Bank-aware Dynamic Cache Partitioning for Multicore Architectures

A Seminar Report
Submitted in partial fulfilment of
the requirements for the award of the degree of

Master of Technology
in
Computer Science and Engineering

by

Anoop S S
105102

Department of Computer Science & Engineering
College of Engineering Trivandrum
Kerala - 695016
2010-11

College of Engineering Trivandrum
Department of Computer Science & Engineering

Certified that this Seminar Report entitled

Bank-aware Dynamic Cache Partitioning for Multicore Architectures

is a bonafide record of the seminar presented by

Anoop S S
105102

in partial fulfilment of
the requirements for the award of the degree of
Master of Technology
in
Computer Science & Engineering

Mr. Rameez Mohammed A
Guide
Lecturer
Dept.of Computer Science & Engineering

Dr. M.S Rajasree
Professor and
Head
Dept.of Computer Science & Engineering

Acknowledgement

I would like to express my sincere gratitude and heartful indebtedness to my guide **Mr Rameez Mohammed A**, Lecturer , Department of Computer Science And Engineering for his valuable guidance and encouragement in pursuing this seminar .

I am thankful to **Dr. M S Rajasree**, Head of the Department , **Shine S** , P.G Coordinator, **Sreelekshmi R**, Asst. Professor and **Anwar A**, Asst. Professor and project coordinator Department of Computer Science and Engineering for their help and support.

I also acknowledge my gratitude to other members of faculty in the Department of Computer Science And Engineering and all my friends for their whole hearted cooperation and encouragement.

Above all I am thankful to the God Almighty.

Anoop S S

Contents

1 Introduction **5**

2 CMP-Baseline **6**

3 Bank-Aware Cache Partitioning **7**
3.1 Cache Profiling of Applications 7
3.2 Bank-aware Assignment of Cache Capacity 9
3.2.1 Cascade 10
3.2.2 Address Hash 10
3.2.3 Parallel 10
3.3 Allocation Algorithm on CMP 12

4 Evaluation **14**
4.1 Bank-aware vs. Unrestricted Partitioning 14
4.2 Detailed Simulation Results 14

5 Conclusion **16**

Abstract

As Chip-Multiprocessor systems (CMP) have become the predominant topology for leading microprocessors, critical components of the system are now integrated on a single chip. This enables sharing of computation resources that was not previously possible. In addition, the virtualization of these computational resources exposes the system to a mix of diverse and competing workloads. Cache is a resource of primary concern as it can be dominant in controlling overall throughput. In order to prevent destructive interference between divergent workloads, the last level of cache must be partitioned. In the past, many solutions have been proposed but most of them are assuming either simplified cache hierarchies with no realistic restrictions or complex cache schemes that are difficult to integrate in a real design. To address this problem a dynamic partitioning strategy based on realistic last level cache designs of CMP processors was proposed. This uses a cycle accurate, full system simulator based on Simics and Gems to evaluate the partitioning scheme on an 8-core DNUCA CMP system. Results for an 8-core system show that the proposed scheme provides on average a 70 percent reduction in misses compared to non-partitioned shared caches, and a 25 percent misses reduction compared to static equally partitioned (private) caches.

1 Introduction

Chip Multiprocessors (CMP) have gradually become an attractive architecture for leveraging system integration by providing capabilities on a single die that would have previously occupied many chips across multiple small systems. This integration has brought abundant on-chip resources that can now be shared in finer granularity among the multiple cores. Such sharing though has introduced chip-level contention and the need of effective resource management policies is more important that ever.

To efficiently exploit these resources, systems require multiple program contexts and virtualization has become a key player in this arena. Many small and/or low utilization servers can now be easily consolidated on a single physical machine, allowing higher utilization of the available resources with significant energy reductions. Such consolidation presents both opportunities and pitfalls to computer architects to best manage these once isolated resources on large CMP designs.

In such virtualization environments, workloads tend to place dissimilar demands on shared resources and therefore, due to resource contention, are much more likely to destructively interfere in an unfair way. Consequently, shared resources' contention become the key performance bottleneck in CMPs Shared resources include, but are not limited to: main memory bandwidth, main memory capacity, cache capacity, cache bandwidth, memory subsystem interconnection bandwidth and system power.

Among these resources, several studies have identified the shared last-level cache (here L2) of CMPs as a major source of performance loss and execution inconsistency. As a solution, most of the proposed techniques control this contention by partitioning the L2 cache capacity and allocating specific portions of it to each core or execution thread. There are both static and dynamic partitioning schemes available that use workload profiling information to make a decision on cache capacity assignment for each core/thread. All of the above techniques are usually based on high-level system characteristic monitoring since low-level activity based algorithms such as LRU replacement fail to provide a strong barrier among workloads competing for shared resources.

2 CMP-Baseline

Fig. 1 shows a 8-core CMP-NUCA baseline system. This design uses as the last-level of cache a DNUCA L2 cache with 16 physical banks that provide a total of 16MB of cache capacity. Each cache bank is configured as an 8-way set associative cache. Another way to see the cache is as a 128-way equivalent cache that is separated in 16 cache banks of 8 ways each. The eight cache banks that are physically located next to a core are called Local banks and the rest are characterized as Center banks. Cores located next to Local banks have the minimum access latency but that delay can significantly increase when a core needs to access a Local bank physically located next to another core. Center banks have, on average, higher access latency than Local banks but their distance for each core has smaller variation than Local banks and so does the access latency. The access latency to a L2 cache bank varies from 10 up to 70 cycles depending on the physical location of both the core requesting the access and the L2 bank containing the data. A core physical located next to a Local cache bank has to wait 10 cycles to access the bank. The maximum possible latency, without significant network contention, is equal to 70 cycles (i.e core 0 to access the Local bank next to core 7 since it requires 7 hops).

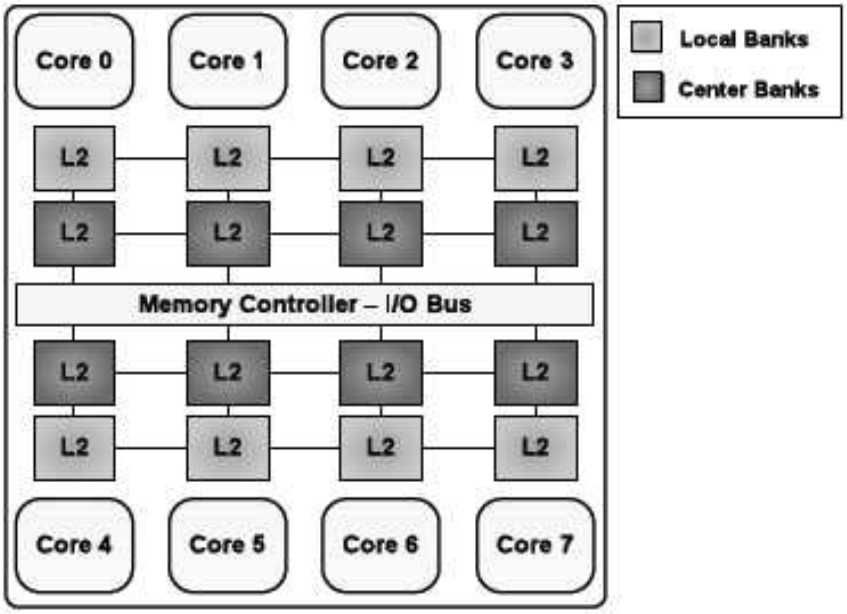

Fig. 1. Baseline CMP system

3 Bank-Aware Cache Partitioning

This section provide details about their application profiling mechanism followed by partitioning algorithm for assigning cache capacity to each core. In the end, the cache partitions allocation algorithm for allocating the cache partitions on the CMP-baseline system are described.

3.1 Cache Profiling of Applications

In order to dynamically profile the cache requirements of each core, a cache miss prediction model based on Mattsons stack distance algorithm is implemented. Mattsons stack algorithm (MSA) was initially proposed by Mattson et al. in for reducing the simulation time of trace-driven caches by determining the miss ratios of all possible cache sizes with a single pass through the trace. The basic idea of the algorithm was later used for efficient trace-driven simulations of a set associative cache. More recently, hardware-based MSA algorithms have been proposed for CMP system resource management .

MSA is based on the inclusion property of the commonly used Least Recently Used (LRU) cache replacement policy. Specifically, during any sequence of memory accesses, the content of an N-sized cache is a subset of the content of any cache larger than N. To create a profile for a K-way set associative cache K+1 counters are needed, named Counter1 to CounterK+1. Every time there is an access to the monitored cache increment only the counter that corresponds to the LRU stack distance where the access took place. Counters from Counter1 up to CounterK correspond to the Most Recently Used (MRU) up to the LRU position in the stack distance, respectively. If an access touches an address in a cache block that was in the i-th position of the LRU stack distance, increment Counteri counter. Finally, if the access ends up being a miss, increment CounterK+1.

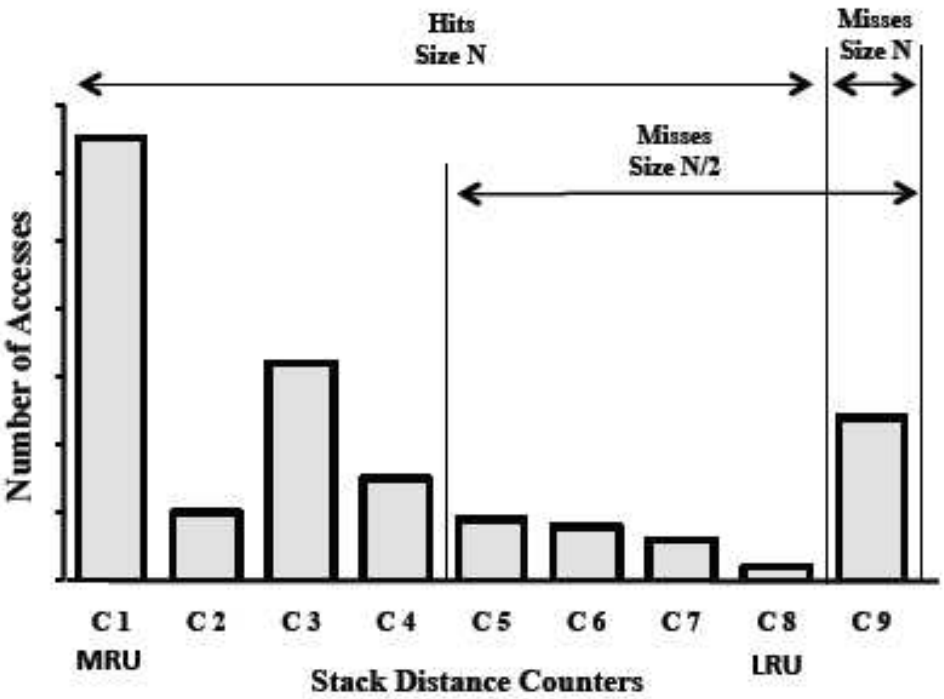

Fig. 2. LRU histograms based on Mattson's stack algorithm

Fig. 2 demonstrates such a MSA profile for an application running on an 8-way associative cache. The application in the example shows a good temporal reuse of stored data in the cache since the MRU positions have a significant percentage of the hits over the LRU one. Based on the application spatial and temporal locality, the graph of Fig. 2 can change accordingly. Using the inclusion property of the LRU replacement policy and having a MSA profile of an N-sized cache, allow us to make a straight-forward prediction of the misses for every L2 cache with size smaller than N. For example, the number of misses that will occur if the cache of Fig. 2 half the size, that is using 4 ways instead of 8 ways, would be the previously measured misses plus the hits of the positions 5 up to 8 of the previous case LRU stack distance. For those positions, the LRU replacement policy will replace the stored data to make room for the one in the MRU positions before they are accessed again. Therefore the accesses that were previously recorded as hits would be misses in the 4-way cache case.

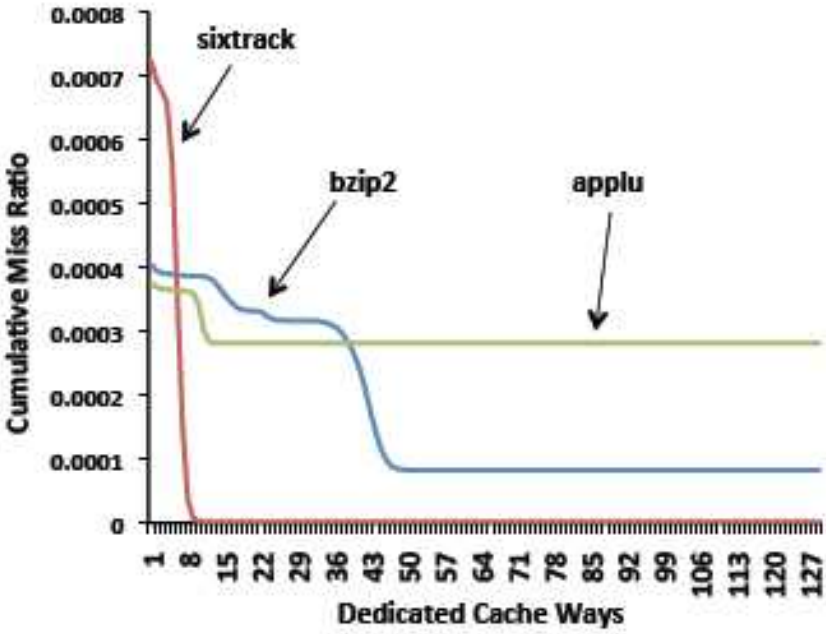

Fig. 3. LRU histograms examples of SPEC CPU2000 benchmarks

Fig. 3 shows the projected cumulative miss ratio of three benchmarks of SPEC CPU2000 benchmark suit. Three examples are selected out of the 26 SPEC CPU2000 workloads, as examples of varied behavior within the whole suit. To create the figure, the stack distance profiles of bzip2, sixtrack and applu with each application executing stand-alone on the baseline CMP using just a single core are collected. To collect the profiles the L2 cache accesses of each core are fed to the MSA histogram profiler described above. The x-axis represents the number of cache ways that are dedicated to each application and the y axis shows the MSA-based projected cumulative miss rate of each application. Sixtrack features a lot of misses with less than six cache ways dedicated to it but after that point, by giving more ways, its misses are close to zero. Therefore, we can have a good fit of sixtracks cache requirements using only one bank (8 ways). In the same way, applu shows a reduction of misses when more than ten ways are dedicated to it, but in this case the miss rate remains flat after more than 10 ways. Therefore, assigning more ways is not beneficial for applu. Lastly, bzip2 shows a behavior somewhere between the two previous cases since additional assigned ways improve miss ratio up to the point where dedicating 45 ways that finally flattens out. Consequently, MSA-based profiling allows us to monitor each cores cache capacity requirements during the execution of an application and based on which we can find the points of cache allocation that can benefit

the miss ratio the most.

A simple with no restrictions implementation would require a complete copy of the cache block tags for each cache set in each one of MSA profilers, which is prohibitively high. The overhead can be greatly reduced by using: a) partial tags b) set sampling and c) maximum assignable capacity reduction techniques. With partial tags one can use less than full tags to identify the cache blocks assigned at each counter thus reducing the storage overhead. Set sampling involves the profiling of a fraction of the available cache sets and therefore it also reduces the number of stored cache tags in the circuit. In addition, the maximum assignable capacity approach assumes that the number of cache-ways that can be assigned to each core is less than the overall number of available cache-ways. In that case, the number of counters are reduced to the maximum number of assignable cache ways per core. The first two reduction techniques are subject to aliasing, which introduces errors and affects the overall accuracy of the profiling circuit. In addition, the maximum assignable capacity can potentially restrict the effectiveness of partitioning scheme by not dedicating bigger portions of a cache to a specific core.

In this paper an implementation based on all of the above methods is proposed. The overhead analysis showed that the use of 12 bit partial tags combined with 1-in-32 set sampling produced error rates within 5% of the profiling accuracy obtained using a full tag implementation. In addition, Bank-aware partitioning assignment algorithm limits each core to a maximum of 9/16 of the total cache capacity. Overall, the implementation overhead is estimated to be 83.25 kbits per cache profiler, which is approximately 0.4% of the 16MB LLC cache design for all the profilers.

3.2 Bank-aware Assignment of Cache Capacity

Prior work in MSA-based cache partitioning was analyzed on fully configurable caches shared among a small number of CPUs. This type of partitioning algorithm is refered as Unrestricted. On the other hand, while Huh et al. in proposed a method for partitioning a CMP-NUCA cache, this relied on a highly banked structure that, features an unrealistic physical implementation. As a solution,the proposed method is to partition cache bank structures using a MSA-based profiling mechanism aligned with current industry directions, that is, using a smaller number of higher capacity cache banks. Such configuration limits the granularity of possible partitions and imposes a set of restrictions over the Unrestricted techniques proposed in the past. This is rooted in the need to aggregate multiple cache banks into a single partition. In the following section several potential aggregation methods that are shown in Fig. 4 are discussed. Aggregation possibilities:

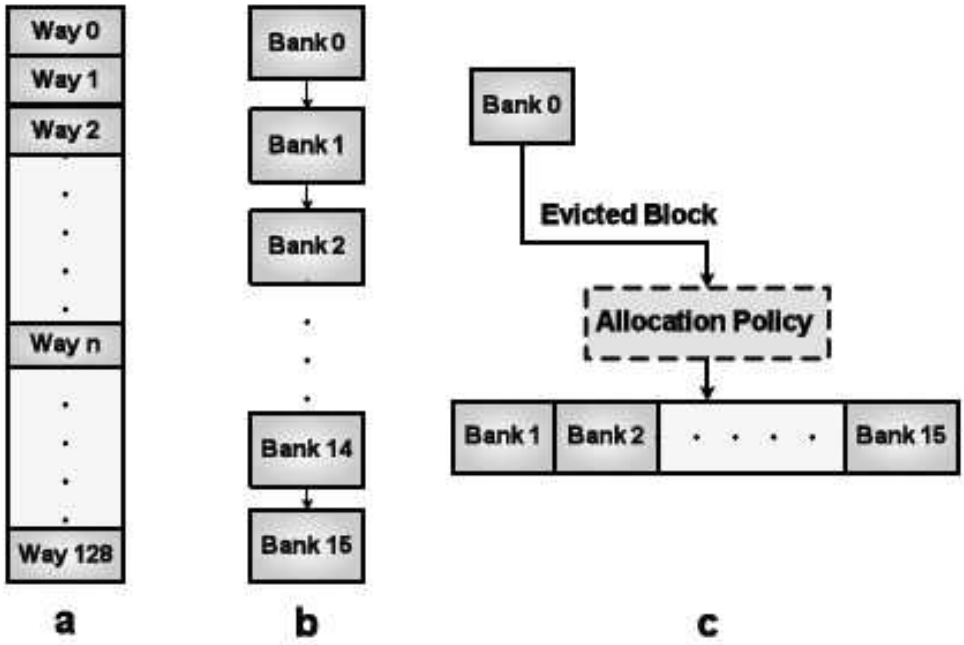

Fig. 4. Cache banks aggregation schemes

3.2.1 Cascade

In this approach, all cache banks that contain portions assigned to a given core are connected head to tail. To match the MSA LRU strategy (Fig. 4.a), all allocations are placed as MRU at the head of the chain. Each allocation causes a shift down of the LRU. Evictions are passed down the chain from LRU out to MRU position in the next bank until a free spot is located (potentially formed from the cache hit that was moved to the top). This structure is show in Fig. 4.b. This method provides for an LRU policy (assuming the banks are also LRU). The advantage of this method is that one can stitch together arbitrary fractions of banks which will emulate the MSA very closely. The primary disadvantage is the high migration rate between cache banks.

3.2.2 Address Hash

A common approach to cache bank aggregation is the use of an address hash. Typically this method is used with a power of two number of cache banks, such that lower order address bits can directly select the bank. While systems have also been built with non-power of two hashes, these require complex modulo operations in the hardware hash function. An example would be the IBM POWER4 and POWER5 processors, which hash across three banks. In addition, Gao et al.and Seznec et al. proposed non-power of two hashing functions with increased complexity over simple hashing functions. Irrespective of the number of cache banks aggregated, this method requires symmetry in that each hashed bank must have the same cache capacity. As such this method has some restrictions. Lastly, address hash features low migration rates.

3.2.3 Parallel

This method is very much like Address Hash, except that a line can be stored in any of the cache banks. Allocation is controlled by round robin/random selection. As such, any given

line can be stored in any of the cache banks. This forces additional look-up operations in the directory structure (which are implement as partial tags). This is less restrictive than Address hash in bank configuration, in that, non-power of 2 aggregations of banks are possible without complex modulo address computations. The migration rate is equivalent to Address Hash, however, power is higher due to wider directory look-ups.

Even though Cascade provides the greatest flexibility, the migration rates observed in simulation are prohibitively high. Both Address hash and Parallel provide reasonable solutions to aggregating cache banks. The only restriction is that multiple banks must have the same capacity. The degradation can be mitigated using the structure shown in Fig. 4.c. In this structure, limit the level of cascading to two. The allocation policy can be either Address Hash or Parallel (Parallel used here). These issues present problems in direct application of currently proposed Unrestricted cache partitioning schemes. Bank-aware assignment algorithm is proposed as a solution for this. This algorithm is based on progressive control of bank granularity. Essentially, as the capacity assignment increases, small deviations from the ideal assignment are tolerable with respect to overall miss reduction. Based on this observation and the bank aggregation requirements, the proposed policies are:

1) Center cache banks are completely assigned to a specific core. This prevents situations where aggregated banks are of different capacities.
2) Any core that is allocated Center banks, will receive a full Local bank.
3) Local cache banks can only be shared with an adjacent core. They only allow per assignment control at Local cache banks. In addition, requiring adjacent sharing provides for low latency and minimal network loads for data transfers.

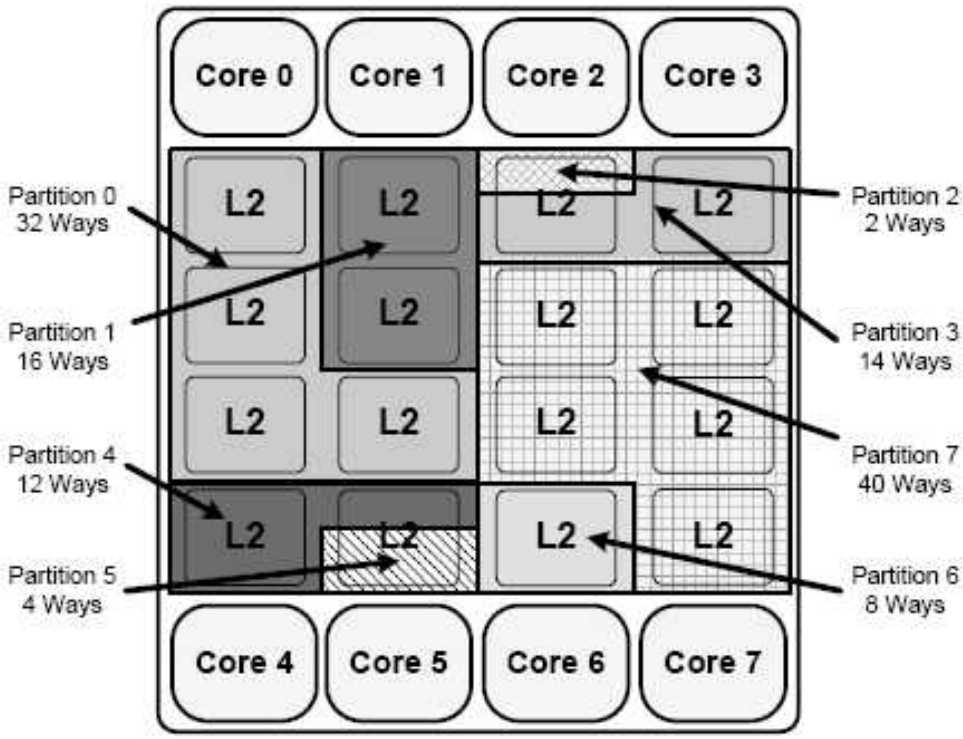

Fig. 5. An example of typical CMP cache partitioning

A typical allocation is shown in Fig. 5. From the figure, most of the cores have multiple L2 cache banks allocated to them except core 2 and core 5. Those cores share the capacity of a single L2 bank with core 3 and 4, respectively. To enforce the selected cache partitions, modify the typical design of a cache bank to support a vertical, finegrain, cache-way partitioning scheme. According to this scheme, each cache-way of a set associative cache can belong to one or more specific cores.

When a specific core suffers a cache miss, a modified LRU policy is used to select the least recently used cache block among the ones that belong to that specific core, for replacement. Therefore, only cache-ways that belong to a specific core or set of cores can be accessed and the rest of the cache-ways that belong to other cores are not affected, eliminating the destructive interference between different workloads running on different cores. To reduce the design complexity, all of the sets in a cache bank are vertically partitioned with the same cache-ways assignment and therefore the granularity of assigning a different cache-way partition is a single cache bank.

3.3 Allocation Algorithm on CMP

In this section Bank-aware assignment algorithm is described in detail. The assignment policy was based on the concept of Marginal Utility. This concept originates from economic theory, where a given amount of resources (in this paper cache capacity), provides a certain amount of utility (reduced misses). The amount of utility relative to the resource is defined as the Marginal Utility.

The MSA histogram provides a direct way to compute the Marginal Utility for a given workload across a range of possible cache allocations. This capability is used to make the best use of the limited cache resources. An iterative approach is, at any point we can compare the Marginal Utility of all possible allocations of unused capacity. Of these possible allocations, the maximum Marginal Utility represents the best use of an increment in the assigned capacity. This algorithm arrives at a capacity assignment via successive steps determining the maximum Marginal Utility for a subset of processors and assignment restrictions.

The overall flow is shown in Fig. 6.The first step of the algorithm is to assign the cache-ways in Center cache banks. Following that, in Box 1, the maximum Marginal Utility is calculated and cache banks are assigned accordingly. For the calculations,we can assume that each Local bank is assigned to the associated processor. In Box 2, check if all the banks are assigned, if not, step 1 is repeated. Following Rules 1 and 2, then mark all processors with Center banks complete (Box 3).

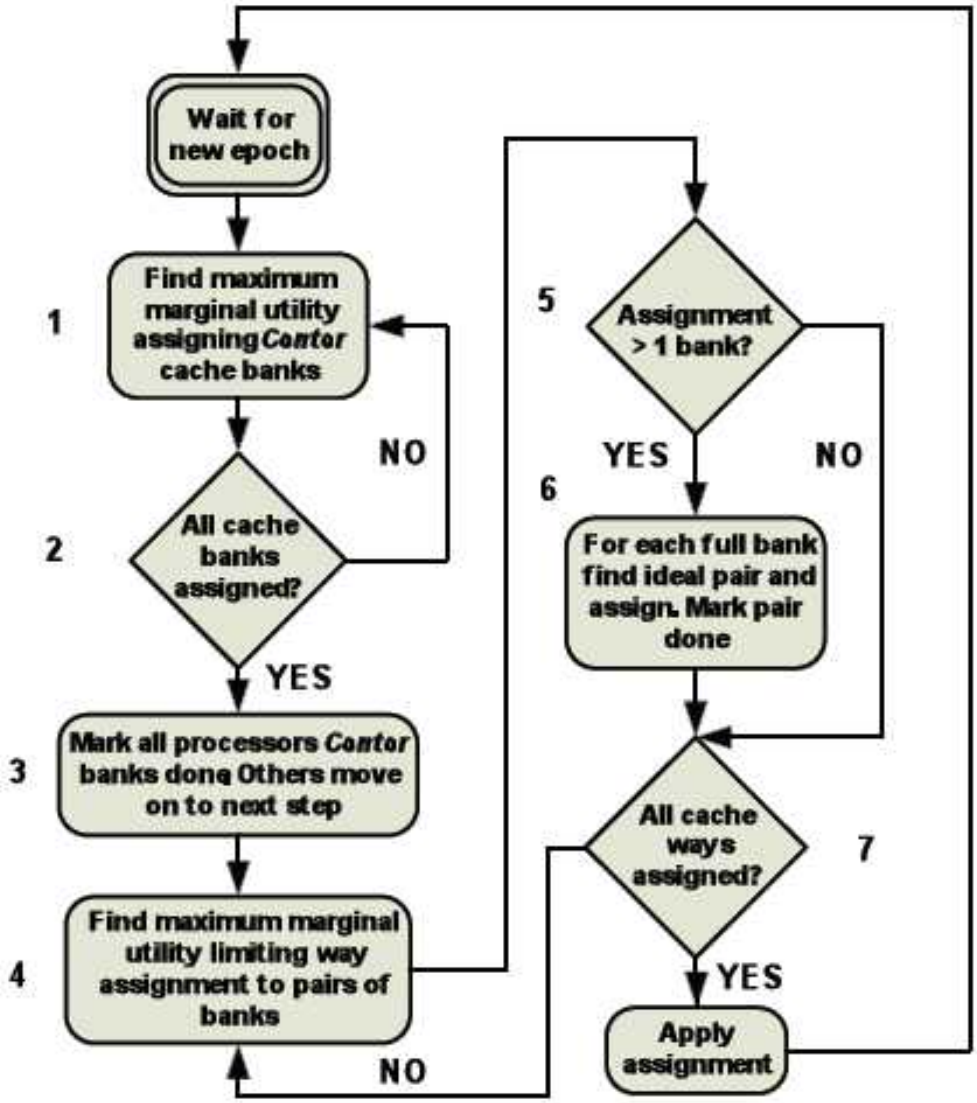

Fig. 6. Cache allocation algorithm flow chart

The next steps are used to solve the Local cache bank partitions. In Box 4, once again find the maximum Marginal Utility,but assignments are limited to possible pairs of processors(in keeping with Rule 3). In Box 5, check if the new assignment has caused any processor to overflow into another processors Local region. If so, find the ideal pair with respect to minimal misses. Essentially defer the pairing as many steps as possible, and make the best pairing choice once it is decided a processor should receive a fraction of an adjacent Local bank. Once the pair is assigned, both processors are marked complete. This step is repeated until all the cache ways are assigned.

4 Evaluation

4.1 Bank-aware vs. Unrestricted Partitioning

Fig. 7 shows the miss rate relative to the even partitions for the Unrestricted and Bank-aware algorithms. A ratio of one represents no reduction in misses compared to static fixed partitioning, while zero indicates all misses are removed with the MSA-based partitioning scheme. From the 1000 sorted results with respect to the miss rate reduction of the Unrestricted scheme. The even partitions and Unrestricted essentially form a performance envelope. Ideally, all of the Bank-aware assignments would fall on the Unrestricted line, which is in general true except some outliners that achieved smaller miss rate reductions than Unrestricted. Both figures give an indication of the range of miss rate reductions possible. On average, the miss rate reduction from the Unrestricted and Bank-aware algorithms are quite comparable. The Unrestricted averages a 30% reduction in misses compared to 27% for the Bank-aware over the case of even partitions. This result shows that the restrictions placed on the allocation algorithm due to the more realistic implementation of L2 cache do not adversely affect the benefits of the MSA-based dynamic cache partitioning scheme.

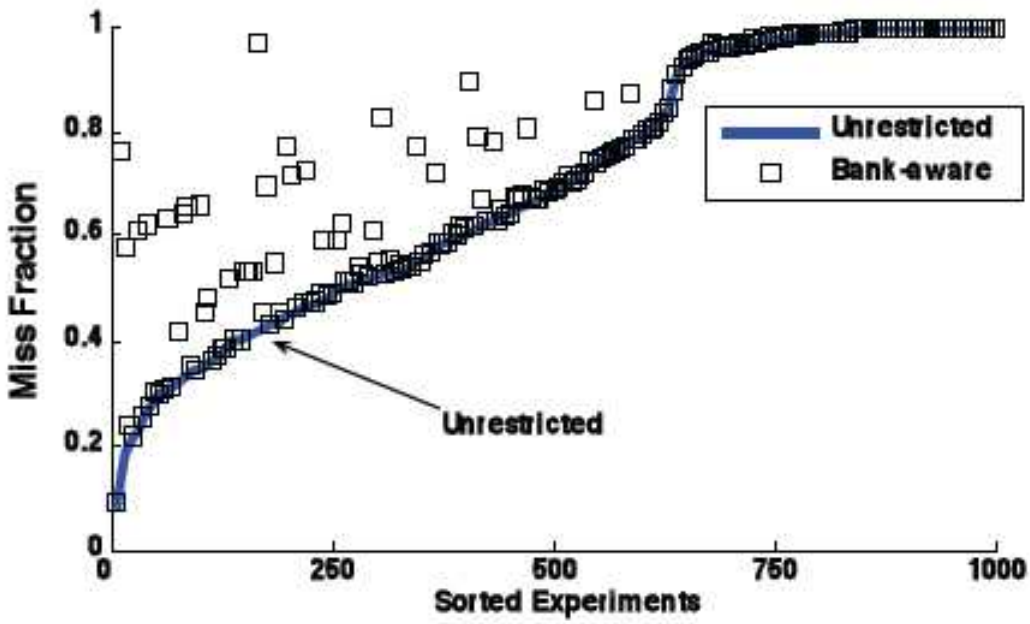

Fig. 7. Relative miss ratio to fixed-share for *Unrestricted*

4.2 Detailed Simulation Results

Eight workload sets are randomly selected from the previous simulations to evaluate the proposed partitioning scheme on the 8-core full system shown in Fig. 1. Fig. 8 and Fig. 9 shows the relative miss rate and CPI of Equal-partitions and Bank-aware partitioning over the simple case of No-partitions. Equal-partitions is equivalent to assigning private cache partitions of equal size to each core. From the figures, both partitioning schemes show a significant reduction in misses and CPI over the simple Nopartitions one, which is a strong indication of the need for partitioning the last level of cache. On average, Bank-aware shows a 70% and 43% reduction in misses and CPI over No-partitions, respectively. Moreover, from

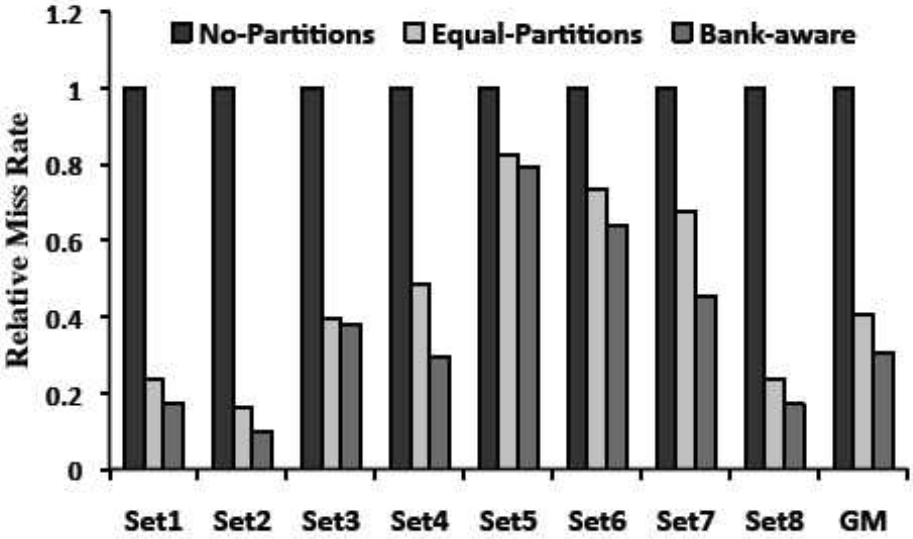

Fig. 8. Relative miss rate of 8-core sets over the no-partitioning scheme

Fig. 8, The Bankaware partitioning scheme shows on average a 25% reduction over simple Equal-partitions. This reduction is inline with the reduction estimated in the Monte Carlo experiment of the previous section.

In addition, Fig. 9 shows that Bank-aware partitioning can achieve an 11% reduction in CPI over the Equal-partitions scheme. Comparing Fig. 8 and 9, some sets of workloads demonstrate a much higher performance sensitivity to misses than others since a reduction on L2 misses does not always result in an equal size reduction in CPI. For example, in Set 1 even though it significantly reduced the overall fraction of misses, that reduction in not translated in CPI gain due to the overall small number of misses in that set and the performance characteristics of the applications that feature the highest miss reduction.

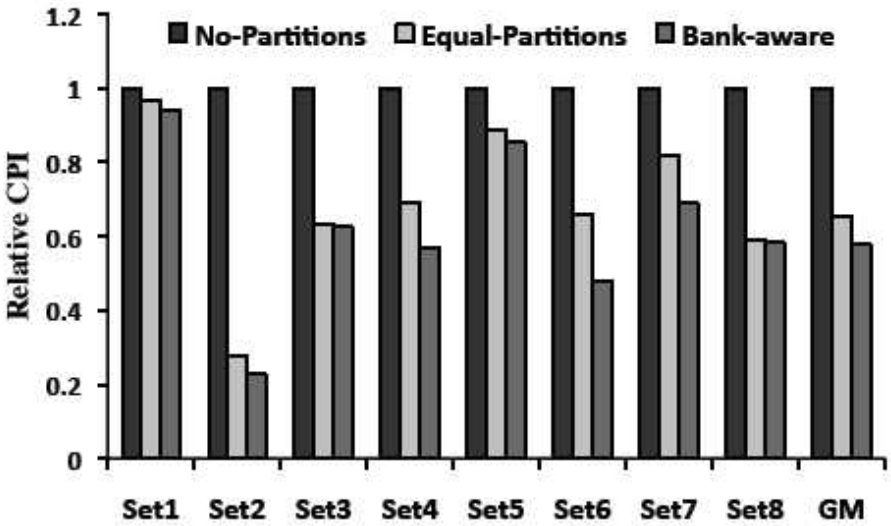

Fig. 9. Relative CPI of 8-core sets over the no-partitioning scheme

5 Conclusion

Shared resource contention in CMP platforms has been identified as a key performance bottleneck that is expected to become worse as the number of cores on a chip continues to scale to higher numbers. Many solutions have been proposed, but most assume either simplified cache hierarchies with no realistic restrictions or complex cache schemes that are difficult to integrate in a real design. Therefore, both approaches could lead to conclusions that are unrealistic when implemented in a real system. This paper highlights the problem of sharing the last level of cache in CMP platforms and motivate the need for a low-overhead cache partitioning scheme that is aware of the banking structure of the L2 cache design. Bank-aware partitioning scheme demonstrates a 70% reduction in misses compared to non-partitioned shared caches, and a 25% miss rate reduction compared to even partitioned (private) caches. Lastly, the proposed scheme managed, on average, the same miss reduction achieved with less realistic proposed Unrestricted schemes that are unaware of implementation restrictions.

References

[1] Jeffrey Stuechelix Dimitris Kaseridis and Lizy K. John. "Bank-aware Dynamic Cache Partitioning for Multicore Architectures". 2009.

[2] R. L. Mattson. et al. "Evaluation techniques for storage hierarchies". *IBM Systems Journal*, 1970.

[3] M. D. Hill and A. J. Smith. "Evaluating associativity in CPU caches". *IEEE Transactions on Computers*, 1989.