International Journal of Engineering, Management, Humanities and Social Sciences Paradigms (Volume 31, Special Issue of January 2019)
ISSN (Online): 2347-601X and Website: www.ijemhs.com

NCID: A Powerful and flexible and Efficient Cache Topology Architecture with Non-inclusive Cache and Encompassing Directory

Mr.Gyana Prakash Bhuyan[1], Ms. Smruti Mishra[2]*

*[1]*Assistant Professor,Dept. Of Computer Science and Engineering, NIT , BBSR*
[2]Assistant Professor,Dept. Of Computer Science and Engineering, NIT , BBSR

gyanprakash@thenalanda.com, smrutimishra@thenalanda.com*

ABSTRACT

Like Intel's Nehalem CPU and AMD's Barcelona processor, chip-multiprocessor (CMP) architectures use multi-level cache hierarchies with private L2 caches per core and a common L3 cache. The inclusion policy—inclusive, non-inclusive, or exclusive—is one of the important design options when creating a multi-level cache hierarchy. There are advantages and disadvantages to both options. An inclusive cache hierarchy, such as the L3 in Nehalem, has the advantage of allowing incoming snoops to be filtered at the L3 cache, but it also suffers from (a) decreased space efficiency due to replication between the L2 and L3 caches and (b) decreased flexibility because it cannot bypass the L3 cache for transient or low priority data. Because the inclusion can begin to impair performance, it also becomes challenging to flexibly reduce L3 cache size (or raise L2 cache size) for various product instantiations in an inclusive L2/L3 cache hierarchy (due to significant back-invalidates). In this research, we propose an unique method to address the shortcomings of inclusive caches while preserving the advantages of snoop filtering. In this article, we introduce NCID, a non-inclusive cache, inclusive directory architecture that keeps tag inclusion in the directory to provide full snoop filtering while allowing data in the L3 to be non-inclusive or exclusive. Afterwards, we go over a variety of NCID-based architecture solutions and policies and evaluate them. Our analysis demonstrates that NCID allows for an adaptable and effective cache hierarchy for future CMP systems and has the potential to considerably enhance performance for a number of critical server benchmarks.

Categories and Subject Descriptors

B.3.2 [**Hardware**]: Design Styles of Memory Structures – cache memories.

General Terms: Performance, Design, Experimentation

Keywords: Directory, Cache

1. INTRODUCTION

In this chip-multiprocessor (CMP) era, it becomes more and more important for architects to design efficient cache hierarchies so that

data access latency within the cache hierarchy as low as possible. To support larger cache sizes with minimized access latency, the off-chip memory accesses can be minimized while still keeping

current CMP cache hierarchy in mainstream processors is moving from a two-level hierarchy to a three level cache hierarchy like Intel's quad-core processor (Nehalem [8]) and AMD's quad-core processor (Barcelona [24]). We believe that future larger scale chip-multiprocessor architectures will also use three level cache hierarchies for scalability and performance.

One of the important aspects of building efficient cache hierarchies is the consideration of the inclusion policy between cache levels. Each level in a cache hierarchy can be inclusive, non-inclusive or exclusive. Intel's L3 cache (in Nehalem), is 8MB in size, and is inclusive of its four 256K L2 caches [8]. On the other hand, AMD's Barcelona [24] employs semi-exclusive cache hierarchies with 512K L2 cache per core and a 2MB shared L3 cache across four cores. One advantage of inclusive cache is its snoop filtering capability, i.e., external snoops from another sockets or chipset do not require L2 lookup if a miss occurs in L3. However since L3 is inclusive of L2, data is duplicated in both L2 and L3, which reduces the cache space efficiency. On the other hand, exclusive caches have better space efficiency but do not have snoop filtering capability. Since both inclusive and exclusive caches have their pros and cons, this paper focuses on defining an alternative cache hierarchy solution that allows both flexibility and efficiency in terms of space allocation and snoop filtering.

(a)

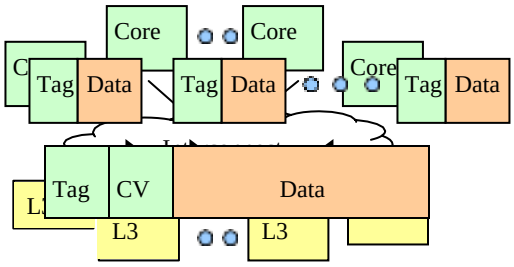
L2

L3

(b)

Figure 1. A 3-Level CMP Cache Hierarchy: (a) Basic Architecture, (b) L2/L3 inclusion policy

Figure 1(a) shows a typical CMP architecture with several cores, each with a private L1 and a private L2 cache. A shared L3 cache can be monolithic or can be implemented as a non-uniform cache



access (NUCA) organization [12]. In a NUCA organization, the L3 is distributed across an interconnect and the access latency to an L3 bank is dependent on its distance from the requesting core. In such an architecture, the latency to a small L2 cache is usually less than ten clock cycles, whereas access to the L3 cache can be several tens of clocks depending on the interconnect design. Figure 1(b) illustrates the organization of the L3 cache and inclusion policy with respect to the L2 caches (L2s). Each cache line in the L3 cache contains the following key components (as illustrated in Fig 1b): (1) the tag containing the higher order bits of the address, the state information and the replacement bits, (2) the directory or core valid bits indicating the potential presence of the cache line in the core's private L1/L2 caches and (3) the 64 byte data for the cache line

In an inclusive cache hierarchy, the L3 cache is enforced to be inclusive of the L2 caches. As a result, every line in the L2 cache can also be found in the L3 caches. When a cache line is evicted from the L3 cache, a back-invalidate is generated to the appropriate L2 caches (based on the core valid bits) and correspondent cache lines are invalidated from the L2 caches before eviction from the L3 cache. The use of an inclusive L2/L3 cache has the benefits of complete snoop filtering, but also has the following limitations:

(a) Replication of data between the L2 and L3 cache reduces the overall space efficiency. For example, a 256K L2 and a 2MB L3 bank can have a replication of up to ~12.5%.

(b) Inflexibility to bypass the L3 cache for transient or low priority data. Research work on adaptive fill policies [17] has recently shown that workloads possess significant amounts of transient data and avoiding allocation of the data in the L3 cache can improve the miss rate significantly. Recent work on quality of service (QoS) [9][10] has also shown that it is desirable to allow very low priority data to bypass the L3 cache. However, bypassing the L3 cache cannot be allowed in an inclusive hierarchy, thereby requiring a minimum occupancy in L3 cache even for low priority or transient data.

(c) Inflexibility in terms of reducing L3 cache or increasing L2 caches. In an inclusive cache hierarchy, it is important to maintain a size ratio of L3 cache to L2 cache of about 8:1. As a result, increasing the L2 cache requires increasing the L3 cache appropriately as well. However, with significant die budget constraints, it is desirable to enable a cache hierarchy where L2 caches can be significantly larger and L3 caches are smaller in size. In such scenarios, (semi-)exclusive hierarchies are much more appropriate.

(d) A distributed shared L3 cache generally employs hashing techniques to ensure that data access across the banks is uniform (i.e. no particular bank becomes a hotspot due to the access pattern). However, in such a configuration, private and shared data accesses for any core incur the same average bank access latency. If private data can be placed closer to the cores and shared data can be placed in a central location, it is possible to improve access latency significantly and thereby improve performance.

To solve the above limitations, we propose a novel cache hierarchy approach called NCID -- Non-inclusive Cache, Inclusive Directory. NCID enables data to be non-inclusive or semi-exclusive or exclusive while maintaining tag inclusion to stillkeep the snoop filtering capability. The contributions of this paper are the following:

- We propose a new multi-level NCID cache hierarchy that enables flexibility and cache space efficiency, and describe its detailed hardware implementation.
- We demonstrate that NCID can reduce on-die interconnect traffic significantly while still maintaining the performance benefits of non-inclusive caches.
- We propose a selective allocation scheme based on NCID support to optimize workloads that have a significant amount of transient data.
- We show that using NCID, exclusive cache hierarchies can be used while still maintaining the snoop filtering capability.
- We also show that NCID can support flexible data placement in a distributed shared L3 cache by building a hybrid L2-L3 cache organization.

The rest of this paper is organized as follows. Section 2 describes our proposed NCID architecture and discusses the implementation considerations for NCID architectures. Section 3 outlines a range of potential NCID architecture and policy options. In Section 3, we also describe the methodology used to evaluate NCID options. Section 4 presents the results from our detailed simulations and analyzes performance implications of NCID architectures. Section 5 covers a comparison to related work. Section 6 summarizes our key contributions and presents a direction for future work in this area.

2. NCID ARCHITECTURE

In this section, we describe our proposed NCID architecture and compare/contrast it with previous hierarchies. We will also discuss the NCID implementation options along with associated hardware cost.

NCID Basics

Figures 2(a) – (c) summarize the cache hierarchy choices and the pros/cons for each. As shown in the figure, the inclusive cache hierarchy provides the benefits of complete snoop filtering, but requires full data replication supported by back-invalidate messages. Since most incoming snoops will miss in the L3, it is guaranteed that the L2 caches do not need to be snooped since the L3 is inclusive. The non-inclusive cache hierarchy is desirable because it reduces the need for replication. In this hierarchy, evictions from the L3 do not require back-invalidations to be sent to the L2 caches. However, at the same time, it has the drawback of requiring incoming snoops that miss in the L3 to be sent to all of the L2 caches since it is not guaranteed that they will not be in the L3. The figure also illustrates the exclusive cache hierarchy, where the space usage is maximized by ensuring that any data in the L2 is not replicated in the L3 and vice-versa. It should be noted that there are a plethora of hybrid policies that are possible between inclusive caches and exclusive caches which are referred to as semi-inclusive or semi-exclusive caches.

Figure 2(d) illustrates the proposed NCID architecture. The key objective of NCID is to decouple tag and data management in a multi-level cache design. The L3 data is not required to be inclusive while L3 tags are inclusive of all L2 tags. This is accomplished by maintaining a larger number of tags than data entries in the L3 cache. As shown in Figure 3, the number of tag and directory array entries is larger than that of the data array, thus some lines in L2 have their duplicated tags in L3 but no data is required to exist in L3. In this paper, we call the extended tags in the directory array as directory cache and the traditional cache array as L3 cache. Since the space required for tags/directory is

much smaller than the space required for data, this is a cost-effective approach to maintain tag inclusion for snoop filtering but relax data inclusion and allow either non-inclusive data allocation or exclusive data allocation. To demonstrate that NCID allows the L3 cache to be highly flexible, let us now review one possible flow to understand the NCID-based hierarchy in a more detailed fashion.

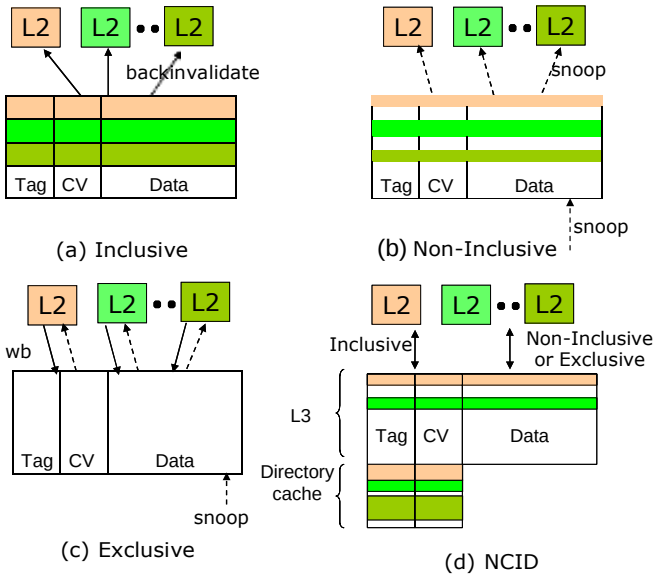


Figure 2. NCID Architecture Overview

• Upon a memory access that misses in all levels of cache, the returning data is always provided to the L2 and can be allocated in the L3 cache in one of the following ways: (a) the missing address and core valid bits are installed in the main tag/directory and the data is also stored in the data arrays or (b) the missing address and core valid bits are installed in the extended tag/directory and the data is not stored in the data arrays. The policy for deciding between these two options is kept flexible and will depend on whether inclusive, non-inclusive or exclusive policies are desired.

• Upon an eviction from the L3 cache, the eviction can result in the following different cases: (a) the address being evicted from the L3 cache also has data associated with it in the data arrays. In this case, it is possible to evict the data and retain the address in the directory cache. (b) the address being evicted from the L3 cache does not have data associated with it. In this case, back-invalidates are generated to the L2 cache (based on the core valid bits in the directory). Once the L2 caches have evicted the data, the address eviction is completed at the L3 cache.

• Upon an incoming snoop either from another socket or a lookup from another core, the L3 and the directory cache is consulted to find the location of the line in one or more L2 caches. If the line does not exist, then the request is sent to memory. If the line exists (but no data exists in the L3 cache), the request is forwarded to a L2 cache(s) that may have a cached copy.

Hardware Implementation

NCID-based architectures can be implemented in one of two ways:

(a) Directory cache as a separate structure: In this approach, as shown in Figure 3 (a), we design an independent associative tag/directory structure to maintain additional addresses with no associated data. This extended tag/directory structure is exclusive with respect to the main tag/directory. It can have a different indexing. Migration of addresses occurs between these structures when the main tag/directory structure evicts an entry (which is moved to extended directory) or when data for an address in the extended tag/directory structure is requested by a core and it does with the key requirement still being maintaining tag inclusion with respect to the L2 caches.

(b) Directory cache as additional ways in the main tag structure: In this approach, we can design the directory cache entries as just additional ways in each set of the main tag structure. In doing so, it should be noted that some ways are associated with data and some are data-less. Differentiating these two different subsets of ways can be achieved in a static scheme or in a dynamic scheme. For a static scheme as shown in Figure 3 (b), specific ways are predetermined to be data-less. For a dynamic scheme as shown in Figure 3 (c), any given way can be data-less as required at runtime. This requires additional bits to be maintained to associate the data with the ways. Overall, this approach of increasing ways in each set avoids the complexity of maintaining an additional directory structure. However, both the approaches can achieve the same functionality.

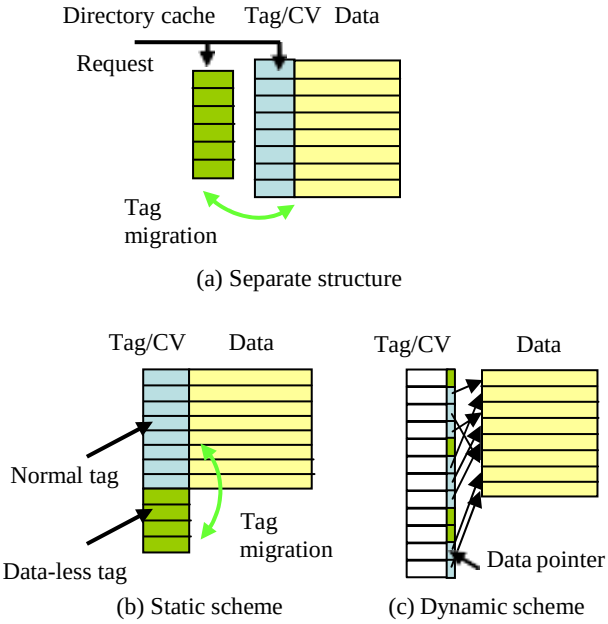


Figure 3. Implementation Options for NCID Directory Cache

Hardware Cost of NCID

The additional die area cost of implementing NCID is the size of the extended tag/directory required to maintain inclusive tags. The size of additional tag/directory entries in the NCID architecture is a function of the coverage it provides. We define *coverage* to be the data array size that a tag array corresponds to. For example, 1MB cache size needs about 80KB for its tag array, so we say that 80KB of directory cache can provide a coverage of 1MB. Based on our estimation, (assuming 64 bits of the physical memory address, 64B

of cache line size, 8 bits for core valid bits, 2 bits for states), Figure 3 shows the size (in Kbytes) of the directory cache (per L3 cache bank) as a function of coverage and L2 size. The x-axis shows the L2 cache size and bars denote the coverage. The coverage of an inclusive directory cache is generally required to be greater than 2X in order to ensure limited number of back-invalidates and negligible impact on miss rate [1]. However, if replacement hints are implemented, then the size of the directory cache can be reduced down to 1X as well. We can see that to cover 256K L2 cache, a directory cache with 2X of coverage requires about 40KB. Compared to a baseline L3 bank size of about 2M, this adds only 2% overhead.

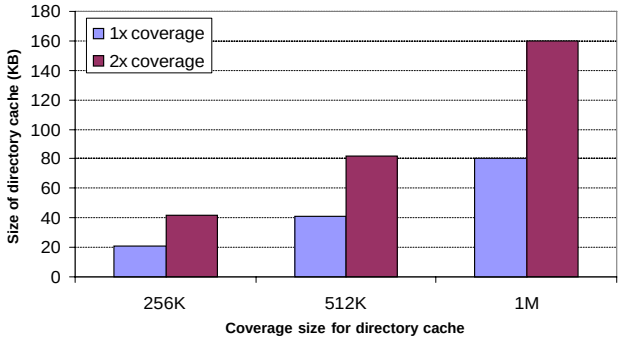


Figure 4. Directory Size (main + extended) as a function of cache size and coverage

3. NCID ARCHITECTURE OPTIONS AND EVALUATION METHODOLOGY

To show the effectiveness of NCID architectures, we evaluated several NCID architecture options and policies. The NCID architecture options can be placed into the following three broad categories: (a) small L2, large L3 options, (b) larger L2, smaller L3 options and (c) hybrid L3 options. The first category (where inclusive or non-inclusive policies are more applicable) evaluates the utility of adding NCID functionality into a cache hierarchy where the L2 size is small whereas each L3 bank size is much larger. The second category (where exclusive policies are more applicable) evaluates the benefits of NCID when the L2 cache is increased significantly and the L3 is correspondingly reduced to maintain a constant die budget. The last category evaluates the benefits of NCID to support flexible data placement in a distributed shared L3 caches. All these options are compared against the base inclusive cache hierarchy. As shown in Figure 5, we enumerate the configurations being evaluated as follows (note that L3 in the figure means L3 data only, and DIR means directory cache plus L3 tag):

(1) Base NCID: directory cache of 2X coverage of L2 caches added to allow for a non-inclusive data allocation, but inclusive tag allocation.

(2) NCID to support selective allocation policy to address transient data: use probability to allocate some of the lines to the L3 cache and some of the lines to the directory cache.

(3) NCID to support QoS policy: Here, we show how the NCID architecture allows low priority data to bypass L3 cache and therefore improve cache efficiency as well as provide QoS.

(4) NCID to support large L2, Smaller L3 configurations with non-inclusive and exclusive hierarchies: Here, the L2 sizes are increased from 256K to 2M (doubling along the way) and the L3 sizes are correspondingly reduced from 3M to 1M in order to the keep the total die budget constant.

(5) NCID to support hybrid L3 configurations to improve proximity to private data in the distributed shared L3: Here we study the use of NCID to partition the L3 into a private and shared area and allow private data to be cached closer to the core to reduce latency.

We use a trace-driven platform simulator called ManySim [26] to evaluate NCID options in CMP platforms. ManySim simulates the platform resources with high accuracy, while abstracting the core to optimize for speed. ManySim contains a detailed cache hierarchy model, a detailed coherence protocol implementation, an on-die interconnect model (simulating a bi-directional ring) and a memory model that simulates the maximum sustainable bandwidth specified in the configuration. In order to evaluate NCID options, ManySim was modified to add the control for directory cache and required protocol changes as well as various policies supported. The simulated CMP architecture is similar to the one shown in Figure 1. There are eight cores, with private L1 and L2 caches. All cores share a distributed L3 cache consisting of 8 banks. Table 1 summarizes the simulation configurations.

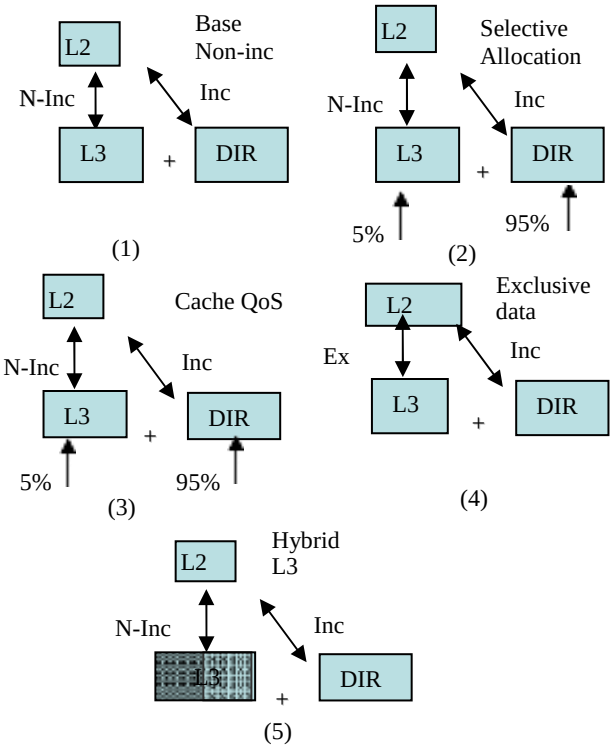


Figure 5. NCID configurations under Evaluation

Table 1. Simulation configurations

Parameters	Values
Core	2GHz
L1 I/D cache	32 Kbytes, 4-way, 64-byte block
L2 cache	8-way, 64-byte block
L2 cache access time	10, 11, 12 and 13 cycles for 256K, 512K, 1M, and 2M respectively
L3 cache	Varied (e.g. 1MB banks, 16-way, 64-byte)
L3 cache access time	28, 29, 30 cycles for 8M, 16M, 24M respectively
Interconnect bandwidth	128GB/s
Memory access time	400 cycles
Memory bandwidth	16GB/s

We use several commercial benchmark traces which include TPC-C [22], TPC-E [23], SPECjbb2005 [20], SAP SD/2T [18]. Each trace is a long bus trace collected on Intel Xeon MP platform with 8 hardware threads running simultaneously with the L2 cache disabled. The traces include both instruction and data accesses, synchronization and inter-thread dependencies.

4. RESULTS AND ANALYSIS

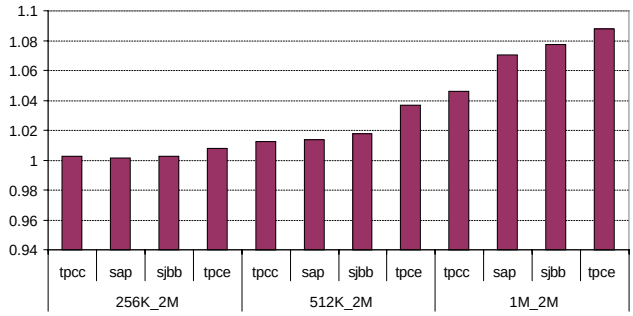
In this section, we present an in-depth evaluation of the performance impact of NCID configurations. As we use various L2/L3 cache size configurations, we use "X_Y", where X is the L2 cache size per core and Y is the L3 cache size per core (note that although the L3 is shared, the L3 size specified is averaged by the number of cores). For example, 256K_2M means 256K L2 and 2M L3 per core (or 16M L3 cache in total because there are 8 cores in the configuration). We will show the performance speedup of using NCID architecture compared to the base inclusive cache hierarchy.

NCID Benefits for Non-inclusive Hierarchies

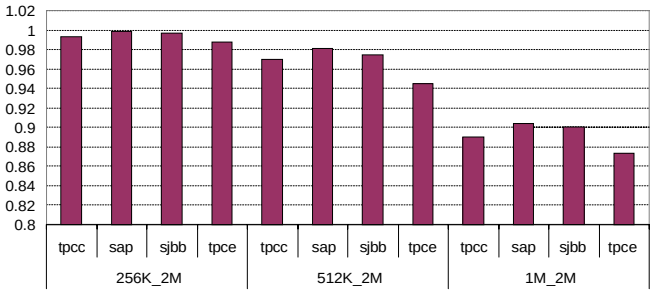
Figure 6 compares the base NCID hierarchy with non-inclusive data to the base inclusive cache configuration. The directory cache coverage is maintained at 2X the total size of L2 caches in the configuration. We choose 3 configurations: (a) 256K_2M, (b) 512K_2M, and (c) 1M_2M, where we fix the total L3 cache size as 16MB while increasing the L2 cache size.

Figure 6 (a) shows the speedup of using NCID over the base inclusive cache. As shown in the figure, the NCID configuration provides equal or better performance as compared to the base inclusive hierarchy. When the size ratio between L2 and L3 is high (256K_3M bars), NCID does not provide additional benefit because inclusion does not have a significant drawback. However, as larger L2s are considered for latency reduction, the size ratio between L2 and L3 is bound to shrink. In this case, the NCID architecture provides a better speedup. In the last configuration where 1M_2M is used, the speedup is up to 9%.

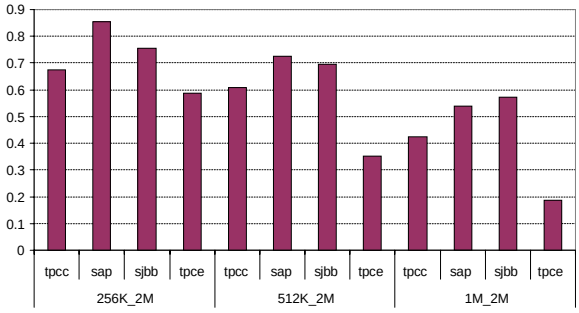
Figure 6 (b) shows the corresponding miss rate reduction, which matches the data for speedup. Figure 6 (c) shows the number of snoop invalidation reduced by NCID. We can see that even in 256K_3M configuration where NCID performs the same as the base inclusive cache, the number of snoop invalidations is reduced significantly. Although this will not affect performance as the simulated interconnect model has enough bandwidth, this has the potential to reduce the power consumption on interconnect, especially if a network-on-chip is employed for future CMP platforms.



(a) NCID Speedup



(b) NCID L3 Miss Rate Reduction



(c) NCID Reduction in Snoop Invalidations

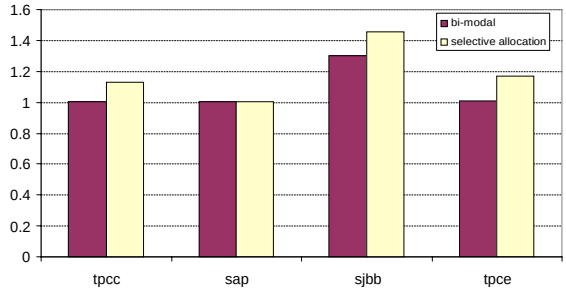
Figure 6. NCID Benefits (compared to inclusive cache hierarchy)

NCID Benefit for Transient Data Optimizations

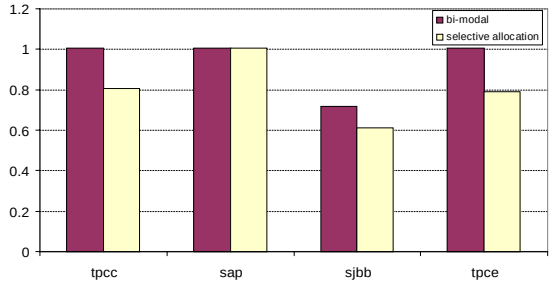
Previous research has shown that there exists a significant amount of transient data in many workloads. For example, Qureshi et al [17] proposed an adaptive fill policy to improve the cache efficiency and performance.

The adaptive fill policy introduces bi-modal fills, where 5% of the lines are filled normally as Most Recent Used (MRU) lines, and 95% of the lines are filled vulnerably as Least Recent Use (LRU) lines. They showed that employing set dueling with some leader sets to choose normal fill and valuable fill dynamically in the run time can allow cache performance to improve significantly. They have also shown that this works better in a non-inclusive cache than in an inclusive cache hierarchy because the vulnerable lines can cause significant back-invalidations if the cache hierarchy is inclusive. With NCID architecture, we evaluate a variant of bi-modal fill policy (similar to selective allocation policy [9]), where we allocate 5% of the allocations normally as MRU in L3 cache and the remaining 95% vulnerably in the directory cache (which is data-less).

We compare the base inclusive cache without transient data optimizations to (a) inclusive cache with bi-modal fills and (b) NCID cache with selection allocation. The configuration chosen for this study is a 256K_1M configuration. Figure 7 shows the speedup and miss rate reduction compared to the base inclusive cache. The NCID approach the best performance for three of the four workloads. Especially for SJBB, the performance is increased by 45% (it should be noted that we validated this by characterizing the SPECjbb cache usage patterns independently using a cache simulator as well). For the SAP workload, we find that all three approaches result in equal performance (because the set sampling approach ensures that the cache always runs with the normal fill policy instead of the selective or bi-model fill policy).



(a) Speedup



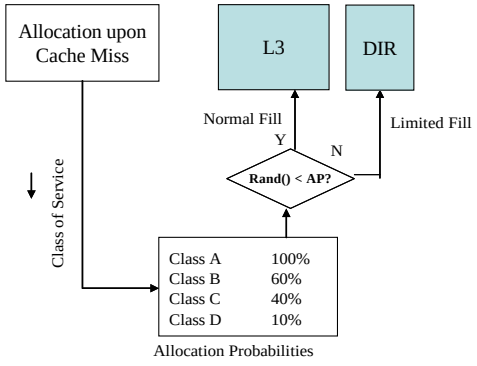
(b) Miss rate reduction

Figure 7. Benefits of NCID with Selective Allocation (256K_1M configuration)

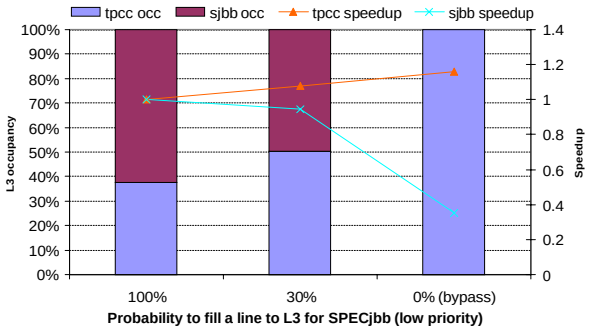
Using NCID to Improve QoS

Previous researchers have proposed QoS mechanisms in cache [9][10] to address contention for shared resources between multiple applications of differing priority running simultaneously on a platform. Cache QoS allows for the OS/VMM to assign classes of service to applications or virtual machines. Each class of service is associated with maximum cache space utilization that is enforced either on allocation or replacement. However, with an inclusive cache where allocation cannot be bypassed, even the class with the lowest priority has to be assigned to certain amount of L3 cache. For example, if the low priority application is running on one core that has 256K private L2, the minimum L3 cache space allocation for this application needs to be at least 512K (2X) in order to ensure that the L2 is still usable. Otherwise, it makes the private L2 under-utilized due the back invalidations. In addition, some low priority applications may need only a small amount of cache, i.e., L2 is enough, and bypassing L3 can provide more space to other high priority applications. With the NCID architecture, L3 bypass is easy to achieve and this allows us to enhance the QoS support in cache.

Here we show an example of employing probabilistic selective allocation per class of service in L3 cache to provide cache QoS enabled by NCID architecture. As shown in Figure 8(a), each class of service is assigned X% probability. When a new line is filled into L3 cache, a random number is generated. If this number is less than X%, it is allocated in L3 cache; otherwise it is allocated into the directory cache. For class D in the example (0% probability), it implies that all allocations for this class should bypass the L3 cache. Figure 8(b) shows the implications of employing different allocation probabilities for SPECjbb as a low priority workload running simultaneously with TPCC as a high priority application in 256K_2M configuration.



(a) QoS Classes of service and probability-based allocation



(b) NCID Benefits via L3 bypass

Figure 8. NCID usage for improving Cache QoS features

TPCC is always assigned 100% allocation probability, whereas the probability for SPECjbb is varied from 100% down to 0%. The bars show the L3 occupancy for these two applications. Note that 30% probability does not equal to 30% of cache occupancy.
Instead we see that SPECjbb still gets about 50% of the L3 cache. The lines show the speedup or degradation. We can see that when L3 bypass is employed, TPCC takes the whole L3 cache, and it achieves about 16% speedup. For SPECjbb on the other hand, its performance is degraded significantly. If the low priority workload is streaming in nature and does not use the L3 cache as much, we expect that its performance will maintain the same. This architecture will be more desirable in a heterogeneous CMP architecture, which has general-purpose cores and programmable hardware engines (like a GPU for graphics processing), because CPU workloads are generally more cache-friendly, and the graphics engines are streaming in nature and generally do not require cache to improve performance.

Using NCID to Support Exclusive Data Hierarchies

Some CMP architectures (e.g. AMD Barcelona) find it desirable for the cache hierarchy to contain larger L2s and proportionately smaller L3s. This is likely because larger L2s may reduce the stall time in the cache hierarchy and smaller L3s are required to ensure that the cache die budget remains constant for cost purposes. For example, instead of a 256K L2 and a 3M L3 bank, a cache hierarchy could consist of 1M L2s and 2M L3 banks. In such scenarios it is important to avoid inclusion in the cache hierarchy since it affects the miss rate significantly (as a result of replication and back-invalidates). With the NCID architecture, we can support exclusive cache hierarchies and complete snoop filtering. In an exclusive cache hierarchy, it is sufficient to provide 1x directory coverage in NCID because victims from L2 are sent to the L3.

For the sake of comparison, we choose 4 configurations: 256K_3M as the base inclusive cache configuration, 1M_2M, 1.5M_1.5M and 2M_1M as the configuration with exclusive data and inclusive tag. These configurations are chosen to make the total size of L2 and L3 roughly the same. Figure 9 shows the hit/miss distribution for four server workloads. In this evaluation, we also added a "MIX" workload to simulate a server consolidation by running all four workloads simultaneously with each one running on two cores. We breakdown the total access to four parts: L2 hit (hits to local private L2), L2 remote hit (hits to other private L2), L3 hit (hits to L3), and miss to memory (the misses to the memory). We can see that the percentage of L2 hit is increased significantly as the L2 size is increased. For example, SPECjbb2005 has only 40% of its accesses hit in the L2 in the base inclusive hierarchy, but has close to 70% of its accesses hit in the L2 in the case where the L2 size is 1M and the L3 size is 2M.

Figure 10 shows the speedup of the 3 configurations compared to the base inclusive cache. We can see that the performance is increased by up to 7.5%. It is observed that SPECJBB and MIX favor 2M_1M configuration, whereas the other three workloads favors 1.5M_1.5M configuration. This is because both SPECJBB and MIX have little sharing and increasing L2 cache size reduces the access latency significantly. However, for the other three workloads, there is significant sharing in the workload. Increasing the L2 cache size ends up as replicating contents across multiple L2s. In doing so, the overall cache space is affected negatively as compared to the base case. We find that the L2 miss rate increases for these workloads and therefore the performance does not improve significantly. As the overall performance is a function of both access latency and the miss rate, 1.5M_1.5M becomes a sweet point for these workloads.

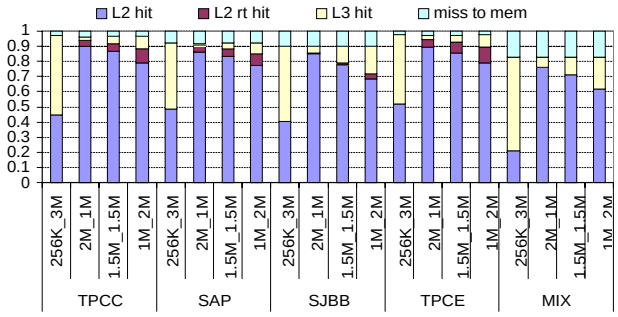


Figure 9. Hit and miss distribution of using NCID to support exclusive data

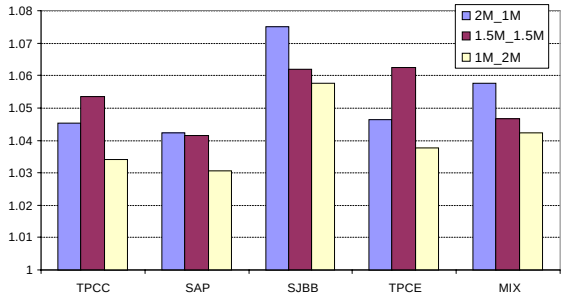


Figure 10. Speedup of using NCID to support exclusive data

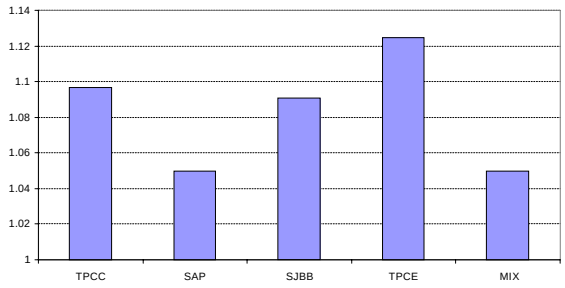


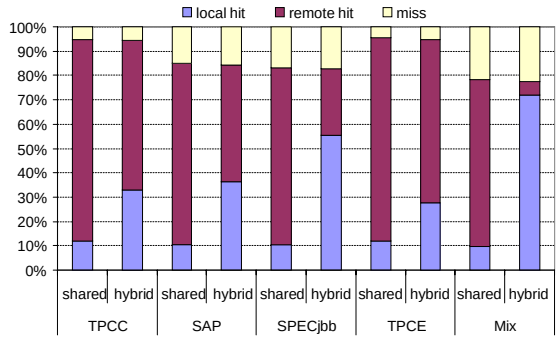
Figure 11. Speedup of limiting the shared data in L2 to be 20% for 1M_2M configuration

To increase L2 hit as well as reducing the miss rate, we introduce a mechanism to limit the amount of shared data in the L2s. A counter is maintained in L2. If the shared data is exceeding the limit, only the shared data is replaced. Figure 11 shows the benefits of limiting shared data to 20% of the L2 cache size. We can see that by limiting the shared data occupancy, the performance is increased by up to 12%.

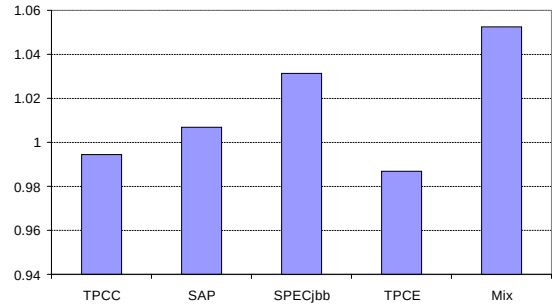
Using NCID to Support Hybrid L3

Our final evaluation of NCID was to support hybrid L3 caches. A basic distributed shared L3 cache does not allow flexible data distribution across the L3 banks. The address hashing mechanism is static -- essentially distributing addresses across the L3 banks with no regard to which core is accessing which cache line. In order to support a hybrid L3 cache that allows private data to be allocated in a L3 bank closer to the core, we divided the L3 into two

partitions using way partitioning [9]: (a) private partition and (b) shared partition. While there are a lot of schemes that can be implemented on a hybrid cache, this paper evaluates one scheme to show the flexibility of the NCID architecture. The hybrid NCID scheme works as follows: when a core misses the L2/L3 and request a line from the memory, the data for that line is allocated into the private partition of the L3 bank nearest to the core. The address of the cache line is also allocated in the NCID directory in order to keep track of the copies in the private L3 partitions. When the line is evicted from this private partition, it is migrated to the shared partition of a L3 bank that is determined by the static address hashing mechanism. In addition, when a line becomes shared, it is also migrated to the shared partition. The migration is not on the critical path, so the performance is not affected.



(a) Hit/miss distribution



(b) speedup

Figure 12. Benefits of using NCID for a hybrid L3

Figure 12 shows the data for hybrid L3s made possible by NCID and compares it to the base shared cache. The base configuration is 256K_3M for shared cache. For hybrid cache, we essentially use 16 ways out of the 24 ways in a 3M L3 bank for the private partition (i.e. 2M private partitions) and 8 ways out of the 24 ways in a 3M L3 bank for the shared partition (1M shared partitions). Employing the hybrid cache has two implications: (a) the average access latency is affected due to local hit (a line is found in the local bank) and remote hit (a line is found in a remote bank), and (b) the overall miss rate to memory. Figure 12 (a) shows that hit/miss distribution for all the four workloads and the consolidated workload. We can see that for workloads with little sharing like SPECjbb2005 and Mix, the local hit is increased significantly. For example, the local bank hits increase from 12.5% in the shared case to about 70% for the hybrid case. For workloads with sharing, the improvement is moderate (e.g. from 12.5% in shared cache to 33% in hybrid cache for TPC-C). Similar to the previous subsection the performance is a function of both access latency and miss rate. So Figure 12 (b) illustrates the speedup compared to the base shared case. It is observed that both workloads that have a lot of sharing like TPCC and TPCE get degradation about less than 1.3%, whereas other workloads get improvement of up to 5%. We believe that the improvement is relatively small in our configuration because of the small-scale nature of the CMP evaluated. With large-scale CMP architectures where the latency difference between private and shared banks is higher, the performance improvement is expected to be much more substantial.

5. RELATED WORK

Cache memories [19] have been studied over several decades now. The implications of the inclusion property in a multi-level cache hierarchy were first studied by Baer and Wang in [1]. Przybylski et al [16] subsequently studied performance-optimal design of multi-level caches. Jouppi et al [11] studied the implications of two-level on-chip caching with specific focus on exclusive hierarchies. Over the last two decades, microprocessors have been designed with inclusive caches (such as Intel's latest Nehalem processor [8]) as well as exclusive caches (such as AMD's Barcelona [24] processor). Both inclusion and exclusion has its pros and cons and the focus of this paper was on identifying a solution (NCID) that combines the benefits of these by decoupling data and tag management. The Piranha research prototype from Compaq [2], comes close to an NCID instance, by implementing a non-inclusive shared L2 with copies of L1 tags for snoop filtering.

In the context of CMP cache hierarchies, there have been several recent papers that look at policies for better management of data within a single level of cache as well as across the hierarchies [3][4][5][6][12][15][25]. Kim et al [12] introduce the concept of non-uniform cache architecture. In this paper, we adopt a distributed shared L3 cache that essentially follows the NUCA organization. We show how a non-inclusive cache with inclusive directories can be implemented in a NUCA L3. In addition, researchers have studied (full and selective) replication of victim data in neighboring caches to reduce the number of memory accesses. In addition, researchers have attempted to find mechanisms to manage private and shared data more effectively in order to improve cache efficiency. In this paper, we show how a hybrid private-shared cache can be implemented effectively using a NCID architecture. We show that the NCID architecture allows for flexible placement of private and shared data such that private data is placed in L3 banks closer to the core, whereas shared data is placed in a more central location.

Researchers have also tried to improve the ability to handle transient data [17] and low priority data [9][10] in shared caches. In [17], Qureshi et al proposed the use of adaptive insertion policies to minimize the impact of transient data, whereas Iyer et al study the impact of restricting low priority data in cache to reduce its interference impact on the performance of higher priority applications [9][10]. In both cases, it is desirable to allow last-level cache bypass for low priority data. However, an inclusive cache hierarchy does not allow the ability to bypass the cache. In this paper, we revisited the handling of transient and low-priority data in the NCID architecture where the directory cache allows for an efficient bypass mechanism for data allocation while retaining a copy of the tag for snoop filtering.

Overall, we believe that NCID introduces a novel approach to designing cache hierarchies by decoupling data and tag management. Although not related at all, this can be viewed as

analogous to the decoupling of performance and correctness achieved by Token coherence [13].

6. CONCLUSIONS AND FUTURE WORK

In this paper, we revisited the inclusion property of multi-level cache hierarchies (inclusive, non-inclusive and exclusive). Since these three inclusion policies are known to have benefits and drawbacks in terms of space efficiency and snoop filtering capability, we focused on identifying an alternative cache hierarchy approach that allows for flexibility in data placement and space efficiency, while retaining the ability to provide snoop filtering. We described our proposed architecture (NCID – non-inclusive cache, inclusive directory) that addresses the above problem by decoupling data and tag management. We showed that this decoupled approach of NCID enables the following benefits: (a) it avoids data inclusion (enabling non-inclusive and exclusive data policies) in the cache hierarchy and (b) it maintains tag inclusion in the directory so that complete snoop filtering is still maintained. We described the implementation considerations for NCID and described how an extended directory can be achieved by either implementing a separate extended tag structure or extending the number of ways in each set of the main tag structure.

We also described a range of NCID-based architecture options (base non-inclusive, selective allocation in NCID, QoS-aware NCID, exclusive NCID and hybrid private-shared L3 with NCID) that provide incremental to significant benefits (ranging from 2% to 45%) for future CMP processors. We showed that implementing a non-inclusive cache with transient data policies such as bi-modal fills or selective allocation improves performance by as much as 45%. In addition, the NCID architecture allows low priority data to bypass the L3 cache for QoS. Then, we presented exclusive policies based on NCID that allowed for maximizing space efficiency in the cache hierarchy and minimizing data access latency through larger L2 caches. Last but not least, we presented flexible handling of private/shared data in the NCID architecture to improve cache efficiency (by limiting replication in the L2 caches) and provide better proximity to private data as compared to shared data per core.

In summary, we believe that implementing an NCID-based architecture in future CMP platforms has significant value since it maximizes space efficiency and provides flexibility by decoupling snoop filtering (tag inclusion) from flexible data placement (data non-inclusion). Although we evaluated quite a few NCID policy options in this paper, we expect that there are additional policies related to victim caching, use of replacement hints, etc that can be exploited to optimize the NCID architecture significantly. We also believe that a decoupled tag structure can be further reduced in size by enabling coarse-grain structures. We also expect that application of NCID architectures to lower-level cache hierarchies (between L1 and L2 for instance) may be valuable.

REFERENCES

[1] J. L. Baer and W .H. Wang. “On the inclusion properties for multi-level cache hierarchies,” Proceedings of the 15th Annual International Symposium on Computer Architecture, page 73-80, 1988.

[2] L. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 282–293, June 2000.

[3] Bradford M. Beckmann, Michael R. Marty, and David A. Wood, “ASR: Adaptive Selective Replication for CMP Caches,“ 39th International Symposium on Microarchitecture (MICRO), December 2006.

[4] Bradford M. Beckmann and David A. Wood, “Managing Wire Delay in Large Chip-Multiprocessor Caches,” 37th International Symposium on Microarchitecture (MICRO), December 2004.

[5] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. Carter. Interconnect-Aware Coherence Protocols for Chip Multiprocessors. In Proceedings of ISCA-33, June 2006.

[6] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. “Optimizing replication, communication and capacity allocation in CMPs,” In the 32nd ISCA, pages 357–368, June 2005

[7] S. Ghai, J. Joyner, and L. John. Investigating the Effectiveness of a Third Level Cache. Technical Report TR-980501-01, Laboratory for Computer Architecture, The University of Texas at Austin, May 1998.

[8] Intel® Microarchitecture (Nehalem), http://www.intel.com/technology/architecture-silicon/next-gen/

[9] R. Iyer, “CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms,” 18th Annual International Conference on Supercomputing (ICS’04), July 2004.

[10] R. Iyer, L. Zhao, et al., “QoS Policies and Architecture for Cache/Memory in CMP Platforms”, the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), June 2007

[11] N. P. Jouppi and Steven J. E. Wilton, “Tradeoffs in Two-Level On-chip Caching,” Proceedings of the 21st annual international symposium on Computer architecture 1994 , Chicago, Illinois, United States

[12] C. Kim, D.C. Burger, and S.W. Keckler, “NUCA: A Non-Uniform Cache Access Architecture for Wire-Delay Dominated On-Chip Caches,” IEEE Micro Special Issue (Top Picks in Computer Architecture), Nov/Dec 2003.

[13] M. M. K. Martin, M. D. Hill, and D. A. Wood. “Token coherence: Decoupling performance and correctness,” In the 30th ISCA, pages 182–193, June 2003.

[14] Michael R. Marty and Mark D. Hill, “Virtual Hierarchies,” IEEE Micro Special Issue: Micro's Top Picks from Microarchitecture Conferences, January-February 2008.

[15] N. Muralimanohar, R. Balasubramonian, “Interconnect Design Considerations in Large NUCA caches,” Proceedings of the 34th annual international symposium on Computer architecture 2007 , San Diego, California, USA

[16] S. Przybylski, M. Horowitz, and J. Hennessy, “Characteristics of performance-optimal multi-level cache hierarchies,” In Proceedings of the 16th Annual international Symposium on Computer Architecture (Jerusalem, Israel). ISCA '89. ACM, New York, NY, 114-121. DOI= http://doi.acm.org/10.1145/74925.74939

[17] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr., and J. Emer. "Adaptive Insertion Policies for High-Performance Caching", in the International Symposium on Computer Architecture (ISCA), 2007

[18] Sap America Inc., "SAP Standard Benchmarks," http://www.sap.com/solutions/benchmark/index.epx

[19] A. J. Smith, "Cache Memories," ACM Computing Surveys, Vol.14, No.3, September 1982.

[20] SPECjbb2005, http://www.spec.org/jbb2005/

[21] E. Speight, H. Shafi, L. Zhang, and R. Rajamony, "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors," Proceedings of the 32nd annual international symposium on Computer Architecture 2005.

[22] The TPC-C Benchmark, http://www.tpc.org/tpcc/

[23] The TPC-E Benchmark, http://www.tpc.org/tpce/

[24] B. Waldecker, "AMD Quad Core Processor Overview", http://www.amd.com/us-en/Processors/TechnicalResources/0,,30_182,00.html

[25] M. Zhang and K. Asanovic. "Victim replication: Maximizing capacity while hiding wire delay in tiled CMPs", In the 32nd ISCA, pages 336–345, June 2005.

[26] L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni and D. Newell, "Exploring Large-scale CMP Architectures using ManySim", IEEE Micro, July/August 2000