Modeling and Stack Simulation of CMP Cache Capacity and Accessibility

X. Shi, F. Su, J.-K. Peir, Y. Xia, and Z. Yang

Abstract— Performance tradeoffs between fast data access by local data replication and cache capacity maximization by global data sharing have been extensively studied for many-core Chip-Multiprocessors (CMPs). Costly simulations over a wide spectrum of the design space are generally required to gain insight for a sound design. To lower the cost, we develop an abstract model for understanding the performance impact of data replication on CMP caches. To overcome the lack of real-time interactions among multiple cores in the model, we further develop an efficient single-pass stack simulation to study the performance of CMP cache organizations with various degrees of data replication. The global stack logically incorporates a shared stack and per-core private stacks; shared/private reuse (stack) distances can be collected in a single-pass simulation. With the reuse distances, one can calculate the performance of CMP cache organizations with various degrees of data replication. We verify both the model and the stack simulation against execution-driven simulations with commercial multithreaded workloads. The results show that the abstract model provides accurate information about performance tradeoffs of data replication. The stack simulation accurately predicts the performance of various cache organizations with 2-9% error margins using only about 8% of the simulation time.

Index Terms— Cache memories, Chip-multiprocessors, Performance Modeling, Stack simulation.

1 Introduction

How to balance between fast data access and efficient use of the on-chip storage capacity has been studied extensively for CMPs [2, 18, 9, 24, 31, 8, 3, 26, 15, 14, 17]. 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 stored 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 storing the recently-accessed blocks in the local cache. However, multiple copies of a single block may exist in multiple caches, which decreases the effective cache capacity and increases off-chip memory traffic.

As a CMP memory hierarchy almost always includes small, private instruction/data L1 caches for fast accesses, the unsettled interesting issue is the organization of the on-die L2 cache. Recently, there have been several research papers [2, 18, 9, 24, 31, 8, 3] proposing variously combined private/shared L2-cache organizations following two general directions. The first is to organize the L2 as a shared cache for maximizing the effective capacity. To reduce the access time, a portion of the L2 can be set aside for replication [31]. The second is to organize the L2 as private caches for minimizing the access time. To achieve a higher effective capacity, data replications among multiple L2s are constrained and different private L2s can steal each other's capacity by block migration [24, 3, 8].

CMP cache studies must examine a wide spectrum of the design space to yield an accurate and comprehensive view of the cache design. Due to the lack of an efficient methodology for performance evaluation, near-sighted conclusions can potentially be drawn as a result of experiments with the design parameters that fall in a narrow range. Analytical models can provide quick performance estimation [1, 13]. However, they usually depend on statistical or other simplifying assumptions, and cannot accurately model systems with complex real-time interactions among multiple processors [7]. For fast simulation, the stack-based technique proposed in [21] 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 applicability to variable set-associativities [21, 4, 27, 13, 16, 25]. However, these stack simulation methods target only uniprocessor caches where the complex cache-coherence issue is not involved and the memory access delay hardly affects the order of memory requests. Extensions of the stack simulation to multiprocessors are reported in [29, 30]. Those studies focus on solving the problem of multiprocessor cache invalidations for general set-associative caches. However, the remote cache hit, an important measure on CMPs, is not considered. Moreover, 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 [23]. Four L2 cache organizations - shared, private, shared with data replica-

- *X. Shi is with the Google Inc., Mountain View, CA 94043. E-mail: xushi@cise.ufl.edu.*
- *J.-K. Peir and Y. Xia are with the Department of Computer Information and Science and Engineering, University of Florida, Gainesville, FL 32611. E-mail: {peir, yx1}@cise.ufl.edu.*
- *F. Su and Z. Yang are with the Nvidia Inc., Santa Clara, CA 95050. E-mail: {fsu, zhyang}@cise.ufl.edu.*

Manuscript received on July 30, 2008.

tion, and private without data replication - are studied. We focus on understanding the performance tradeoff between fast data access and cache capacity loss due to data replication. Both analytical modeling and fast stack-based simulation techniques are considered. The outline and contributions of our approach are as follows.

1) *Modeling Data Replication*: We first develop an analytical model to assess general performance effects of data replication. The model considers injecting 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 performance impact of the replicas. We also derive an expression to calculate the optimal degree of data replication. The results demonstrate that whether data replication helps or hurts cache performance are a function of the working set of the application and the available cache size. Existing CMP cache studies may have not examined a large enough design space and have overlooked important tradeoffs.

2) *Single-Pass Stack Simulation*: To overcome the limitations of the analytical model, we develop a single-pass stack simulation technique for more accurate performance assessment. The stack algorithm can handle interactions among multiple private caches. The technique can produce performance results of shared or private cache organizations with the invalidation-based coherence protocol. For instance, it can provide local/remote hit ratios and the effective cache sizes for a range of physical cache capacities.

3) *Performance Impact of Data Replication*: We demonstrate that we can use the 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 [31].

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

5) *Reduction of Simulation Time*: The total simulation times for the single-pass stack simulation and the individual execution-driven simulations are compared. For the four studied cache organizations, the stack simulation takes about 8% of the execution-driven simulation time.

The paper is organized as follows. Section 2 describes the analytical model. Section 3 introduces the CMP single-pass stack simulation. Section 4 describes the simulation methodology. This is followed by a comparison against execution-driven simulations in section 5. Further related work and the conclusion are given in Section 6 and Section 7.

2 Modeling Data Replication

We first develop an abstract model independent of private/shared organizations to evaluate the tradeoff between the access time and the miss ratio of CMP caches under various degrees of data replication. This model can provide a uniform understanding of the central issues in CMP caching that are present in most cache organizations. This study also highlights the importance of examining a wide range of system parameters in any CMP cache organization, which is time-consuming and costly. The model-based analysis and the stack simulation introduced later are complimentary approaches for studying CMP cache performance. The former makes simplifying assumptions but can provide high-level, panoramic views about the performance behavior of a broad class of cache organizations. It gives important insights and intuitive guidance for cache design. The latter provides more detailed, more accurate performance evaluation for each specific cache organization.

In Figure 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 computed by $\int_0^S f(x)dx$, which is equal to the area under the histogram curve from 0 to S. This well-known stack distance histogram can provide hit/miss information of all cache sizes with a fully-associative organization and the LRU replacement policy.

To model the performance impact of data replication, we consider injecting replicas into the cache. Note that regardless the cache organization, replicas help to improve the local hit ratio since they are created and stored close to the requesting cores, and hence, enable a short access time. On the other hand, having replicas reduces the effective capacity of the cache, and hence, increases cache misses. We need to compare the 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 in developing the abstract model, we do not make reference to any specific cache organization and management scheme. Moreover, the model does not state where precisely the replicas are stored, nor it intends to capture the cache coherence interactions. A simple ratio of data replication is used to derive extra cache misses vs. additional local hits, as will be described next.

We will compare the data replication scenario with the baseline case where all S blocks are distinct. On one hand, the cache misses are increased by $\int_{S-R}^{S} f(x)dx$, since the total number of hits is now $\int_0^{S-R} f(x)dx$. On the other hand, the replicas help to improve the local hits. Assuming data references are evenly distributed among all cache blocks,

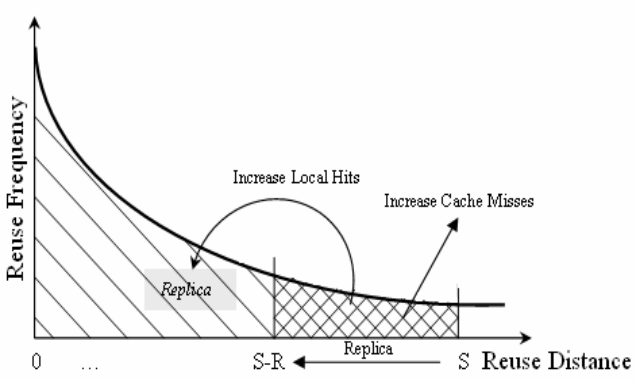


Fig. 1. Cache performance impact with replica.

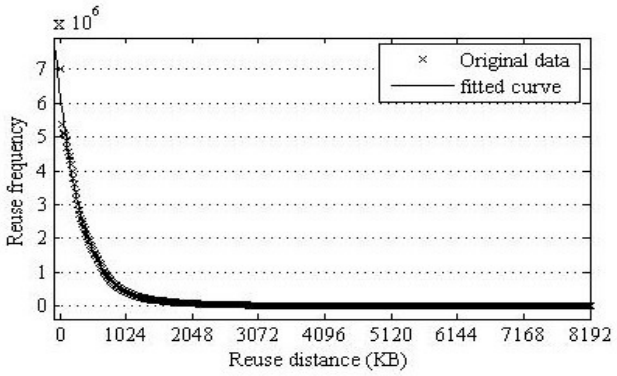


Fig. 2. Curve fitting of reuse distance histogram of OLTP.

within $\int_0^{S-R} f(x)dx$ hits, a fraction *R/S* of them are targeting the replicas. However, depending on the specific cache organization, not all accesses to the replicas result in local hits. For example, a requesting core may find a replica in the local cache of another remote core, resulting in a longer remote hit. We assume that a fraction of accesses to the replicas are actually local hits and denote the fraction by *L*. Therefore, compared with the baseline case, the total change of memory access cycles due to the creation of *R* replicas can be calculated by:

$$P_m \times \int_{S-R}^{S} f(x)\,dx \; - \; G_l \times \frac{R}{S} \times L \times \int_0^{S-R} f(x)\,dx \qquad (1)$$

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 accesses $\int_0^{\infty} f(x)dx$, the average delta of memory access cycles is equal to:

$$\left(P_m \times \int_{S-R}^{S} f(x)dx - G_l \times \frac{R}{S} \times L \times \int_0^{S-R} f(x)dx\right) \Big/ \int_0^{\infty} f(x)dx \qquad (2)$$

Now the key is to obtain the reuse distance histogram *f(x)* to fit into this equation. In [12], an analytical cache miss model was developed based on the power law of block reuses. Instead of deriving *f(x)* analytically; we use the curve-fitting tool of Matlab [20] to obtain the best fit *f(x)*. Note that the curve-fitting approach is capable of handling a wider array of workloads, such as those with cyclical program behavior where it is difficult to apply a pure analytical approach [12]. We conduct an experiment using an OLTP workload [22] and collect its generic reuse distance histogram. With the curve-fitting tool, we obtain the equation (3):

$$f(x) = A \times e^{-Bx} \qquad (3)$$

where $A = 6.084 \times 10^6$, and $B = 2.658 \times 10^{-3}$. This is shown in Figure 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 delta of the average memory cycles as:

$$P_m \times \left(e^{-B(S-R)} - e^{-BS}\right) - G_l \times \frac{R}{S} \times L \times \left(1 - e^{-B(S-R)}\right). \qquad (4)$$

Equation (4) provides the change in L2 access time as a function of the cache space being occupied by the replicas. In Figure 3, we plot this delta of the average 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_l=15, P_m= 400, and we vary *L* with 0.25, 0.5 and 0.75 for each cache size. 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 instance, when *L* = 0.5, the results indicate no replication provides the shortest average memory access time for a 2MB L2 cache, while for larger 4MB and 8MB L2 caches, allocating 40% and 68% of the cache for the replicas has the smallest access time. These results are consistent with the reuse histogram curve shown in Figure 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. Increasing *L* favors data replication slightly. For instance, for a 4MB cache, allocating 34%, 40%, 44% of the cache for the replicas achieves the best performance improvement of about 1, 3, and 5 cycles on the average memory access time for *L* = 0.25, 0.5 and 0.75 respectively. The performance improvement with data replication would be more significant when G_l increases.

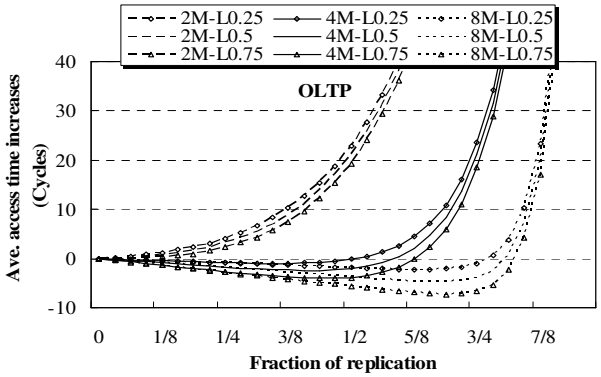


Fig. 3. Performance with replicas for different cache sizes.

The general behavior due to data replication is consistent with the detailed simulation results given in Section 5. Note that the fraction of replicas cannot reach 100% unless the entire cache is occupied by a single block. Therefore, in Figure 3, the average access time increase is not meaningful when the fraction of replicas is approaching the cache size.

From equation (4), we can also derive the optimal frac-

tion of replication under different cache sizes for OLTP based on its reuse distance histogram *f(x)*. The number of replicas *R* is optimal when the derivative of equation (4) with respect to *R* is zero. We can obtain the following equation:

$$(\frac{BSP_m}{G_l L}+1)e^{-BS} = e^{-BR} + e^{-BS}BR. \quad (5)$$

Since $e^{-BS}BR$ is far less than e^{-BR} when *R* is smaller than *S*, we can approximate the above equation as $(\frac{BSP_m}{G_l L}+1)e^{-BS} = e^{-BR}$. Now, solving equation (5), we can get the optimal replication *R* as:

$$R = \frac{1}{B}\ln\frac{e^{BS}}{(\frac{BSP_m}{G_l L}+1)}. \quad (6)$$

By plugging the aforementioned OLTP parameters into equation (6), we get the optimal factions of replication under cache sizes from 2MB to 8MB and *L* from 0.25 to 0.75, as shown in Figure 4. For the 2MB caches, 0%, 0% and 3.2% of replication are the best for *L* = 0.25, 0.5, 0.75; the optimal fractions are 35%, 41%, 45% for the 4MB caches, and 64%, 67%, 69% for the 8MB caches, respectively. We also run the same experiment for two other workloads, Apache and SPECjbb. The same behavior can be observed for both Apache and SPECjbb, i.e. larger caches favor more replications. For example, with *L* = 0.5, allocating 13%, 50%, 72% space for replicas has the best performance for Apache, and 28%, 59%, 78% for SPECjbb. Furthermore, increasing *L* also favors more replication. Due to its smaller working set, SPECjbb benefits the most with data replication among the three workloads.

The analysis shows that it is essential to study a set of representative workloads with different 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.

3 Single-Pass Stack Simulation

Although our analytical model can provide understanding of the general performance trend, its inability to model sufficiently detailed interactions among multiple cores limits its capability for accurate performance prediction. To remedy this problem, we develop a single-pass, global-stack based simulation method for studying the CMP caches.

In our stack simulation, a single global stack is built to record the history of requested block addresses from all cores. Multiple double-link lists are established in the global stack for simulating the generic stack algorithm for both a *shared* and per-core *private* caches. Figure 5 sketches an entry of the global stack data structure; each entry records one memory reference. 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. We maintain one logical private stack for each core, which is organized as a doubly linked list. (Note that for simplicity, we only show one private list in Figure 5.) Each global stack entry is linked by the *Private prev* and *Private next* pointers in exactly one of the logical private stacks determined by the core-id. We also maintain one logical shared stack also as a doubly linked list. The global stack entries are linked together for the shared stack by the *Shared prev* and *Shared next* pointers. Each entry may or may not be in the logical shared stack depending on the recency of the reference, since only a single copy of a block address exists in the shared cache. In addition, a block address-based *hash* list is also established in the global stack for fast searches.

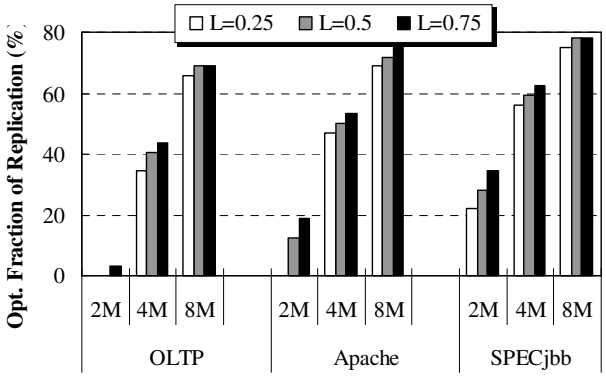


Fig. 4. Optimal fraction of replication.

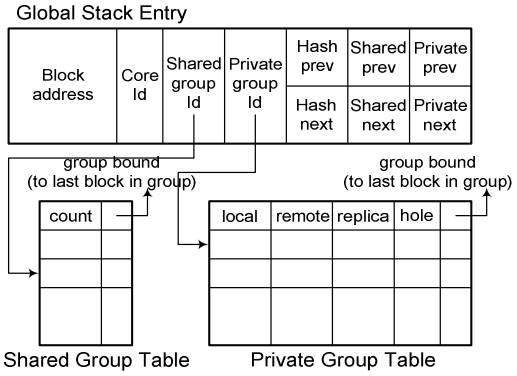


Fig. 5. Global stack organization.

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*, each consisting of multiple cache blocks, for fast search during the stack simulation and for easy calculations of cache hits under various interesting cache sizes after the simulation [16]. 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 blocks for the first three groups, respectively, assuming the block size is 64B. The first group with 256 blocks is to simulate the 16KB cache. The first two groups of (256 + 256) blocks are to simulate the 32KB cache and so on. 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 useful 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 for both shared and private caches.

3.1 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-appearance of a block in the global stack should be in the shared list since there is no replication in a shared cache. A first-appearance 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-appearance blocks from MRU to LRU. Figure 6 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.

1. 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.
2. B2 is removed from the shared list, and B1 is inserted at the top of the shared list.
3. 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 to the group where B2 was previously located.
4. 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.
5. After the simulation, the total number of cache hits for a shared cache that include exactly the first m groups is the sum of all shared reuse counters from group *1* to group *m*.

3.2 Private Caches

The construction and update of the private lists are essentially the same as those of the shared list, except that accesses from the same core are linked 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. Figure 7 draws the contents of the four private lists and the private group table, where we extend the memory sequence (Figure 6) 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.

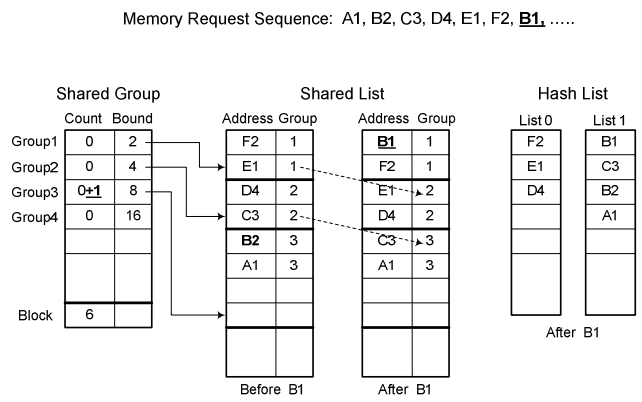


Fig. 6. Shared cache example.

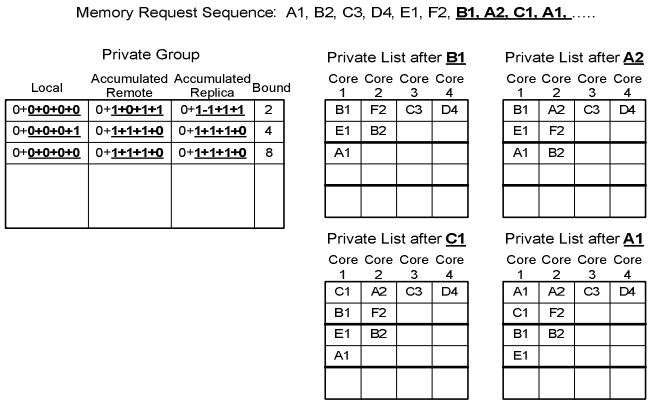


Fig. 7. Private cache example.

Counting the remote hits is somewhat 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 group L and

the minimum remote group that contains the block 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 for a cache with exactly *m* groups. 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, 31, 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 Figure 7, 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. The hole-count for each group avoids searching for existing holes.

4 Simulation Methodology

We use the full-system Virtutech Simics 2.2 simulator [19] 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 simulations, where detailed cache models with proper access latencies are inserted. In those simulations, 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 used to maintain data co-

TABLE 1
Simulation Parameters

CMP: 8 cores, 3.2GHz, 128 entry ROB
Branch predictor: g-share, 64K, 4K BTB
Branch misprediction penalty: 10 cycles
L1-I: 32KB, 4-way, 64B line, MESI
L1-D: 32KB, 4-way, 64B line, MESI
L1-I/L1-D latency: 0/2 cycles
L2: 16-way, 64B line, MESI
Private L2 size (KB): 128/256/512/1024/2048 per core
Private L2 local/remote latency: 15/30 cycles
Shared L2: 8 banks, 1 local/7 remote
Shared L2 size (MB): 1/2/4/8/16
Shared L2 local/remote latency: 15/30 cycles
Memory latency: 400 cycles
Stack: 16KB / group, 1024 groups (16MB maximum)

TABLE 2
Workload Descriptions

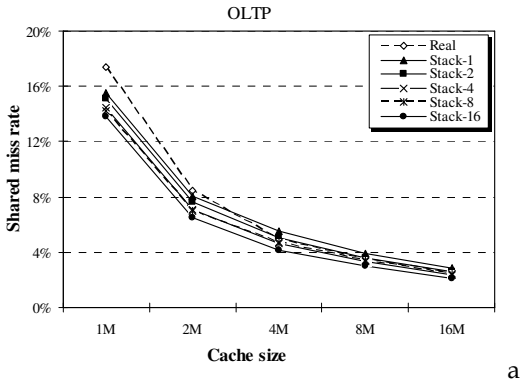
OLTP (Online Transaction Processing): It is built upon the OSDL-DBT-2 [22] and MySQL database server 5.0. We build a 1GB, 10-warehouse database. To reduce the database disk activity, we increase the size of the MySQL buffer pool to 512MB. We further stress the system by simulating 128 users with no keying and thinking time. We simulate 1024 transactions after bypassing 2000 transactions and warming up caches (or stack) for another 256 transactions.
Apache (Static web server): We run apache 2.2 as the web server, and use Surge to generate web requests from a 10,000 file, about 200MB repository. We simulate 8 clients with 50 threads per client. We collect statistics for 8192 transactions after bypassing 2500 requests and warming up for 2048 transactions.
SPECjbb (java server): We simulate 8 warehouses. We first fast-forward 100,000 transactions. Then we simulate 20480 transactions after warming up the structures for 4096 transactions.

herence among 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.

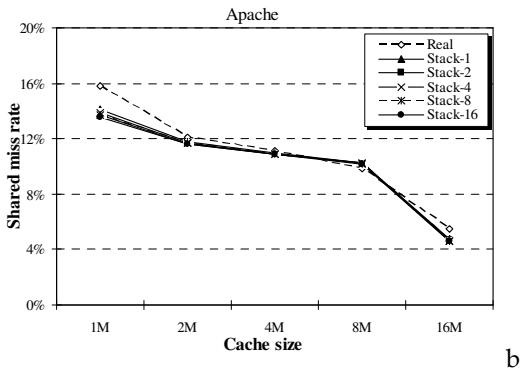
We use three multithreaded commercial workloads, OLTP, Apache, and SPECjbb, as described in Table 2. We consider the variability of these multithreaded workloads by running multiple simulations for each configuration of each workload and inserting small random noises (perturbations) in the memory system timing for each run.

5 Evaluation and Validation

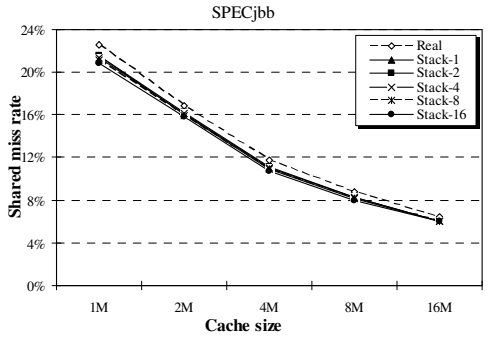
The accuracy of the CMP memory performance projection can be assessed from two different angles, the accuracy of predicting individual performance metrics, and the accuracy of predicting general cache behavior. By verifying the results against the execution-driven simulation, we demonstrate that the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



a



b



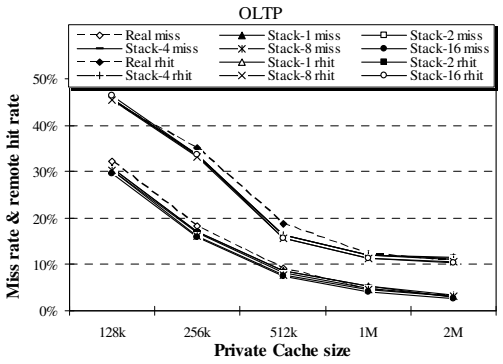
c

Fig. 8. Miss ratios for shared caches.

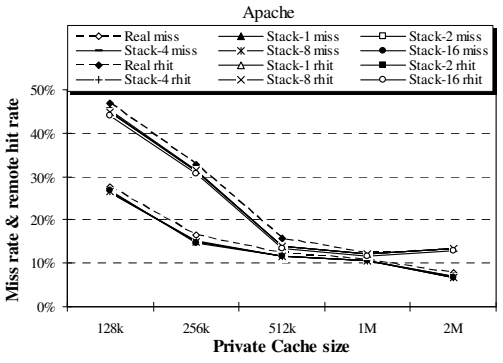
One inherent difficulty of stack simulation is its inability to insert accurate timing delays for variable L2 cache sizes. The fluctuation in memory delays may alter the sequence of memory accesses among multiple processors. We try a simple approach to insert memory delays based on a single discrete cache size. In the stack simulation, we inserted memory delays based on five cache sizes 1MB, 2MB, 4MB, 8MB, and 16MB, 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 the selected discrete cache size.

5.1 Hits/Misses for Shared and Private L2 Caches

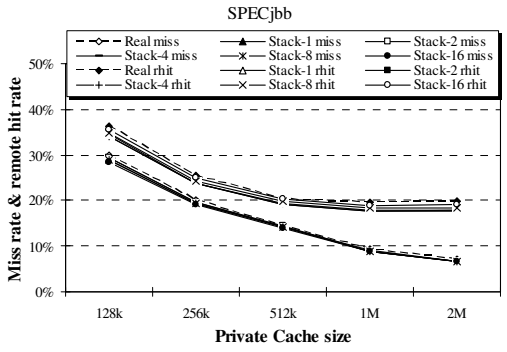
Figure 8 shows the projected and real miss rates for shared caches, where "real" represents the results from individual execution-driven simulations. In general, the stack results follow the execution-driven results closely. For OLTP, stack-2 shows only about 5-6% average error.



a

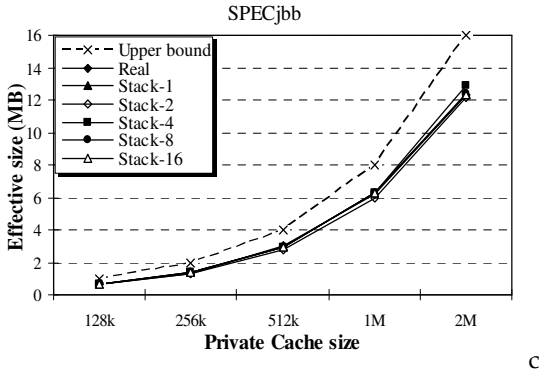


b



c

Fig. 9. Miss ratio, remote hit ratio for private caches.



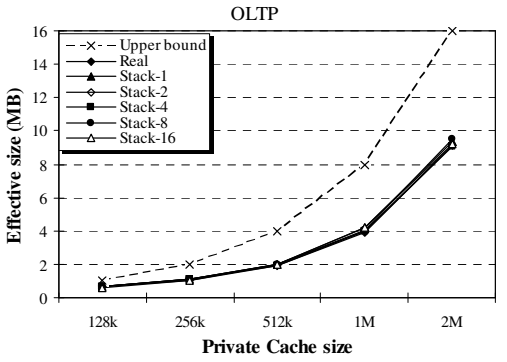
c

Fig. 10. Average effective size for private caches.

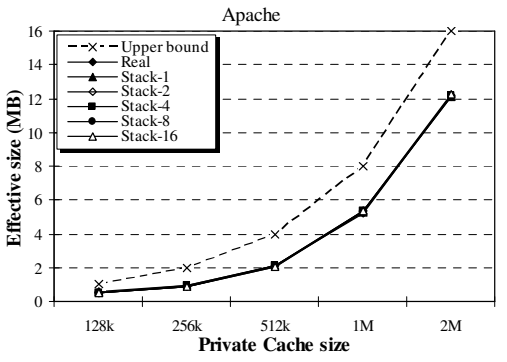
For Apache and SPECjbb, the difference among different delay insertions is less apparent. The stack results predict the miss ratios with about 2-6% error, except for Apache with a small 1MB cache.

Two major factors affect the accuracy of the stack results. 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 more apparent when the cache size is small, due to more conflict misses. The issue can be resolved by more complicated set-associative stack simulations [21, 13]. For simplicity, we keep the stack fully-associative. More sensitivity studies are also helpful to evaluate L2 with smaller set associativity. The other factor is inaccurate delay insertions. For example, in the stack-1 simulation of OLTP, a cache miss latency is inserted whenever the reuse distance is longer than 1MB. Such a cache miss delay is inserted wrongly for caches larger than the 1MB. These 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 smaller caches mistakenly inserts hit latency, instead of miss latency, for accesses with reuse distance from the corresponding cache size to 16MB, causing less OS interferences, thus less misses. In this case, both the full cache associativity and the delay insertion lead to underestimate of the real misses, which makes the stack-16 simulation the most inaccurate.

For private caches, Figure 9 shows the overall misses and the remote hits. 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 Figure 8. We can make three important observations. First, comparing with the shared cache, the simula



a



b

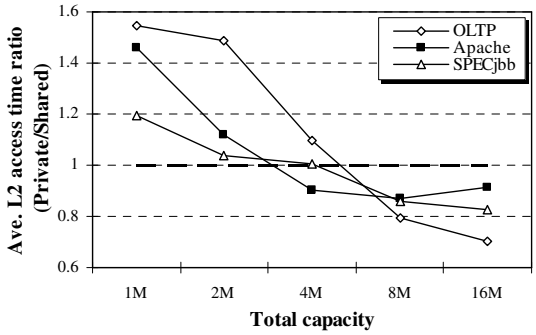


Fig. 11. Average L2 access time ratio (private / shared Caches).

tion 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 private cache sizes from 128KB to 1MB. 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. Second, the estimated miss and remote hit rates from the stack simulation match closely to the results from the execution-driven simulations, with less than 10% margin of errors.

We also simulate the effective capacity for the private-caches as shown in Figure 10. The effective cache size is the average over the entire simulation period. In general, the private cache reduces 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. We also observe that the estimated capacity from the stack simulation is almost identical to the result from the execution-driven results. Note that due to its higher accuracy, we use the stack-2 simulation in the following discussion.

For comparison, Figure 11 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 large, the private cache becomes better. With larger caches, the difference of L2 misses diminishes, but the private L2s have much more local hits, which makes the average L2 access time shorter.

5.2 Shared Caches with Replication

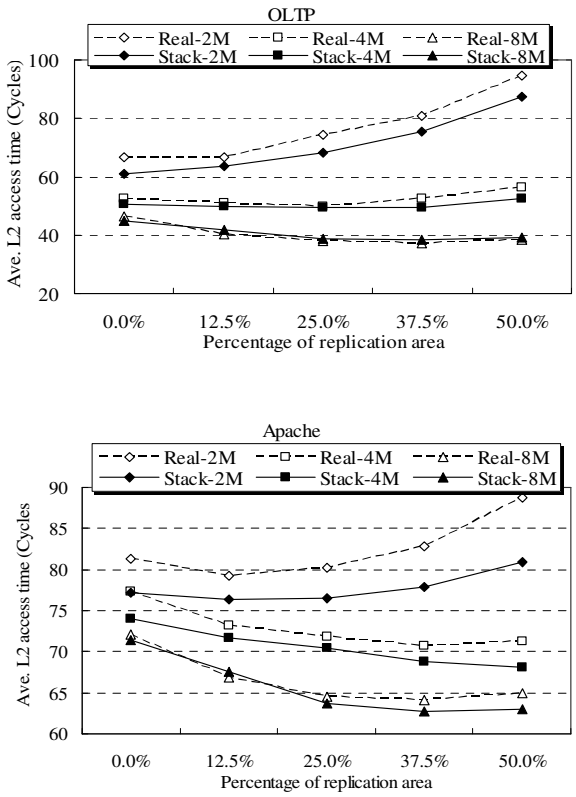
To balance accessibility and capacity, victim-replication [31] creates a dynamic L1 victim cache for each core in the local slice of the L2 to trade capacity for fast local access. In this section, we show a quick estimation of the performance of a static victim-replication scheme. We allocate 0% to 50% of the L2 capacity as L1 victim caches with variable L2 sizes from 2MB to 8MB. For performance comparison, we use the average memory access time, which is calculated based on the local hits to victim caches, the hits to shared portion of L2, and L2 misses.

The average memory access time of the static victim replication can be derived directly from the results of the stack simulation described in the previous sections. Assuming the inclusion property is enforced between the shared potion of the L2 and the victim potion plus the L1. Suppose the L1 and L2 sizes are denoted by *CL1*, and *CL2*, *r* is the percentage of the L2 allocated for the victim cache, and *n* is the number of the cores. Then, each victim-cache size is equal to *(r*CL2)/n*, and the remaining shared portion is equal to *(1-r)*CL2*. 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: *CL1+(r*CL2)/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 total number of L2 hits (including the victim portion) and L2 misses can be calculated from the shared stack with the size *(1-r)*CL2*. Finally, the hit to the shared portion of L2 can be calculated by subtracting the hits to the victim from the total L2 hits.

Figure 12 demonstrates the average L2 access time with static victim replication. Generally, large caches favor more replications as expected. 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. This general performance behavior with respect to data replication is consistent with what we have observed from the analytical model in section 2. However, the analytical model without cache invalidations should apply lower L for the optimal replication level.

For SPECjbb, 12.5% replication shows the best performance for both 4MB and 8MB L2. For Apache, the best performance is with replications as large as 50% except for 8MB L2s. This seemly contradiction comes from the fact that the number L2 misses is reduced drastically around 8M caches as shown in Figure 8. Providing larger effective capacity for 8M caches is more beneficial than adding replicas. We can also observe that the optimal replication levels match perfectly between the stack simulations and the execution-driven simulations. With respect to the average L2 access time, the stack results are within 2%-8% error margins.

In the above studies, we assume a fixed L1 size of 32KB for the instruction and the data caches. Since the L1 cache size plays an important role in victim replications at the L2 level, we run two more stack simulations with 16KB and 64KB L1 sizes. Figure 13 shows the breakdown of L1 hit, L2 local hit, L2 remote hit and L2 miss with different L1 sizes from 16KB to 64KB and different replication levels. Since the general behavior is similar for all three workloads, we only show the results for SPECjbb with an 8MB L2 cache. Generally, the L1 size does not significantly changed the replication behaviors. For this configuration, 50% of victim space is the best for all of the three L1 sizes. However, with smaller L1 caches, victim replication of various degrees has larger fraction of L2 local hits, potentially more performance improvement. This is due to the fact that the number of L1 misses is much more for smaller L1 cache. Hence, victim replication can generate more local L2 hits.



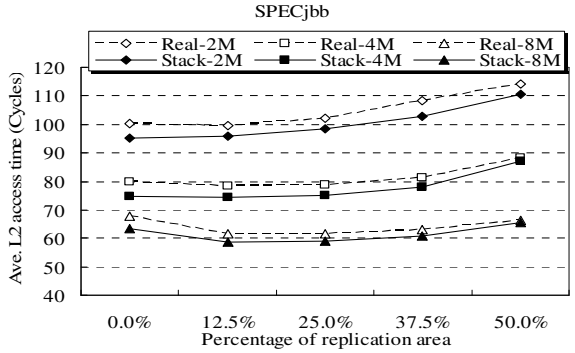


Fig. 12. Average L2 access time with different replication.

5.3 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 impact of the private L2 without replication, we run a separate stack simulation in which the creation of a replica causes the invalidation of the original copy.

Figure 14 demonstrates the L2 access delays of the private caches without replication, shown as the ratio to those of the private caches with full replication. 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 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.

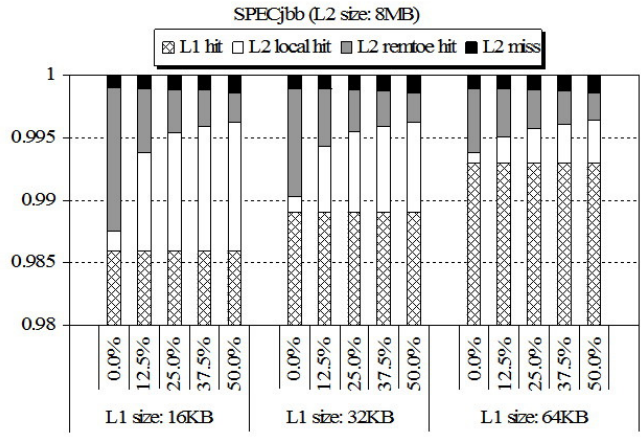
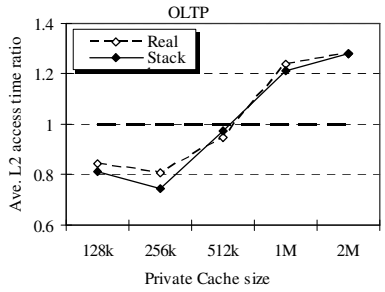


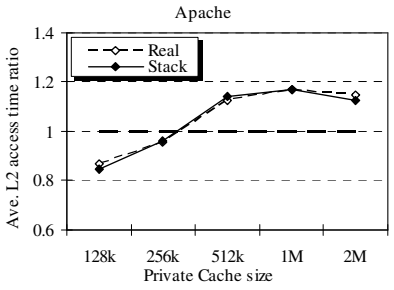
Fig. 13. Effects of L1 size on replication for SPECjbb with 8MB L2.

5.4 Simulation Time Comparison

We run the full-system Virtutech Simics 2.2 simulator [19] to simulate an 8-core CMP system with Linux 9.0 and x86 ISA on Intel Xeon 3.2 GHz 2-way SMP. The simulation time of each stack or execution-driven simulation is measured on a dedicated system without other interference. A timer was inserted at the beginning and the end of each run to calculate the total execution time. Detailed descriptions of the three workloads have been given in Table 2. In the single-pass stack simulation, each stack is partitioned into 16KB groups with a total of 1024 groups for the 16MB cache. This small 16KB groups are necessary in order to study shared caches with variable percentage of replication areas as shown in Figure 12. The stack simulation time can be further reduced for cache organizations that only require a few large groups.

Table 3 summaries the simulation times for the stack and the execution-driven simulations using the three workloads. For each workload, two stack simulation runs are needed. One run is for producing the results for shared caches, private caches, and shared caches with replication, and the other run is for the private L2 without replication. In execution-driven simulations, it requires a separate run for each cache size resulting in five runs for each cache organization. In studying the shared cache with replication, five separate runs are needed for each cache size in order to simulate five different replication percentages. No separate stack simulation is required for the shared cache with replication. Similarly, no separate execution-driven simulation is needed for shared caches with 0% area for data replication. Therefore, we need 20 runs for the shared cache with replication for the execution-driven simulation. The total number of simulation runs is also summarized in Table 3. The total stack simulation time is measured to be about 4751 minutes, while the execution-driven simulation takes 58016 minutes, a ratio of over 12 times. This gap can be much wider if more cache organizations and sizes are simulated.





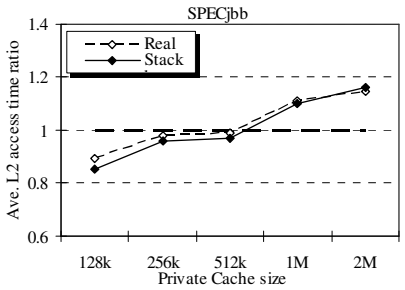


Fig. 14. Average L2 access time ratio of private caches without replication.

TABLE 3
SIMULATION TIMES COMPARISON (IN MINUTES)

Measurement	Workload	Stack	Execution-Driven
Shared / Private (Section 5.1)	OLTP	1 Run: 835	(5+5) Runs: 6252
	Apache	1 Run: 901	(5+5) Runs: 6319
	SPECjbb	1 Run: 582	(5+5) Runs: 4220
Shared with replication (Section 5.2)	OLTP	0 Run: 0	20 Runs: 11976
	Apache	0 Run: 0	20 Runs: 12211
	SPECjbb	0 Run: 0	20 Runs: 8210
Private no replication (Section 5.3)	OLTP	1 Run: 872	5 Runs: 3257
	Apache	1 Run: 948	5 Runs: 3372
	SPECjbb	1 Run: 613	5 Runs: 2199
Total		4751	58016

6 RELATED WORK

Optimizing on-chip storage space on CMPs has been studied extensively [2, 18, 9, 24, 31, 8, 3, 26, 15, 14, 17]. The goal is to dynamically allocate data blocks for fast access without adversely increasing off-chip traffic due to the L2 misses. With many CMP cache organizations, 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. [21] 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, 28] are proposed. Kim, et al. [16] provides 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 [13, 25] 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, 28, 5, 6, 12]. They apply statistical models to analyze the stack reuse distances. But, it is generally difficult to precisely 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 running on different processors. Several works aim at modeling multiprocessor systems [29, 30, 7, 6]. StatCacheMP [6] extends StatCache to incorporate communication misses. 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 [29, 30] extends stack simulations to multiprocessors. However, they pay attention only to miss ratios, update ratios, and invalidate ratios. The proposed single-pass 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 creates a global stack to simulate both shared and private L2 caches. The results can be used to project the cache performance for various CMP shared, private, and a combination of both cache organizations with different degrees of data sharing.

7 CONCLUSION

In this paper, we developed an abstract analytical model for understanding the general performance behavior of data replication on CMP caches. The model showed that data replication could degrade cache performance without a sufficiently large cache capacity. We then developed a global stack simulation method 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 evaluate a wide-spectrum of the cache design space in a single simulation pass. Based on the stack simulation results, we can estimate performance of regular shared / private caches, shared caches with data replication, and private caches without data replication of various cache sizes. We verified the modeling and stack simulation results against detailed execution-driven simulations using commercial multithreaded workloads. We showed that the analytical model and the single-pass stack simulation can characterize the CMP cache performance with high accuracy. Our results also demonstrated that the effectiveness of various techniques to optimize the CMP on-chip storage is closely related to the total L2 cache size.

ACKNOWLEDGMENT

This work was supported in part by NSF grant EIA-0073473 and by research and equipment donations from Intel Corp. We also thank anonymous referees for their helpful comments.

REFERENCES

[1] A. Agarwal, M. Horowitz and J. Hennessy, "An Analytical Cache Model," *ACM Transactions on Computer Systems, Vol. 7, No. 2*, May 1989, pp. 184-215.

[2] B. Beckmann and D. Wood, "Managing Wire Delay in Large Chip-Multiprocessor Caches," *Proc. of 37th Int'l Symp. on Microarchitecture*, Dec. 2004, pp. 319-330.

[3] B. M. Beckmann, M. R. Marty, and D. A. Wood. "ASR: Adaptive Selective Replication for CMP Caches," *Proc. of the 39th Int'l Symp. on Microarchitecture*, Dec. 2006.

[4] B. T. Bennett and V. J. Kruskal, "LRU Stack Processing," *IBM journal of R & D*, July 1975, pp. 353-357.

[5] E. Berg and E. Hagersten, "StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis," *Proc. of Int'l Symp. on Performance Analysis of Systems and Software*, March 2004.

[6] E. Berg, H. Zeffer and E. Hagersten, "A Statistical Multiprocessor Cache Model," *Proc. of Int'l Symp. on Performance Analysis of Systems and Software*, March 2006.

[7] D. Chandra, F. Guo, S. Kim and Y. Solihin, "Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture", *Proc. of 11th Int'l Symp. on High Performance Computer Architecture*, Feb. 2005, pp. 340-351.

[8] J. Chang and G. Sohi, "Cooperative Caching for Chip Multiprocessors," *Proc. of 33rd Int'l Symp. on Computer Architecture*, June 2006.

[9] Z. Chishti, M. D. Powell and T. N. Vijaykumar, "Optimizing Replication, Communication, and Capacity Allocation in CMPs," *Proc. of 32nd Int'l Symp. on Computer Architecture*, June 2005.

[10] G. Edwards, S. Devadas, and L. Rudolph, "Analytical Cache Models with Applications to Cache Partitioning," *Proc.of 15th Int'l Conf. on Supercomputing*, June 2001, pp. 1-12.

[11] B. Fraguela, R. Doallo, and E. Zapata, "Automatic Analytical Modeling for the Estimation of Cache Misses," *Proc. of Int'l Conf. on Parallel Architectures and Compilation Techniques*, Sep. 1999.

[12] A. Hartstein, V. Srinivasan, T. R. Puzak, and P.G. Emma, "On the Nature of Cache Miss Behavior: Is it sqrt(2)?" *Journal of Instruction-Level Parallelism 10 (2008)*, pp.1-22.

[13] M. Hill and J. Smith, "Evaluating Associativity in CPU Caches", *IEEE Transactions on Computers*, Dec. 1989, pp. 1612-1630.

[14] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger and S. W. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," *Proc. of 19th Int'l Conf. on Supercomputing*, June, 2005.

[15] C. Kim, D. Burger, and S. Keckler, "An Adaptive Non-uniform Cache Structure for Wire-delay Dominated On-chip Caches," *Proc. of 10th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[16] Y. H. Kim, M. D. Hill and D. A. Wood, "Implementing Stack Simulation for Highly-associative Memories," *Proc. of 1991 SIGMETRICS conf. on Measurement and Modeling of Computer Systems*, May 1991, pp. 212-213.

[17] R. Kumar, V. Zyuban, and D. M. Tullsen. "Interconnections in Multi-core Architectures: Understanding Mechanisms, Overhead and Scaling," *Proc. of 32nd Int'l Sump. on Computer Architecture*, June 2005.

[18] C. Liu, A. Sivasubramaniam and M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," *Proc. of 10th Int'l Symp. on High Performance Computer Architecture*, Feb. 2004, pp. 176-185.

[19] P. S. Magnusson et al. "Simics: A Full System Simulation Platform," *IEEE Computer*, Feb. 2002, pp. 50-58.

[20] Matlab, *http://www.mathworks.com/products/matlab/*.

[21] R. Mattson, J. Gecsei, D. Slutz, and I. Traiger, "Evaluation Techniques and Storage Hierarchies," *IBM Systems Journal, 9, 1970*, pp. 78-117.

[22] Open Source Development Labs. Open source development labs database test 2. *http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-2/*.

[23] X. Shi, F. Su, J. Peir, Y. Xia, and Z. Yang, "CMP Cache Performance Projection: Accessibility vs. Capacity," *Workshop on Design, Architecture and Simulation of Chip Multi-Processors (dasCMP2006), in conjunction with the 39th Annual International Symposium on Microarchitecture*, Dec 2006.

[24] E. Speight, H. Shafi, L. Zhang and R. Rajamony, "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors," *Proc. of 32nd Int'l Symp. on Computer Architecture, June 2005*, pp. 346-356.

[25] R. A. Sugumar and S. G. Abraham, "Set-associative Cache Simulation using Generalized Binomial Trees," *ACM Transactions on Computer Systems, Vol. 13, No. 1*, Feb. 1995, pp. 32-56.

[26] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic Partitioning of Shared Cache Memory," *The Journal of Supercomputing, 28(1)*, 2004, pp. 7-26.

[27] J. G. Thompson, "Efficient Analysis of Caching Systems," *Computer Science Division Technical Report UCB/Computer Science Dept. 87/374, University of California, Berkeley*, October 1987.

[28] X. Vera and J. Xue, "Let's Study Whole-Program Cache Behavior Analytically," *Proc. of 8th Int'l Symp. on High Performance Computer Architecture*, Feb. 2002.

[29] C. E. Wu, Y. Hsu, Y. Liu, "Efficient Stack Simulation for Shared Memory Set-Associative Multiprocessor Caches," *Proc. of 1993 Int'l Conf. on Parallel Processing*, Aug. 1993.

[30] Wu, Y. and Muntz, R. 1995, "Stack Evaluation of Arbitrary Set-Associative Multiprocessor Caches," *IEEE Transactions on Parallel and Distributed Systems*, Sep. 1995, pp. 930-942.

[31] M. Zhang, and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," *Proc. of 32nd Int'l Symp. on Computer Architecture*, June 2005, pp. 336-345.