A Study of Cache Organizations for Chip-Multiprocessors

Shatrugna Sadhu, Hebatallah Saadeldeen
{ssadhu,heba}@cs.ucsb.edu
Department of Computer Science
University of California, Santa Barbara

Abstract—The organization of on-chip storage for chip multiprocessor has a significant impact on its performance. The rapid increase in the number of processing cores, the likely scaling of the working set sizes of future workloads, and the increasing costs of off-chip misses make the organization of such limited on-chip memory very crucial for achieving better performance. Thus, the goal of cache paradigms in CMPs is to minimize the number of off-chip accesses through various organizations of L2 cache. The main challenge for these organizations is the increase in the wire delays as the size of the cache increases and thus limiting the benefits achieved by decreasing the number of misses. Therefore, recent research focuses on keeping data close to the processor that frequently access it in order to reduce the L2 hit latency. In this paper we provide survey of different cache organizations for CMPs. We then implemented one of these organizations, Shared Processor-Based Split L2 proposed in [4], and evaluated it against basic organizations (private and shared L2) for database tpc-w workload. The experimental results show that split shared L2 cache outperforms both private and shared L2 caches for both hit rate as well as time spent on memory stalls. This suggests that having a limited degree of sharing; some point between private and shared cache organizations; can help improve the performance of database workloads.

I. INTRODUCTION

Assuming that power ceilings do not limit continued CMP integration, future chip multiprocessors (CMPs) will continue to have large number of processors along with larger caches which will require innovative on-chip memory hierarchy design. While level one caches will likely remain private to each processor, level two caches will be the key design decision. L2 caches could be private to each processor, shared among all processors or any point in between.

With the emergence of the CMP design paradigms, multiple processors and large caches have been integrated on a single chip. The CMP makes it possible to execute multiple processes/threads simultaneously and exploit process/thread-level parallelism. The CMP architecture has been explored and studied in the context of a wide range of applications, such as commercial transactional server, network devices, personal computers, and embedded systems, etc. Obviously, parallel multiprocessor computing and high capacity caches dramatically improve system performance. Therefore, optimal design of cache architecture plays an important rule in improving the system performance of CMP.

There are tradeoffs in deciding the degree of sharing among processors. Increasing the degree of sharing among processors will reduce the number of on-chip memory access due to two reasons. First, it will reduce the number of shared copies of a single line existing on-chip, since each line maps to only one place in shared caches. Second, it tolerates imbalances among the sharers' working sets. However, increasing the sharing degree will lead to longer hit latencies due to the larger size of L2 cache, longer wire delays and increased bandwidth requirements. In addition, more L1 caches sharing L2 cache makes the maintenance of L1 caches more expensive. On the other hand, decreasing the degree of sharing among processors leads to lower hit latencies. However, it can be disadvantageous if the load among different processors is not balanced as one of the CPUs could suffer from capacity misses while the other CPUs have unused cache space. In addition, maintaining L2 coherence is expensive.

To reduce the access latency while keeping lower miss rate as in shared caches, a number of cache designs have been investigated recently. Most of them start with a shared L2 cache and attempt to keep data closer to the processor that frequently uses them. Keeping data close to the consuming processor reduces the hit latency for L2. Some of these papers use data migration and replication to further optimize the performance.

In this paper, we give a survey about various cache organizations for CMPs as well as study the effect of a commercial database workload on three specific cache paradigms to understand the tradeoffs better. We implemented Shared Processor-Based Split L2 cache organization and compared it to two basic organizations (private L2 cache, and shared L2 cache). We evaluate and compare the number of L2 misses as well as the total time spent on memory stalls for each of the three organizations.

This paper is organized as follows: We will start by explaining in section 2 the current design paradigms in CMP cache architectures based on Non-Uniform Cache Access organization and further improvements on the same, section 3...
will cover a survey of some of the other CMP cache design paradigms which are hybrid of some sorts of some of the existing organizations. In section 4 we show how one of the modern DBMS benchmarks: tpc-w performs under three different CMP cache architectures. We finally conclude in section 5.

II. NUCA BASED CACHE PARADIGMS

In basic uniform cache design architectures like the one shown in the Figure 1a, each cache is formed of multiple bank but they are not inter-connected. The banks within the cache which are further away from the processor will have higher access latency compared to the ones nearer to the processor due to wire-delay. NUCA (Non-Uniform Cache Access) [6] designs solve this on-chip wire delay problem by embedding an inter-connect network into the cache itself. As shown in Figure 1b, the cache is partitioned into banks, and each bank has a certain access latency based on its distance from the processor. Accordingly the more frequent data accesses are placed into the banks closer to the processor.

One thing also to be observed in case of uniformly cached architectures is the fact that each line in lower level will always have a second copy leading to consumption of extra space. In case of NUCA the mapping of data into banks is statically (S-NUCA) or dynamically (D-NUCA) determined, based on the block index, and thus can reside in only one bank of the cache.

![Figure 1: (a) UCA (b) NUCA](image)

The core concept behind the design of NUCA which makes it one of the most favorable base architecture for forming CMP cache hierarchy lies in the fact that they are highly flexible and scalable. A natural organization in CMP places multiple processors and an array of cache banks on a single die. As the workload changes, NUCA cache banks can be dynamically partitioned and reallocated to different processors. Since the banks are individually addressable, the memory system may be reconfigured to support different programming models such as streaming or vector workloads by sending configuration commands to individual banks.

A. A NUCA based cache organization in CMP

[7] proposes an organization for the on-chip memory system for a chip multiprocessor, wherein the L2 cache is organized as a NUCA array with a switched network embedded in it for high performance. They carry out an extensive study on the effect of various sharing degrees of L2 cache under the NUCA organization, from unshared to completely shared, among processors to find the optimal cache sharing degree that would give the best average performance across a range of commercial and scientific applications. The authors define the term sharing degree as the number of processors that share the L2 cache.

The authors employ two techniques for placing blocks into the memory banks: S-NUCA (static) and D-NUCA (dynamic). In S-NUCA each block is placed in one specific memory bank, whereas in D-NUCA each block can reside in one of multiple banks and can migrate towards the requesting processor based on its access frequency and thus reduce the hit latency. In spite of this, D-NUCA has its own inherent problems. Blocks requested by multiple processors will end up ping-ponging between the memory banks and hence would negate any advantage derived from reduction in hit latencies. More importantly, searching blocks in bank sets becomes more complicated as the block might be in any memory bank and thus would require a smart search to identify the position of the memory block.

[7] solves the problem of ping-ponging of blocks between two banks by using saturated counters which allows a block to migrate only if the relevant counter for that moving direction is saturated. On the other hand, in order to provide a fast searching mechanism, a partial tag array is assigned with each column to keep track of the state of blocks cached in the respective column. Any changes in a bank columns contents is to be reflected in its partial tags synchronously.

On simulation of the CMP cache paradigm under consideration, it is observed in [7] that, although no single sharing degree exists to provide the best performance for all the benchmarks, a simple static NUCA with sharing degree of 2 or 4 does well among the majority of benchmarks. On the other hand, dynamic NUCA, though it tries to decrease the wire delays (thus, decreasing the hit latency time) by moving the block towards the CPU that frequently accesses it, does not have significant improvement in performance and rather increases the overall energy consumption due to the complex searching mechanism and cache line movement as shown in Figure 1b.

B. NuRAPID

While the idea of NUCA appears to be attractive, NUCA’s design choices have certain inherent problems. Some of the problems discussed in [8] and pertinent to current CMP cache architectures are as follows:

- Tag search: Unlike small caches which access both the tag and the data arrays in parallel, large caches often probe the tag array first, and then access only the matching data way. Although intended for large caches, NUCA does not employ a sequential tag-data access. It either does a parallel or sequential search for the ways. One has to take into consideration that the entire tag array is smaller than even one data way and hence sequential tag-data access is more energy efficient than sequential way search.

Adding to NUCA’s energy inefficiencies is the fact that the tag array and the data array is distributed
throughout the die, which consequently requires traversing a switched network.

- Placement: NUCA artificially couples data placement with tag placement [6], which means the position in tag array implies the position in data array. This coupling implies that NUCA can place only a small number of ways of each set in the fastest distance-group (a collection of data subarrays all of which are at the same latency from the processor). NUCA uses a policy of promoting frequently-accessed blocks from slower to faster d-groups by swapping the ways within a set, but then these swaps use up a lot of energy.

NuRAPID (Non-uniform access with Replacement And Placement using Distance associativity) [8] addresses the above problems. For the first problem discussed, NuRAPID uses a sequential tag-data access with a centralized tag array. The above solution also implies a solution to the second problem of decoupling the tag and the data placement problem. This decoupling allows distance associativity, which allows a completely flexible choice of d-groups for data placement. In solving these problems distance associativity fundamentally changes the trade-offs made by NUCA’s best-performing design, resulting in higher performance and substantially lower energy.

C. CMP-NuRAPID

[9] extends the NuRAPID from uniprocessor to CMPs. It is called CMP-NuRAPID. The motivation behind CMP-NuRAPID revolves around three basic ideas.

- Placing copies close to requestors allow fast access for read-only sharing but effectively reduces on-chip capacity in CMPs. [9] proposes controlled replication.
- Rather than incur slow access to read-write shared data through coherence misses, [9] proposes in-situ communication which provides fast access to the data without making copies or incurring coherence misses.
- Much migration of data blocks as is in case of NUCA may result in inefficient use of the on-chip capacity. For example, if one core exceeds the capacity of its private cache, migrating new blocks closer to the core will cause evictions even if there is unused on-chip capacity in a neighbor’s private cache. [9] proposes capacity stealing which enables a core to migrate its less-frequently-accessed data to unused frames in neighboring caches with less capacity demand.

As neither pure shared nor pure private cache could accommodate the aforementioned ideas, [9] proposes a hybrid of private, per-processor tag arrays and a shared data array. [9] employs NuRAPID to hold frequently-accessed data in regions close to the requester. To provide fast access to the tag, CMP-NuRAPID provides each core with its own private tag array, which snoops on a bus for coherence. Though CMP-NuRAPID uses non-uniform cache access like NUCA [6] there is one key difference between them: While CMP-NuRAPID employs replication in the shared data array to allow fast access for shared data, and customizes its replication via controlled replication and in-situ communication to exploit CMP’s latency and capacity characteristics, NUCA [6] inflexibly opts for disallowing replication altogether and relies only on migration.

CMP-NuRAPID uses controlled replication to force only one data copy between the reader and the writer, while they have independent tag copies which point to the single data copy. CMP-NuRAPID also introduces a new state called the “communication” state in order to prevent the writer from invalidating the readers’ tag copies. This is a state to incorporate the situation wherein the writer can write to the data copy and the reader can read the copy without incurring coherence misses. For capacity stealing of private data, CMP-NuRAPID exploits non-uniform access and modifies NuRAPID’s promotion and demotion[8] policies to migrate frequently-accessed blocks close to the core.

[9] shows that CMP-NuRAPID improves performance by 13% over a shared cache and 8% over private caches for three commercial multi-threaded workloads.

III. HYBRID CACHE MODELS

A. Hybrid structure to manage wire delay

[10] proposes a cache model for CMPs which incorporates some of the existing paradigms to form a hybrid cache model exclusively designed to suffice the needs of future CMPs with wire delay as their major challenge. The design in [10] revolves around the solution to the following three access latency causes:

- CMPs often share the on-chip L2 cache, requiring multiple ports to provide sufficient bandwidth.
- Multiple threads mean multiple working sets, which compete for limited on-chip storage.
- Sharing code and data interferes with block migration, since one processor’s low-latency bank is another processor’s high-latency bank.

[10] incorporates the following techniques in the cache design to take care of the above mentioned issues:

- Strided Prefetching: Stride prefetchers utilize repeatable memory access patterns to tolerate cache miss latency.
- Block migration: Block migration reduces global wire delay from L2 hit latency by moving frequently accessed cache blocks closer to the requesting processor.
- On-chip Transmission Lines: On-chip transmission line technology reduces L2 cache access latency by replacing slow conventional wires with ultra-fast transmission lines.

B. Cache model adaptive to Hybrid Workloads

Most of the CMP cache architectures proposed till now to improve system performance have not considered parallel execution of mixed single threaded and multi-threaded workloads. [11] proposes a hybrid workload-aware cache structure namely SPS2, in which each processor has both private and shared L2 caches. This way, SPS2 in its design
takes advantage of the low latency of private and the high capacity of shared L2.

The proposed SPS2 cache scheme is shown in Figure 2. In the figure, SL2 (shared L2) is a multi-banked multi-port cache that could be accessed by all the processors directly over the bus. In order to minimize the latency of PL2 (private L2) and SL2 (shared L2) [11] has adopted design layout similar to one used in CMP-NUCA[7]. According to the design plan SL2 resides in the centre of the chip with processors placed around the SL2 while the PL2s are located around the outer boundary, close to the processor. As shown in Figure 2, two local buses are used to connect L1 and PL2 (dashed line), and L1 and SL2 (solid line). Bus transactions for cache coherence separately run on another bus between L2s and memory (bold line).

According to the design, data in PL1 and PL2 are exclusive but data in PL1 and SL2 may be inclusive. All new data is fetched from memory to PL1 as private data initially. If private data is evicted from PL1, it is placed in PL2. If shared data is evicted from PL1 then it is placed in SL2. At times when PL2 is full, the private data might steal the capacity from SL2. In order to get around this problem, unlike the unified L2 cache structure, the SPS2 system with its split private and shared L2 caches can be flexibly and individually designed according to demand of the workload.

PL2 is designed as a direct-mapped cache to provide fast access and low power, while SL2 is designed as a set-associative cache to reduce conflict. PL2 and SL2 do not have to match each other in size, and they could have different replacement policies. The SPS2 cache system does not need any new additional CPU instructions to support its protocol.

C. Nahalal

[12] introduces a new CMP cache architecture, Nahalal, that partitions the L2 cache according to the program’s data sharing behavior, and can thus offer vicinity of reference to both shared and private data.

Uniprocessor microarchitectures typically partition the cache based on content and hierarchy. [12] argues that with the shift to CMP architectures, additional cache dimensions will prove valuable, for example, based on data sharing, cache coherency, and other CMP characteristics. [12] proposes a new CMP cache architecture that partitions the L2 cache according to the programs' data sharing. The motive behind this design is on the lines similar to the SPS2 architecture [11] discussed earlier.

As seen before, D-NUCA uses line migration in order to move frequently accessed data closer to processors that use it. This policy is intended to reduce the average access times to the most frequently accessed data, and thus reduce the average access time compared to static line placement. But it is a well known fact that access to shared data hinders the effectiveness of D-NUCA, since shared data, being pulled by several processors to different direction, ends up in the middle of the chip, far from all of them. [12] show that the Nahalal concept of bring shared data close to all processors can solve this problem of DNUCA, and may provide a platform where D-NUCA can realize its potential.

Using hybrid benchmarks, [11] compares SPS2 with two traditional cache architectures shared and private L2. Simulation results show that remote accesses and off-chip accesses of SPS2 are less than those of L2P and L2S. The SPS2 scheme reduces the L1 miss latency by 9% as compared with the L2P scheme, and 5% compared with L2S scheme.

[11] also observes that the network traffics of SPS2 are 14-29% less than those in L2P, and 33-51% less than L2S.

Based on the research presented in [11], it is clear that hybrid workload-aware cache design scheme allows better system performance under mixed single-thread and multi-thread loads. In addition, through comparison of different CPU affinity configuration, the performance of SPS2 could be improved when single-threads are exclusively assigned to separate CPUs.

Figure 2: SPS2 Cache Architecture

Using hybrid benchmarks, [11] compares SPS2 with two traditional cache architectures shared and private L2. Simulation results show that remote accesses and off-chip accesses of SPS2 are less than those of L2P and L2S. The SPS2 scheme reduces the L1 miss latency by 9% as compared with the L2P scheme, and 5% compared with L2S scheme.

[11] also observes that the network traffics of SPS2 are 14-29% less than those in L2P, and 33-51% less than L2S.

Based on the research presented in [11], it is clear that hybrid workload-aware cache design scheme allows better system performance under mixed single-thread and multi-thread loads. In addition, through comparison of different CPU affinity configuration, the performance of SPS2 could be improved when single-threads are exclusively assigned to separate CPUs.

Figure 3: Two Cache Organization for 8-way CMP

The main concept in Nahalal is to place shared data in a relatively small area in the middle of the chip, surrounded by the processors, and locating the private data in the periphery. Figure 3(a) depicts a general organization for an 8-way CMP where the L2 cache is located in the middle (CIM) and (b) portrays an alternative layout based on the Nahalal concept. In both designs, each processor has a private L1 cache, and all processors share the L2 cache. The L2 cache capacity is partitioned among the processors such that each processor has one cache bank in its proximity.

In both the implementations show in the above figure, each address can be located in any of the banks. Thus, cache management needs to decide where to place the line when it is fetched, and subsequently, if and when to migrate a line from...
its current bank, and to where. In both CIM and Nahalal implementations, on a first line fetch, the line is placed in the bank adjacent to the processor that made the request.

In case of CIM, the line remains in its initial location as long as it is in the cache. In contrast, Nahalal employs migration to steer shared-hot-lines to the center. In order to prevent pollution of the shared cache area, a line is migrated to the shared structure only after N (threshold) accesses from different processors.

During the search process in case of Nahalal, L2 cache lines are likely to be served either from the center or from the local cache structure. This is contrary to CIM, where shared lines can be located in any of the different banks with equal probability.

Nahalal's gain over traditional CMP cache design increases with the frequency of accesses to shared data. The exact extent of this gain in each benchmark depends on the percentage of accesses it makes to shared data among L2 accesses. Thus, for benchmarks with low L2 access rate, where L2 is not the bottleneck, or for benchmarks with almost no sharing Nahalal's superiority is not effectively realized in overall speedup.

In the future, one can expect CMP applications to have larger memory demands and exhibit more sharing; while the growth in wire delays will increase the importance of locality of reference. Therefore, the benefits of Nahalal will become more significant.

D. Cooperative Caching

Cooperative caching [13] is a cache organization to achieve the benefits of both private and shared caches. While each processor has its own private L2 cache, caches can cooperate together to form an aggregate cache to overcome the inefficient use of resources and limitations of traditional private caches. The design of the cooperative cache is shown in Figure 4.

Figure 4: Cooperative caching where the shaded area represents the aggregate shared cache formed via cooperation

The cooperation among caches is achieved through allowing data that are evicted from a local cache to be hosted by another processor's cache which is called spilling. And thus on a private L2 cache miss, data can be obtained from other L2 caches by cache to cache transfer, if available. Cache to cache transfer of clean data is used to help avoid memory access on an L2 cache miss if data can be obtained from another processor. When a block is "split" by a cache, it is hosted by another L2 cache which is chosen at random but gives higher probability to closer neighbors. The replacement policy for private cache is to evict replicas as long as there exists one otherwise evict a single by LRU.

The organization presented in [13] allows data to be replicated and thus the replacement algorithm gives more priority to singlets (cache blocks that are not replicated) by evicting them only when no replicas are available as victims. The evicted block can be spilled to another cache (chosen at random but gives more probability for closer neighbors) which also use the same replacement policy. Global replacement of inactive data is achieved by monitoring the number of times a block can be spilled out of a cache.

In order to achieve the above requirements, [13] proposes a hardware design which can be achieved through modification to various existing implementations. [13] uses a central directory to enable cache to cache transfer. In order to select a clean owner that has the required data, directory lookup will be directed to the corresponding bank to search the related cache sets in parallel. In addition, it needs a central coherence engine that updates the information in the central directory to reflect what blocks are stored in each cores private caches.

This cache organization has great advantage because of its ability of achieving the benefits of both private and shared L2 caches. It reduces the runtime of simulated workloads by 4-38 percentage, and performs at worst 2.2 percentage slower than the best of private and shared caches in extreme cases. [13] proposes changes to the hardware implementation that can be achieved by modifications to existing implementations. On the other hand, it requires 2.3 percentage of increase in on-chip cache capacity. In addition, it can suffer from pollution because of the uncontrolled way by which partitions are shared among cores since it only has a co-operation probability that is consulted upon cache replacement.

E. Victim Replication

Victim replication [14] is a variant of the shared scheme which attempts to keep copies of local primary cache victims within the local L2 cache slice. This idea coined is based on the fact that most frequently requested L2 blocks are also the most frequently evicted L1 blocks. Hits to the replicated copies reduce the effective latency of the shared L2 cache, while retaining the benefits of a higher effective capacity for shared data.

[14] considers the base CMP architecture as an array of replicated tiles connected over a switched network. In this architecture each tile contains a processor with primary caches, a slice of the L2 cache, and a connection to the on-chip network, as show in Figure 5.
Victim replication, a simple hybrid scheme, tries to combine the large capacity of shared L2 scheme with the low hit latency of private L2 scheme. L2VR (Victim Replication cache scheme) although based on shared L2 scheme tries to minimize the hit latency by allowing multiple copies of a cache line to co-exist in different L2 slices of the shared L2 cache. Each copy of an L2 cache line residing on a tile other than its home tile is a replica. In effect, replicas form dynamically-created private L2 caches to reduce hit latency for the local processor.

When a processor misses in the shared L2 cache, a line is brought from the memory and placed in the on-chip L2 at a home tile determined by a subset of the physical address bit. The requested line is also directly forwarded to the primary cache of the requesting processor. On an eviction of a primary cache line due to conflict or capacity miss, the VR scheme attempts to keep a copy of the victim line in the local slice to reduce subsequent access latency to the same line (based on the basic observation). If the line’s residency in the primary cache is terminated because of an incoming invalidation or writeback request, then a normal L2S protocol is followed.

All primary cache misses must now first check the local L2 tags in case there’s a valid local replica. On a replica miss, the request is forwarded to the home tile. On a replica hit, the replica is invalidated in the local L2 slice and moved into the primary cache. When a downgrade or invalidation request is received from the home tile, the L2 tags must also be checked in addition to the primary cache tags.

L2VR has a small area overhead over L2S, because the L2 tags must be wide enough to hold physical addresses from any home tile, thus the tag width becomes the same as L2P. Global L2 lines redundantly set these bits to the address index of the home tile. Replicas of remote lines can be distinguished from regular L2 lines as their additional tag bits do not match the local tile index.

Victim replication reduces the average memory access latency of the shared L2 cache by an average of 16% for multi-threaded benchmarks and 24% for single-threaded benchmarks, providing better overall performance than either private or shared schemes.

F. Adaptive Selective Replication for CMP Caches

Some CMPs use shared L2 cache to maximize the on-chip cache capacity and minimize the off-chip misses. Others use private L2 caches, replicating data to limit the delay due to global wires and minimize cache access time. Some of the hybrid proposals discussed earlier like CMP-NuRAPID and Victim Replication have used selective replication to balance latency and capacity but the designs have been based on static replication rules. These schemes perform better than private and shared caches for selected workloads and system configuration. As a result rather than bettering performance at times, it results in performance degradation for some combinations of workloads and system configurations.

IV. EXPERIMENTATION

Prior research has shown that database system performance is highly dependent on memory hierarchy as off-chip data stalls dominate its overall performance to a large extent. On the other hand, Chip Multiprocessors are gradually emerging as the prime processors of the future mainstream servers due to their ability to leverage the parallelism of multithreading and multitasking to achieve higher performance. We through these experiments aim to understand the effects of a commercial DBMS workload tpc-w [16] on three different kinds CMP cache architecture model and look to find the L2 cache size best suited for the given workload. The units in particular that we look to study and base our results on are Hit Rate and the Total time spent on stall due to L2 cache misses.

A. Experimental Setup

We characterized the performance of the tpc-w workload on three different CMP cache configurations using a trace driven CMP cache simulator [1] which we implemented. The memory trace is generated using pintool [16]. While generating the trace, we skipped the first billion instructions to avoid the overhead of loading the database server. Due to certain constraints (computational and otherwise), the trace we used consists of 1 million memory addresses. As for the cache simulator, it has three distinct implementations depending on the configurations taken into consideration. The configurations are as follows:

- Private L2 (L2P): In this approach the L2 cache is partitioned between the chip processor cores equally, and each core can access its own distinct L2 space.
- Shared L2 (L2S): In this configuration the L2 cache is placed on the other side of the interconnect bus between the cores, i.e. the same L2 space is shared among all the cores, and the coherence actions take place between the L1s. Upon missing in the L1s, L2 is looked up and only upon a miss is an off-chip access needed. This approach is also the organization in current chip multiprocessors such as the Piranha [2] and Hydra [3].

Figure 5: A tiled CMP design

Victim replication reduces the average memory access latency of the shared L2 cache by an average of 16% for multi-threaded benchmarks and 24% for single-threaded benchmarks, providing better overall performance than either private or shared schemes.

F. Adaptive Selective Replication for CMP Caches

Some CMPs use shared L2 cache to maximize the on-chip cache capacity and minimize the off-chip misses. Others use private L2 caches, replicating data to limit the delay due to global wires and minimize cache access time. Some of the hybrid proposals discussed earlier like CMP-NuRAPID and Victim Replication have used selective replication to balance latency and capacity but the designs have been based on static replication rules. These schemes perform better than private and shared caches for selected workloads and system configuration. As a result rather than bettering performance at times, it results in performance degradation for some combinations of workloads and system configurations.

IV. EXPERIMENTATION

Prior research has shown that database system performance is highly dependent on memory hierarchy as off-chip data stalls dominate its overall performance to a large extent. On the other hand, Chip Multiprocessors are gradually emerging as the prime processors of the future mainstream servers due to their ability to leverage the parallelism of multithreading and multitasking to achieve higher performance. We through these experiments aim to understand the effects of a commercial DBMS workload tpc-w [16] on three different kinds CMP cache architecture model and look to find the L2 cache size best suited for the given workload. The units in particular that we look to study and base our results on are Hit Rate and the Total time spent on stall due to L2 cache misses.

A. Experimental Setup

We characterized the performance of the tpc-w workload on three different CMP cache configurations using a trace driven CMP cache simulator [1] which we implemented. The memory trace is generated using pintool [16]. While generating the trace, we skipped the first billion instructions to avoid the overhead of loading the database server. Due to certain constraints (computational and otherwise), the trace we used consists of 1 million memory addresses. As for the cache simulator, it has three distinct implementations depending on the configurations taken into consideration. The configurations are as follows:

- Private L2 (L2P): In this approach the L2 cache is partitioned between the chip processor cores equally, and each core can access its own distinct L2 space.
- Shared L2 (L2S): In this configuration the L2 cache is placed on the other side of the interconnect bus between the cores, i.e. the same L2 space is shared among all the cores, and the coherence actions take place between the L1s. Upon missing in the L1s, L2 is looked up and only upon a miss is an off-chip access needed. This approach is also the organization in current chip multiprocessors such as the Piranha [2] and Hydra [3].

Figure 5: A tiled CMP design

Victim replication, a simple hybrid scheme, tries to combine the large capacity of shared L2 scheme with the low hit latency of private L2 scheme. L2VR (Victim Replication cache scheme) although based on shared L2 scheme tries to minimize the hit latency by allowing multiple copies of a cache line to co-exist in different L2 slices of the shared L2 cache. Each copy of an L2 cache line residing on a tile other than its home tile is a replica. In effect, replicas form dynamically-created private L2 caches to reduce hit latency for the local processor.

When a processor misses in the shared L2 cache, a line is brought from the memory and placed in the on-chip L2 at a home tile determined by a subset of the physical address bit. The requested line is also directly forwarded to the primary cache of the requesting processor. On an eviction of a primary cache line due to conflict or capacity miss, the VR scheme attempts to keep a copy of the victim line in the local slice to reduce subsequent access latency to the same line (based on the basic observation). If the line’s residency in the primary cache is terminated because of an incoming invalidation or writeback request, then a normal L2S protocol is followed.

All primary cache misses must now first check the local L2 tags in case there’s a valid local replica. On a replica miss, the request is forwarded to the home tile. On a replica hit, the replica is invalidated in the local L2 slice and moved into the primary cache. When a downgrade or invalidation request is received from the home tile, the L2 tags must also be checked in addition to the primary cache tags.

L2VR has a small area overhead over L2S, because the L2 tags must be wide enough to hold physical addresses from any home tile, thus the tag width becomes the same as L2P. Global L2 lines redundantly set these bits to the address index of the home tile. Replicas of remote lines can be distinguished from regular L2 lines as their additional tag bits do not match the local tile index.

Victim replication reduces the average memory access latency of the shared L2 cache by an average of 16% for multi-threaded benchmarks and 24% for single-threaded benchmarks, providing better overall performance than either private or shared schemes.

F. Adaptive Selective Replication for CMP Caches

Some CMPs use shared L2 cache to maximize the on-chip cache capacity and minimize the off-chip misses. Others use private L2 caches, replicating data to limit the delay due to global wires and minimize cache access time. Some of the hybrid proposals discussed earlier like CMP-NuRAPID and Victim Replication have used selective replication to balance latency and capacity but the designs have been based on static replication rules. These schemes perform better than private and shared caches for selected workloads and system configuration. As a result rather than bettering performance at times, it results in performance degradation for some combinations of workloads and system configurations.

IV. EXPERIMENTATION

Prior research has shown that database system performance is highly dependent on memory hierarchy as off-chip data stalls dominate its overall performance to a large extent. On the other hand, Chip Multiprocessors are gradually emerging as the prime processors of the future mainstream servers due to their ability to leverage the parallelism of multithreading and multitasking to achieve higher performance. We through these experiments aim to understand the effects of a commercial DBMS workload tpc-w [16] on three different kinds CMP cache architecture model and look to find the L2 cache size best suited for the given workload. The units in particular that we look to study and base our results on are Hit Rate and the Total time spent on stall due to L2 cache misses.

A. Experimental Setup

We characterized the performance of the tpc-w workload on three different CMP cache configurations using a trace driven CMP cache simulator [1] which we implemented. The memory trace is generated using pintool [16]. While generating the trace, we skipped the first billion instructions to avoid the overhead of loading the database server. Due to certain constraints (computational and otherwise), the trace we used consists of 1 million memory addresses. As for the cache simulator, it has three distinct implementations depending on the configurations taken into consideration. The configurations are as follows:

- Private L2 (L2P): In this approach the L2 cache is partitioned between the chip processor cores equally, and each core can access its own distinct L2 space.
- Shared L2 (L2S): In this configuration the L2 cache is placed on the other side of the interconnect bus between the cores, i.e. the same L2 space is shared among all the cores, and the coherence actions take place between the L1s. Upon missing in the L1s, L2 is looked up and only upon a miss is an off-chip access needed. This approach is also the organization in current chip multiprocessors such as the Piranha [2] and Hydra [3].
Shared Processor based split L2 (L2SP): This configuration based on the concept coined in [4] is a hybrid of the L2P and L2S. It captures the benefits of the two basic organizations, while avoiding many of their drawbacks. In this structure there are multiple L2 units on the other side of the bus, but is different from L2S in that the units for lookup are selected based on the processor cores that issued the requests. This allows better utilization of the L2 space to balance the load between CPUs, while still relatively insulating such usage between the CPUs.

Overall, the CMP base simulation platform we consider can simulate on 4-12 cores, with the total L2 cache capacity being divided according to the implementation (L2P, L2S, L2SS). If for example the total L2 capacity being considered is 1MB. Then in case of L2P each core will have its L2 capacity of 256KB while in case of L2S there would be one single L2 cache of size 1MB being shared between the cores. In our implementation of L2SP, for example with no. of cores as 4, the L2 cache is partitioned into 2 slices of 512KB each. 2 cores share one of the L2 partitions.

As a part of our Cache implementations we do not treat read and write accesses as being different. It has been shown in [14] that, for commercial workloads, shared read-only blocks account for 42-71% of L2 requests. Our implementation incorporates basic invalidation protocol.

B. Observations and Analysis

We started by studying the effects of the tpc-w workload on CMP architecture with 4 cores with L2 cache sizes ranging from 1MB to 10MB. As seen in Figure 6, when it comes to absolute Hit Rate the Shared Processor based split L2 (L2SS) outperforms the two base models namely: L2P and L2S. L2SS improves the hit rate by approximately 32% and 60% respectively on an average. From the graphs in Figure 6, one can observe that the ideal L2 cache size for CMP architecture with split shared cache organization is around 6-7MB.

With increased cache size there is always an increase in the access time. We felt an understanding of total time spent on memory stalls was essential to complement the study of absolute Hit rates. In order to obtain the total time spent on stalls, we applied the following formula:

Total time spent on stalls = (No. of L1 misses x L2 access time) + (No. of L2 misses x memory access latency)

We estimate cache access latencies using Cacti 5.3 [5] and considered memory access latency as 55ns. Figure 7 shows that for the three organizations, the time spent on memory stalls increase as the size of the cache increase which indicates that the increased access latency outweigh the gain achieved by higher hit rate. In addition, it is clear that the total time spent on memory stalls for L2SS outperforms the other two base cache paradigms. This suggests that having a limited degree of sharing; some point between private and shared cache organizations; can help improve the performance of database workloads.

Taking into consideration the fact that a CMP architecture with 4 cores and with cache paradigm as L2SS perform really well, we studied the impact of high levels of core integration on chip. As Figures 6 and 7 reflect the cache size of 6MB being optimal either way - total time spent on stalls and hit rate, we consider the effect of no. of cores ranging from 4-12
on the same two parameters with the total capacity of L2 cache being 6MB.

Figure 8: Hit rate vs No. of cores

Figures 8 and 9 suggest that with increase in number of cores from 4 to 12 Hit-rate increases by as much as 5.8% while access time decreases by 55%, but the change is not significant enough. This implies that until and unless we actually test with multiple independent threads/hybrid workload we won’t get to see the actual advantage of increasing the no. of cores.

Figure 9: Total time spent on stalls vs No. of cores

VI. CONCLUSION

We presented a survey about various cache organizations for chip-multiprocessors. We chose to implement three of these organizations which are private, shared, and split shared L2 cache. We compared these various organizations on the basis of hit rate as well as the time spent on memory stalls. The results shows that split shared L2 organization outperforms both private and shared L2 caches which suggests that having a limited degree of sharing; some point between private and shared cache organizations; can help improve the performance of database workloads.

ACKNOWLEDGMENT

We thank Susmit Biswas for his help.

REFERENCES