which the creation of a replica causes the invalidation of the original copy.

Fig. 11 demonstrates the L2 access delays of the private caches without replication, shown as the ratio to those of the private caches with full replication, obtained from both the stack and timing simulations. As expected, with small 128KB and 256KB private caches per core, the average L2 access times without replication are about 5-17% lower than those with full replication for all the three workloads. This is because the benefit of the increased capacity more than compensates the loss of local accesses. With large 1MB or 2MB caches per core, the average L2 access time of the private caches without replication is 12-30% worse than the full-replication counterpart, suggesting that increasing local

---

**Figure 9. Ave. L2 access time ratio (private/shared)**

**Figure 10. Ave. L2 access time with different replication**
accesses is beneficial when enough L2 capacity is available. The stack simulation results follow this trend perfectly. They provide very accurate results with only 2-5% margin of error.

VI. RELATED WORK

Optimizing on-chip storage space on CMPs has been studied extensively [2, 17, 9, 22, 29, 8, 3, 24, 14, 13, 16]. The goal is to dynamically allocate data blocks for fast access without adversely increasing off-chip traffic due to the L2 misses. These studies must examine a wide-spectrum of the design space, which requires costly simulations.

There have been several techniques for speeding up cache simulations. Mattson, et al. [20] presents a stack algorithm to measure cache misses for multiple cache sizes in a single pass. For fast search through the stack, tree-based stack algorithms [4, 26] are proposed. Kim, et al. [15] provide a much faster simulation by maintaining the reuse distance counts only to a few potential cache sizes. All-associativity simulations [7, 2] and generalized forest simulations [12, 23] allow a single-pass simulation for variable set-associativities. Meanwhile, various prediction models have been proposed to provide quick cache performance estimation [1, 11, 10, 26, 5, 6]. They apply statistical models to analyze the stack reuse distances. But, it is generally difficult to model systems with complex real-time interactions among multiple processors. StatCache [5] estimates capacity misses using sparse sampling and static statistical analysis.

All above techniques target uniprocessor systems where there is no interference between multiple threads. Several works aim at modeling multiprocessor systems [27, 28, 7, 6]. StatCacheMP [6] extends StatCache to incorporate communication misses. However, it assumes a random replacement policy for the statistical model. Chandra, et al [7] propose three analytical models based on the L2 stack distance or circular sequence profile of each thread to predict inter-thread cache contentions on the CMP for multiprogrammed workloads that do not have interference with each other. Two other works [27, 28] pay attention only to miss ratios, update ratios, and invalidate ratios for multiprocessor caches. The proposed stack simulation method aims at the L2 caches on CMPs where the remote cache hits are an important performance metric. The single-pass stack simulator simulates both shared and private L2 caches, and projects the cache performance for various CMP cache organizations with different degrees of sharing.

VII. CONCLUSION

In this paper, we developed an abstract model for understanding the general performance behavior of data replication in CMP caches. The model showed that data replication could degrade cache performance without a sufficiently large capacity. We then used the global stack simulation for more detailed study on the issue of balancing accessibility and capacity for on-chip storage space on CMPs. With the stack simulation, we can explore a wide-spectrum of the cache design space in a single simulation pass, which otherwise requires multiple, costly simulations. We simulated the schemes of shared caches, shared caches with replication, private caches, and private caches without replication of various cache sizes, and generated the cache hits/misses and the average memory access time of each scheme. We verified the stack simulation results with the execution-driven simulations using a set of commercial multithreaded workloads and demonstrated that the single-pass stack simulation results can characterize the CMP cache performance with high accuracy. In addition, the stack simulation can accurately estimate the trends in the average L2 access time when data replicas are injected into the shared caches or when they are limited to the private caches, under different L2 sizes. Our results proved that the effectiveness of various techniques to optimize the CMP on-chip storage is closely related to the L2 size.

REFERENCES


