The Case for a Scalable Coherence Protocol for Complex On-Chip Cache Hierarchies in Many-Core Systems

Lucía G. Menezo Valentin Puente Jose Angel Gregorio

University of Cantabria
Santander, Spain
{gregoriol, vpuente, monaster}@unican.es

Abstract—This paper introduces a new coherence protocol that addresses the challenges of complex multilevel cache hierarchies in future many-core systems. In order to keep coherence protocol complexity bounded, inclusiveness is required to track coherence information across levels in this type of systems, but this might introduce unsustainable costs for directory structures. Cost reduction decisions taken to reduce this complexity may introduce artificial inefficiencies in the on-chip cache hierarchy, especially when the number of cores and private caches size is large. The coherence protocol presented in this work, denoted MOSAIC, introduces a new approach to tackle this problem. In energy terms, the protocol scales like a conventional directory coherence protocol, but relaxes the shared information inclusiveness. This allows the performance implications of directory size and associativity reduction to be overcome. Contrary to the common belief that inclusiveness is inescapable when attempting to maintain complexity constrained, MOSAIC is even simpler than a conventional directory. The results of our evaluation show that the approach is quite insensitive, in terms of performance and energy expenditure, to the size and associativity of the directory.

Keywords – coherence protocol; multi-core; CMPs

I. INTRODUCTION

Among academia and industry, [1][2][3][4][5][6], the consensus is that Cache Coherence could extend its current predominance in future chip multiprocessors (CMP). In order to sustain future performance improvement, systems will need to increase the number of cores per CMP. To successfully do so, challenging problems such as the off-chip bandwidth wall will have to be faced. As is well known [7], providing a large on-chip cache is one of the most effective ways of mitigating this problem. Nevertheless, a hardware truism is that plain large capacities and fast access are not possible. To deal with these problems, the approach followed by current commercial systems is to use complex on-chip cache hierarchies, usually with three levels of cache, the two closest to the processor containing private copies and the last level cache (LLC) somehow shared among all the cores in the CMP. Complex cache hierarchies are intended to provide short-steps in a "stairway" to main memory that is moved further away from the core. The most common number of levels used nowadays is three [4][5][2], but this is subject to change in future systems, some commercial systems already using four levels [8].

The responsibility of the coherence protocol is to ensure that all the potential copies of a memory block scattered over different caches are coherent. In other words, any processor should see the same content of all the memory locations under any circumstance. A large number of cores and complex cache hierarchies might increase coherence protocol responsiveness. On the one hand, having a large number of cores in the chip makes it unfeasible to rely on broadcast-based coherence protocols. Although in current commercial CMPs this is the predominantly used approach [2][4][5][9] it has foreseeable difficulties to achieve success with larger numbers of cores in the system, due to their higher energy requirements [10]. On the other hand, complex cache hierarchies increase the likelihood of having multiple copies of shared blocks scattered over private levels, which, as we will show later, is challenging for pure directory-based coherence protocols. The private section of the cache hierarchy in current systems is quite large, in order to achieve progressive hit-times [11] throughout the different levels of the memory hierarchy. As the memory wall effects become more relevant, more on-chip cache capacity will be required and therefore large private caches will be needed in the medium term. In the long term, with the advent of 3D stacking or beyond-CMOS technologies, this tendency will be accentuated. Under these conditions, the amount of precise sharing information required by directory protocols in shared level caches or directories will be increased. Additionally, when the aggregate private cache available to each processor becomes larger, the access pattern perceived at the structures where the sharing information is kept will diverge faster than the one actually executed by the cores [12].

Under the previously depicted context, we have developed a coherence protocol suitable to confront the problem comprehensively. MOSAIC[1] is constructed on top of a conventional directory protocol [13], but instead of using inclusiveness to guarantee system correctness, MOSAIC will use a token coherence correctness substrate [14]. The proposal inherits the Token Coherence protocols' simplicity, their lack of precise sharing knowledge and the power efficiency of conventional Directory protocols. Additionally, MOSAIC circumvents not only most of the multicast traffic of Token Coherence, but also the inelegant starvation avoidance mechanisms needed due to the lack of serialization points. Although from a performance and a cost point of view non-inclusiveness is desired, the common assumption is that inclusiveness is inescapable to keep coherence protocol complexity attainable [15][12]. As a matter of fact, MOSAIC is

This work has been supported by the Spanish Ministry of Science and Innovation, under contract TIN2010-18159, the Spanish Ministry of Education, under grant PRX12/00006, and by the HiPEAC European Network of Excellence.

[1] Due to the 6 states used in it: M (modified) O (owned) S (shared) A (allocated block) I (invalid) C (constructing)

simpler than a plain directory coherence protocol and any block stored in private caches may not be tracked (i.e. no entry will necessarily be allocated in the directory). The protocol is engineered to reconstruct the entry under demand (i.e. if a core misses at its private cache levels for an untracked cache block that is stored in other cores' private cache). We will use token counting to guarantee coherence correctness (i.e. single writer, multiple readers) and to filter unnecessary traffic. Although our proposal is utilizable using in-cache or sparse directories, we will focus our attention on sparse directories [13]. We will show that even with extremely small directories and/or associativity, it is possible to sustain the performance and energy consumption of the system. The key aspect of this remarkable achievement is that token counting allows the data stored in LLC to perform directory entry reconstructions without any extra traffic. As the reader may remember, token coherence [14] is based on assigning a fixed number of tokens per cache block and requires at least *one* token to read and *all* of them to write. The most common case is that the data accessed will be private so LLC will have data with all the tokens. Taking this into account, LLC will, in most cases, have all the tokens, making it unnecessary to broadcast a message to reconstruct the block. In this way, LLC data will serve indirectly as the most effective filter to determine whether a data block is shared or not. To our knowledge, this is the first proposal that uses LLC data cache contents to effortlessly filter most of the on-chip traffic regardless of the complexity of the directory.

II. MOTIVATION

A. Directory Schemas

Traditional directory-based coherence protocols need a structure where sharing information, i.e. sharer vector and block state, is kept all the time. There are two main approaches to keep this information: in-cache and sparse directories [13].

For in-cache directories, each block stored in LLC has the tag and data attached to the block state and the sharers information (sharers bit vector, pointers, etc). The coherence controller uses this information to deal with incoming requests and having a precise knowledge of the block's sharing status is necessary to guarantee correctness. Therefore, in this case LLC inclusiveness with the previous level caches is necessary because it is the only way to have knowledge of private level contents. For small private levels, this approach has a substantial overhead because, in order to keep track of the sharing status of a handful of data blocks, any LLC block has to have a substantial storage space reserved per block (at least $\log_2 P$ bits for P processors).

If we take this into account, sparse directories seem a better approach for CMPs, because directory entries are allocated under demand and therefore the overhead is proportional to the aggregate private cache levels size (and not to the LLC size). When a block arrives at the chip in response to a request, a new directory entry is allocated. It will have to include at least, the block tag, the block state and the sharer vector. This entry is allocated in a separate structure from the data. In current NUCA caches, to guarantee scalability, the most extended strategy is to bank the directory over the chip, keeping the data and directory slices connected to the same router [16]. In most cases, the address-to-slice mapping used is statically determined by the lowest bits (closest to the byte offset) of the address. The capacity and associativity of the directory has to be sufficient to keep private-level cache tags. In small systems [17] with small private caches and low associativity, the coverage can be full, commonly denoted Duplicate Tag Directory. Nevertheless, for medium-to-large numbers of cores, there is no feasible way to use such large-scale associativity, it only being possible to keep a subset of L1 tags and enable conflict misses in it through an associativity reduction [18].

B. Inclusiveness implications with large private caches

The previous discussion assumes an inclusive directory, i.e. any block stored at any private cache level should have an entry allocated in the directory structure. This significantly simplifies coherence protocol implementation and verification effort [15]. Unfortunately, in a many-core CMP, the aggregate private capacity could be substantial. While L1 has to be small enough to attain core clock cycle, L2 should attenuate the latency of the presumably large L3 cache with a suitable hit-rate. As for commercial systems, the average private capacity of L2 is between 1/8 and 1/4 of L3 size. Similar optimal capacity ratios have been indicated by academic works such as [12]. The relevance of the hit-rate in private L2 cache makes it necessary to use highly associative caches, with at least 8 ways. Combining these three factors (large number of cores, large private caches and large cache associativity), the complexity of a Duplicate Tag Directory is unsustainable [18]. If either of these values is not sufficient, private block evictions will occur because of the lack of space available in the directory and so the system performance may be affected [19].

C. Previous approaches to the problem

Since directory-based coherence protocols are the most suitable choice for large-scale CMPs in energy terms, significant previous work was done to tackle the aforementioned problems. For sparse directories, works such as [19] only allocate directory entries for shared blocks, minimizing the pressure on the storage structure. Orthogonally, works such as [20][21] proposed skewed replacement algorithms to minimize directory conflicts. Other approaches propose minimizing the sharing information per directory entry in order to minimize the evictions [16][22]. Finally, solutions such as [23] opt for directory information reconstruction based on hardware probing. A comprehensive review of previous strategies can be read in [18]. All of these methods assume a conventional directory-based protocol and they do not alter the basic operation of the coherence protocol. For this reason, our approach could be combined with any of them in order to further improve the coherence protocol scalability.

III. MOSAIC

In contrast to how the aforementioned proposals confront the problems derived from inclusiveness, we choose a different approach: a new coherence protocol that does not require inclusiveness to guarantee correctness and which is still considerably simpler than a traditional directory protocol. Next, we will detail how the approach works.

A. Conceptual Approach

The proposal is focused on reducing one of the main problems that the directory approach has: the space needed to hold the coherence information for all the cache blocks stored in the private levels. The cost of the directory is determined by the size and plurality of the private levels. With a Duplicate Tag Directory, such as the one used by [6], it is necessary to have in the directory as many sets as there are in the private levels and the associativity for each of them has to be at least the product of the private levels' associativity by the number of processors. This would make the required associativity unfeasible, due to the large values needed if we use a large number of processors and a large private cache capacity, so this solution will not scale beyond a modest number of processors. To provide more reasonable associativity, yet keep the coherence information needed in most cases, it is necessary to increase the number of entries in the directory. This solution implies that new conflicts will appear because of coincident requests from different addresses requiring the same entry. These conflicts may induce the eviction of actively used blocks from the private cache levels. In contrast, under these situations, MOSAIC will not evict a block from the private level just because there is not enough space in the directory to hold its coherence information. In other words, directory evictions will be silent. Therefore, when an entry is being evicted from the directory, private copies are not invalidated. Consequently, after a miss in the directory, valid copies of the block might be in any of the private levels, in the LLC data slices or in off-chip memory.

After any subsequent miss in the directory, in order to guarantee coherence correctness, an on-chip reconstruction of the directory entry is initiated. This process will end when all the coherence information associated with that block (i.e. the sharers of the block and their state) has been collected. Token counting [24] is used to perform this process. This methodology states that all the block tokens (which are at least equal to the number of processors in the system) are required for a write operation and at least one token is required for a read operation. This approach makes the process simple and avoids negative acknowledgements [9]. To explain the process, Fig. 1 presents a simple example started when the processor P_0 sends a *read* request. After missing in its private cache (which might be composed of multiple levels) the request is forwarded to the directory slice corresponding to that address (step 1). If there is no valid entry for this address, the request is forwarded to the last-level cache associated with the address (step 2). In a correctly designed system, directory and LLC slices for the same address will be side by side [13]. If all the tokens of the cache block are stored in the LLC, the sharing information is directly reconstructed, since that means there is no other sharer in the system for this address. If not, a multicast is sent to all private caches and memory controllers (step 3). In this particular case, since the CMP has three processors, the tokens per block will be fixed to three, two being stored in P_1 private cache and one in P_2 private cache. To implement the owner state, Token Coherence uses a particular token denoted *Owner Token* to identify the forwarder of the data in a read operation. In this case, we assume that P_1 has the owner token (and consequently the block is in owner state (O)). If the owner token was located in LLC, data would be forwarded in parallel with the multicast request to the other private L1 caches. In this particular case P_1 will forward a token and a copy of the data to P_0 (step 4). Simultaneously the cores with tokens will notify the directory about the number of tokens they have for that block, if they have any, including P_0 with its new token (not shown in the figure). This information will be stored in the corresponding entry, silently evicting the previous one. For a *write operation*, the process will be similar with the difference that all of the sharers will forward their tokens to the requestor (rendering their copies invalid). After collecting all the tokens, the requesting processor will notify the directory. If P_1 had had the block in a modified state, a copy of the data and the required tokens would have been forwarded to P_0.

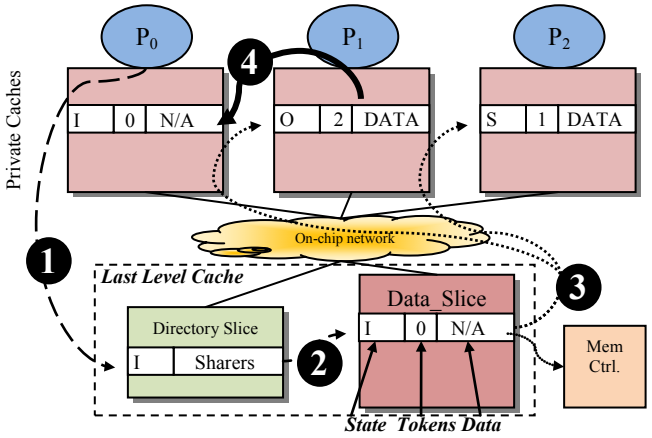

Fig. 1. Sketch of an example of a directory entry reconstruction.

Token counting is a key component in our proposal because it simplifies all the handshaking protocol used to reconstruct directory entries. Since the directory behaves like a serialization point, concurrent operations initiated by different processors in the same cache block will never end up suffering starvation. In this way, the directory coherence controller will avoid these problems without requiring persistent request [14] or added token tracking facilities [25][10]. Above all, token counting is useful to filter most of the multicast messages and speed up the directory entry reconstruction. As LLC capacity will be substantially higher than the directory's, this will be the most habitual scenario for actively used private data blocks, which is the common case. Therefore, in most situations the data and all the tokens will be allocated there. If all tokens are in LLC, it is known that no other copy of the block is located in any of the private caches and the directory entry reconstruction will proceed without broadcast. Additionally, it should be noted that actively shared data (such as those associated with frequent state changes, i.e. producer-consumer scenarios) will require frequent accesses to the directory. A plain LRU replacement algorithm in that structure, even with a low associativity, will evict entries of private data sooner.

As the reader might appreciate, the protocol is conceived to trade directory overheads for bandwidth. Although there are other similar solutions following this overhead-bandwidth tradeoff (such as [13][16][23][22]), in contrast with them, MOSAIC habitually uses a regular directory and requires extra bandwidth consumption in fringe situations. The key point is to use the large LLC capacity to indirectly filter unnecessary snoops without adding complexity in the coherence protocol or using any auxiliary hardware structure.

The benefits provided by MOSAIC will decrease the number of misses in private cache, which will reduce not only average access time, but also bandwidth consumption. In the previous example P_1 and P_2 will not be invalidated when directory entry is replaced. Therefore, the extra bandwidth required by MOSAIC can be compensated in conventional sparse directories. Although the case of in-cache directory is not studied in this paper, the effect will be more relevant since an early directory eviction will imply a subsequent memory access for later processor requests [12].

B. Protocol Details and Complexity Discussion

The states of each data block that have been considered are those corresponding to the token-based protocol [14]: Invalid (I), Modified (M), Owner (O), and Shared (S). There are two new states of the sparse directory, Constructing (C) and Allocated (A), which provide the key implementation details of the MOSAIC protocol.

Each of the entries in the directory will hold the coherence information about the address it makes reference to. Besides the necessary state of the block, this information is summarized in the sharers of that block, the core holding the owner token (as it will be in charge of forwarding data if necessary) and a token-count field of that block (we will see next why this is necessary). We assume the sharers are kept as a full bit vector, but we could use any other existing method [18][20] to maintain this sharers information.

In Table 1 a simplified table-based state-transition table [26] of a sparse directory controller is shown. With this table, states, events and main actions are represented. For each row, the first field corresponds to a possible state of an entry in the directory. There are three main states (I, C, A) and their corresponding transient states for performance improvement. A brief description of each one is shown in Table 2.

Each of the columns in Table 1 represents the event triggered by the controller when receiving a specific message. Each table cell shows the main actions taken by the controller and, in the bottom right corner, the state the block changes to in that transition (if not defined, the entry stays in the same state).

TABLE 2. DESCRIPTION OF MOSAIC'S MAIN COHERENCE PROTOCOL STATES

States	*Description*
I	Invalid. Block is not present in the sparse directory.
C_S	Constructing the block after receiving a read request (GETS) from a core.
C_X	Constructing the block after receiving a write request (GETX) from a core.
A	Allocated. Block is fully constructed with all the coherence information about that block.
A_S	Allocated and a read request (GETS) has been received from a core. Waiting for an unblock message.
A_X	Allocated and a write request (GETX) has been received from a core. Waiting for an unblock message.
A_I	Invalidating a block.

When receiving a request (GETS or GETX), if the block is not present in the directory (state I), a reconstruction process like the one explained in the previous section is initiated. In order to avoid putting the reconstruction process in the critical path when dealing with a request, MOSAIC piggybacks the original requestor and the type of request to the reconstruction message. This way, the reconstruction and resolution of the request are done in parallel. This is the reason why two construction states (C_S and C_X) are needed. During the reconstruction, when the controller receives information about some tokens' location, it adds that sharer to the sharers bit vector and updates the number of known located tokens. When the request triggering the reconstruction is a read request (state C_S), the cache with the owner token of the block will send a copy of the data with one of the tokens to the requestor and will inform the directory about how many tokens it has left.

TABLE 1. MOSAIC DIRECTORY CONTROLLER TRANSITIONS TABLE

Events / *States*	GETS	GETX	Token Info	Last Token Info	Unblock	PUT Data	PUT Tokens	Silent Replace	Replace with tokens	Ack From LLC
I	• initiate reconstruction → C_S	• initiate reconstruction → C_X				• write data in LLC	• write tokens in LLC			
C_S	wait	wait	• add sharer • update num tokens known	• add last sharer • wait for Unblock	• add last sharer → A	• bounce data to requestor	• update tokens	wait	wait	
C_X	wait	wait			• add exclusive sharer → A	• bounce data to requestor	• bounce tokens to requestor	wait	wait	
A	• forward request to Owner → A_S	• multicast request to all sharers → A_X				• update tokens	• update tokens	• invalidate block → I	• invalidate block • write Tokens in LLC → A_I	
A_S	wait	wait			• add new sharer → A	• update tokens	• update tokens	wait	wait	
A_X	wait	wait			• remove old sharers • add exclusive sharer → A	• bounce data to requestor	• bounce tokens to requestor	wait		
A_I	wait	wait				• write data in LLC	• write tokens in LLC			• remove block → I

When the requestor finishes its request, it sends an unblock message. The directory will always wait for the requestor's unblock message to finish the reconstruction. This will guarantee that no other request is dealt with until the entry is fully constructed and the request is completely resolved. If the request is a write request, all the caches with a copy of the requesting block will have to forward their tokens to the requestor, which will send the unblock message when it has collected all of them and so its request is finished. In this case, the directory controller will add the requestor as the exclusive sharer of the data (state C_X, event Unblock).

If the coherence information needed is in the directory (state A), all the data locations are known so the directory only has to forward the request to the appropriate sharer. If it is a read request (GETS), it sends it to the cache holding the owner token; if it is a write request (GETX), it sends it to all the sharers of the block.

The directory needs to be informed about all the replacements occurring in the private levels in order to have updated information about the sharers if the block is constructed. Any private L2 cache replacing a block sends a request with the tokens (PUT Tokens) or with the data if it has the owner token (PUT Data). If the entry is not constructed (state I), data and tokens are written back to LLC. If the block is being constructed (C_S or C_X) or the directory is dealing with a request (A_S or A_X) then the directory might be the one in charge of resolving the pending request with the replaced data so it bounces it to the requestor. If the block is constructed and there is no pending request (state A) then the directory updates the number of tokens at the entry (this is why it has to hold a token count field) or writes back data in LLC.

The rest of the events, state transitions, and actions are self-explanatory. However, the interested reader can access the full protocol specification (and the conventional directory) at [27].

IV. Evaluation Methodology

A. System Configuration

To analyze MOSAIC, we will use aggressive out-of-order cores, similar to those used by commercial systems [2][5][4]. The rationale behind this decision is that instruction-level parallelism (ILP) performance should not be underestimated [6]. Out-of-order processors will exert a high pressure on the coherence fabric. Since the number of pending instructions per core could be large, the concurrent coherence operations could be orders of magnitude bigger than those observed with a large count of simple in-order cores.

In our particular case, we will use 4-wide issue cores with 128 in-flight instructions and up to 16 pending memory operations. The number of cores chosen in our evaluation is 8 and 16 cores per CMP. The on-chip hierarchy configuration, like in [2][5][4], is composed of three levels. The first two are private, strictly non-inclusive layers. The third level is similar to the one proposed in [5], shared following a static NUCA [28] approach. In contrast with this system, instead of an ultra-wide ring network (which is imposed by the coherence protocol used) we will use a mesh network, which is characterized by better on-chip bandwidth scalability and better performance/cost ratio. We will assume that the routers in the network can handle multicast traffic natively [29]. Although for this size of system a broadcast protocol might perform better [25][10], our objective is to prove that MOSAIC is capable of overcoming classic directory limitations, which will be necessary with a much higher number of cores in the CMP. Nevertheless, to evaluate systems with tens or hundreds of cores is unfeasible with current evaluation tools because of the computational effort of such a task and the limited availability of scalable workloads. Comparing MOSAIC with a conventional protocol, varying the directory properties (i.e. associativity and capacity) might be enough to demonstrate the advantages of the proposal. Similarly, studying the evolution of the benefit and drawbacks of 8-core CMP compared to 16-core CMP will allow us to glimpse the scalability of the idea with a higher number of cores.

Table 3. Summary of 8-core CMP System Configuration (or 16-core CMP)

Core Arch.	Functional Units	4×I-ALU/4×FP-ALU/ 4×D-MEM
	Instruction Window size / Issue Width	128, 4-way
	Frequency / Processor Count	3Ghz, 8 (or 16)
Private Caches	(L1)Size/Associativity / Block Size / Access Time	32KB I/D, 2-way, 64B, 1 cycles
	(L2)Size / Associativity/ Block Size / Access Time	128KB Unified, 4-way, 64B, 2 cycles, Exclusive with L1
Shared L3	Size / Associativity / Block Size	16MB (or 32MB), 16 (or 32)×1MB, 16-way, 64B
	NUCA Mapping	Static, interleaved by LSB
	Data Slice Size / Access Time	1MB / 6 cycles
Mem.	Capacity / Access Time / Memory Controllers / BW	4GB / 240 cycles / 2 / 32GBs (or 4 / 64GBs)
Network	Topology / Link Latency / Link Width	4×4 (or 6×6) Mesh / 1 cycle / 16B
	Router Latency / Flow Control / Routing	1 cycle / Wormhole / DOR

Although MOSAIC can work in both sparse and in-cache directories, we will focus our analysis on the former, since they have less storage overhead [13][18] and the cache evictions due to directory conflicts have lower impact on performance. Although not shown in the paper, we have carried out a similar performance analysis with in-cache directory designs and MOSAIC provides higher benefits than it does in sparse directories. A summary of the main system parameters used in our analysis is shown in Table 3.

B. Workloads & Simulation Stack

We will use GEMS [33] as the main tool for our evaluation. With GEMS, it is possible to perform full-system simulations. Coherence protocols have been implemented using the SLICC language (Specification Language for Implementing Cache Coherence). For the power modeling we use CACTI 6.5[34] for modeling the cache and DSENT [35] for the network.

Ten workloads, shown in Table 4, are considered in this

Table 4. Multithreaded Workloads (8P and 16P)

SERVER [30]	OLTP	*IBM DB2 DBMS, TPC-C like 10000 Transactions*
	Apache	*Apache web server, SpecWeb like, 25000 Transactions*
	JBB	*SpecJBB, 70000 Transactions*
	Zeus	*Zeus, SpecWeb like, 25000 Transactions*
NPB[31]	Integer Sort (CG)	*CLASS A*
	Fast Fourier Transform (FT)	*CLASS W*
	LU Diagonalization (LU)	*CLASS A*
SPEC [32]	Astar	*Native, 7 thr.(8P), 15 thr. (16P)*
	Hmmer	*Native, 7 thr.(8P), 15 thr. (16P)*
	Omnetpp	*Native, 7 thr.(8P), 15 thr. (16P)*

study, including both multi-programmed and multi-threaded applications (scientific and server) running on top of the Solaris 10 OS. The numerical applications are three of the NAS Parallel Benchmarks suite (*OpenMP* implementation version 3.2 [31]). The server benchmarks correspond to the whole Wisconsin Commercial Workload suite [30]. The remaining class corresponds to multi-programmed workloads using part of the SPEC CPU2006 suite [32] running in rate mode (where one core is reserved to run OS services).

We model hardware-assisted TLB fill and register window exceptions for all target machines. Multiple runs are used to fulfill strict 95% confidence intervals (error bars are not visible in most cases). Benchmarks are fast-forwarded to the point of interest, during which page tables, TLBs, predictors, and caches are warmed up. In iteration-based applications, such as NPB, a warm checkpoint is taken in the middle of the execution and with a reduced number of iteration runs. Transactional workloads are warmed up by running hundreds of thousands of transactions. The chosen workloads have been selected trying to cover diverse use scenarios, varying the sharing degree (from none in SPEC applications to a large amount in Server Workloads) and sharing contention (from none in SPEC to a large amount in scientific applications). Among the NAS applications, we chose the 3 with the highest sharing contention. From the SPEC suite, we chose 3 applications with a variable range in working set size.

V. Impact of Directory Configuration on Performance

When the number of cores is large, conventional directory protocols have to face limitations in two main factors, capacity and associativity. Next we will analyze how sensitive MOSAIC is to both parameters and compare its results with those from a conventional sparse directory implementation. The reference point in this analysis will be a directory with duplicate tags. Since under this configuration there will not be private cache invalidations due to directory misses, there will be no performance differences between MOSAIC and conventional protocols. We will start with small private caches of a 2-way 32 KB L1 I/D and a unified 4-way victim L2 cache of 64KB. Assuming in both cases a block size of 64 bytes, for these cache sizes, the number of required entries in the directory to avoid capacity misses is *2048*#cores*. Until section VII, we will assume that the number of cores in the CMP is eight. Therefore, assuming 8 bytes per directory entry (enough to store tag and sharing information), the total directory size required to avoid capacity misses will be 128KB. The storage overhead will grow with the number of cores since the aggregate private cache will increase (the number of entries needed in the directory) and the sharing vector will be larger (the size of the entries in the directory). With the aim of minimizing the access time to data in data slices and avoiding bottlenecks in the accesses, we distribute the directory in 16 slices (as many slices as the LLC). The slice interleaving of data and directory entries over LLC uses the least significant bits of the address. For the same addresses, the directory slice and data slice are 1 cycle apart. To avoid all conflict misses in the directory, the required associativity will be 64. This large associativity is necessary because on each entry we need as many ways as the sum of both of the private levels' associativity times the number of cores (i.e. *(L1I associativity + L1D associativity + L2 associativity) * #cores*).

A. Sensitivity to Conflict Misses in the Directory.

Initially, we will determine the sensitivity of a conventional directory protocol and MOSAIC when the associativity is reduced, i.e. how the two protocols react when the number of conflict misses in the directory is increased. In order to perform this analysis, we keep the directory capacity fixed at 128KB and modify the associativity from 64-way to 1-way per set. As associativity goes down the number of conflicts grows, because

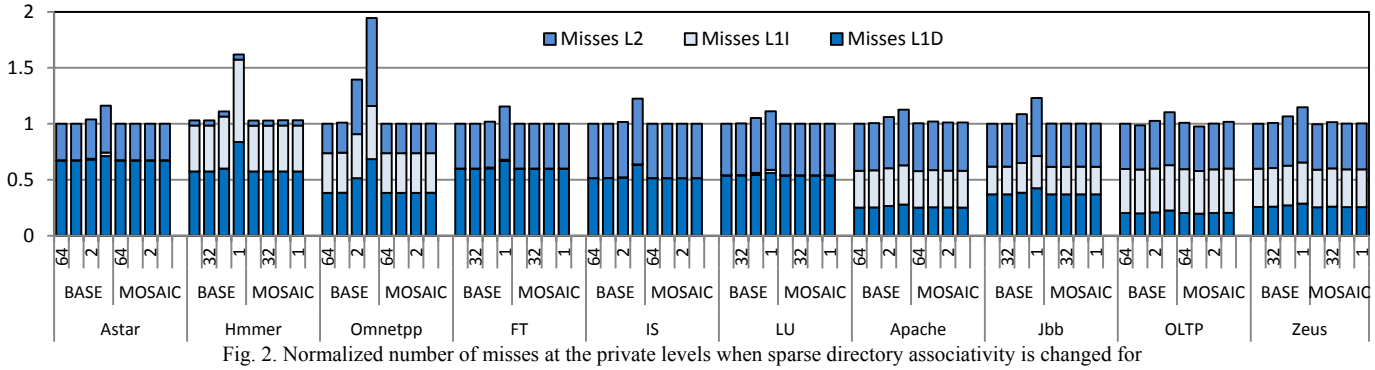

Fig. 2. Normalized number of misses at the private levels when sparse directory associativity is changed for a conventional coherence protocol (BASE) and MOSAIC.

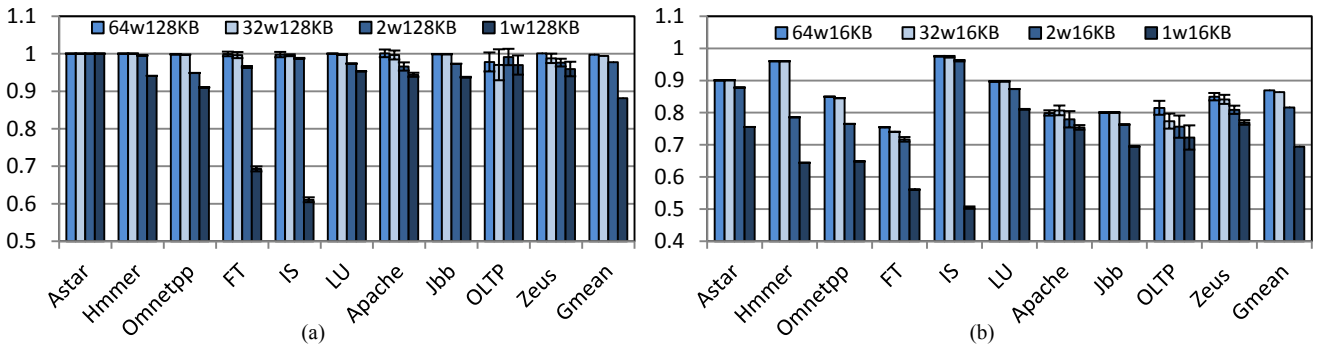

Fig. 3. (a) Mosaic execution time normalized to BASE, while varying the associativity of a fully sized sparse directory (i.e.16K entries). (b) Mosaic execution time normalized to BASE, while varying the associativity for a directory with one eighth of fully sized sparse directory (i.e., 2K entries).

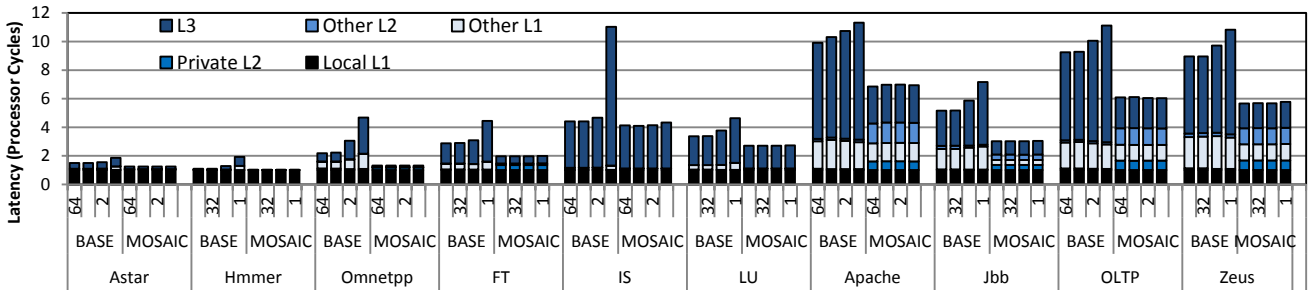

Fig. 4. Average on-chip latency for a 16KB (2K entry) sparse directory when varying its associativity.

even though there is space for all potential blocks stored in private caches, some of them may conflict in the directory.

Fig. 2 shows how the base Directory protocol (to avoid confusion from now on it will be denoted as BASE) and MOSAIC impact in cache level behavior when the number of directory conflicts is increased. Unsurprisingly, BASE directory has a bad reaction to that change in the associativity, obliging a large number of misses in private levels due to directory invalidations. In some applications, such as *Omnetpp* (where the cores are not sharing any data), the misses in those levels are multiplied by two. Nevertheless, and as expected, MOSAIC is completely insensitive to any associativity modifications. These results indicate that the implementation cost can be the same as the simple directly mapped configuration without any performance penalty.

The final performance differences depend on each type of application, i.e. its behavior in private caches using a duplicate tag directory. Fig. 3(a) shows these results, indicating that the MOSAIC protocol could be up to 40% faster than the BASE protocol. For the combination of system size and applications used, the most remarkable effects are found in extreme situations when even with capacity to track all private blocks, the performance will fall, on average, 12%. Previous works, such as [18], have identified limited associativity as a major issue in directory coherence protocols. MOSAIC overcomes this problem completely since a simple direct mapped directory is capable of maintaining the performance.

B. *Sensitivity to Capacity & Conflict Misses in the Directory*

The second effect that might influence performance is the capacity misses in the sparse directory. The combination of capacity misses induced by limited directory storage might increase conflict misses. To compare how both effects might impact on each protocol, we reproduce the previous analysis, but reducing the directory capability to track only an eighth of the private caches capacity, i.e. up to 2K blocks. Fig. 3(b) reproduces the results provided in Fig. 3(a) with the new directory capacity. In this new configuration, misses in private cache for BASE, although not shown, are substantially higher. After reducing the size of the directory, even with an associativity of 64, capacity conflicts in the directory have a relevant impact on performance, degrading it up to 20%. The capacity misses seem to be more relevant in applications with a higher sharing degree (i.e. commercial workloads [30]). Applications with a reduced working set (such as *hmmer*) are less sensitive to capacity misses in the directory. With this directory size, conflicts are more probable than in the fully sized directory and consequently associativity now has a greater influence on performance.

To understand how directory invalidations influence each protocol, we provide the average access time for on-chip hits in Fig. 4. Again, the dissimilar behavior of the two protocols is notable. On some applications, MOSAIC shows half of the on-chip latency of BASE due to the extra misses in private caches in the latter. Those requests are mostly resolved by LLC with extra added latency, which explains its growing contribution when the directory caused evictions in the private caches are more relevant. With MOSAIC, all the applications demonstrate a higher contribution of the private L2. Moreover, for applications with a high sharing degree, the broadcast reconstruction message favors the forwarding between caches as the *Other* L1 and *Other* L2 contributions show, and so avoids an access to L3 as the conventional directory does. The steady miss latency values obtained demonstrate MOSAIC's stability even in the most extreme configurations, a direct-mapped directory with capacity to track just an eighth of the private caches blocks.

C. *Sensitivity to Directory Size in a Realistic Private Cache configuration.*

Up to now, we have been using limited private cache capacity and associativity. If we consider the configuration of commercial systems [2][4][5], L2 caches have between 1/8 and 1/4 of L3 capacity and both L1 and L2 have a larger associativity. Therefore, we will next carry out a sensitivity analysis for the size of the directory with a realistic configuration for private caches. In this particular case, we try to mimic the L2 cache configuration in Intel's Nehalem (4-way 32 KB of L1s and 8-way 256 KB of L2). We will keep the

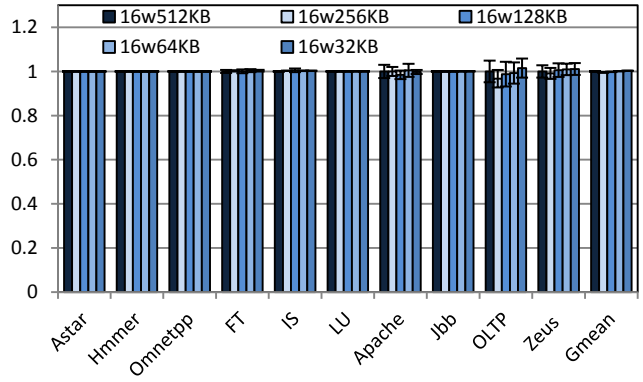

Fig. 5. MOSAIC execution time normalized to Duplicate Tag Directory, for a Nehalem-like private caches configuration varying directory capacity.

associativity fixed at 16-way (like in the data banks) and vary the capacity of the directory, from double [13] the full directory (i.e. 640KB) to a tenth of full directory (i.e. 32KB). Fig. 5 presents the average execution time for each application normalized to the double-sized directory where even with the smallest capacity, there is no performance impact. When reproducing the same experiment for the BASE protocol, the performance impact is greater than 20% in some cases.

VI. Cost Analysis: Bandwidth and Energy Overhead of MOSAIC

In light of the previous results, in contrast to a BASE protocol, MOSAIC's behavior is fairly independent of the directory configuration. Since the rationale of MOSAIC is to trade directory cost for on-chip bandwidth and additional snoops in private caches, we need to analyze the energy overheads. The first step in this analysis is to quantify how directory cost reduction influences on-chip bandwidth consumption. If the network is using routers with support for handling multicast traffic [29], the real measure of bandwidth and energy consumption for the interconnection network is given by the average link utilization and not the end-point traffic consumption [25]. Fig. 6(a) shows, for the initial configuration (i.e. exclusive 32KB L1 and 64KB L2), the average link utilization when the capacity of the directory or its associativity is reduced. The values are normalized for a duplicate tag directory, i.e. capacity for 16K entries (128KB) and 64-way associative. The results are outstanding, showing that on average and under the worst conditions (i.e. a 2-way associative directory, with an eighth of the capacity of the full directory) the traffic is just 5% higher than a duplicate tag directory.

Focusing on each application, when there is no sharing degree (such as the multi-programmed ones), applications are completely insensitive to directory configuration. Since there is no shared information, this is the expected behavior. More noteworthy is the behavior of scientific applications, where there is a substantial amount of shared and highly contended data. In such cases, the directory replacement algorithm prevents the eviction of actively shared data and entries of private blocks are more prone to be replaced. Consequently, traffic does not change. Server workloads seem to be the most sensitive, since in this case the amount of shared data is large, most of them being code. Therefore these blocks will be accessed in read-only mode and the directory will be less frequently accessed. As a consequence, the chances of evicting an actively shared entry are higher than in numerical applications and consequently so too are the chances of requiring a multicast to reconstruct these entries. Nevertheless, even in the most adverse (and unpractical) directory configurations, this increment is below 20%, which is substantially less than in broadcast coherence protocols [10][14][25].

The key point for this behavior is that multicast is only generated when, after a miss in the sparse directory, the data and tokens available in LLC are not enough to reconstruct the sharing information. If the block has all the tokens, we know that there are no copies in any private caches and consequently the multicast can be avoided. Since LLC can be very large, the most usual case will be this one and, therefore, multicast will be required only if the data is really shared. In contrast, if we compare the bandwidth consumption of MOSAIC and BASE protocols when the directory is simplified, the results are very different. As Fig. 6(b) indicates, the BASE protocol requires more on-chip bandwidth in most cases, especially when the directory is highly limited. In the most extreme case, i.e. a 16KB, 2-way associative directory, BASE requires up to 40% extra bandwidth consumption on average. The main reason for this is that MOSAIC has fewer misses in the private caches and directory evictions are silent. For instance, in SPEC applications all processors have independent executions so the conflicts that occur in the sparse directory with a conventional directory provoke a large number of invalidation messages to the private levels. These invalidation requests replace the data needed by the processors which may still be useful. Subsequent misses will require extra communication with the directory. In contrast, MOSAIC leaves these data in the private levels avoiding extra misses in the sparse directory and data travelling through the network. When the difference in the number of misses between the two protocols is small and applications have a high sharing degree, broadcast messages of the reconstruction requests are more noticeable. With highly contended shared data, such as in numerical applications, the replacement algorithm of the directory inhibits evictions of actively used data and therefore the external invalidations in caches with BASE are fewer (at least with directory configurations that are not highly constrained). Under this configuration MOSAIC memory misses might increase the traffic due to the multicast traffic required to deal with them. Although this multicast traffic might be avoided using simple

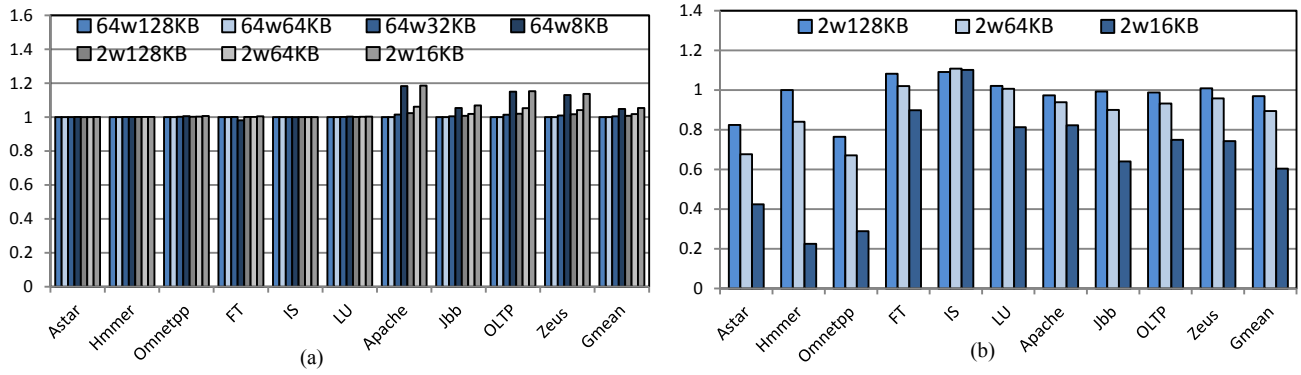

Fig. 6. (a) Average network link utilization of MOSAIC normalized to a duplicate tag directory, varying directory capacity and associativity. (b) Average network link utilization of MOSAIC normalized to BASE directory.

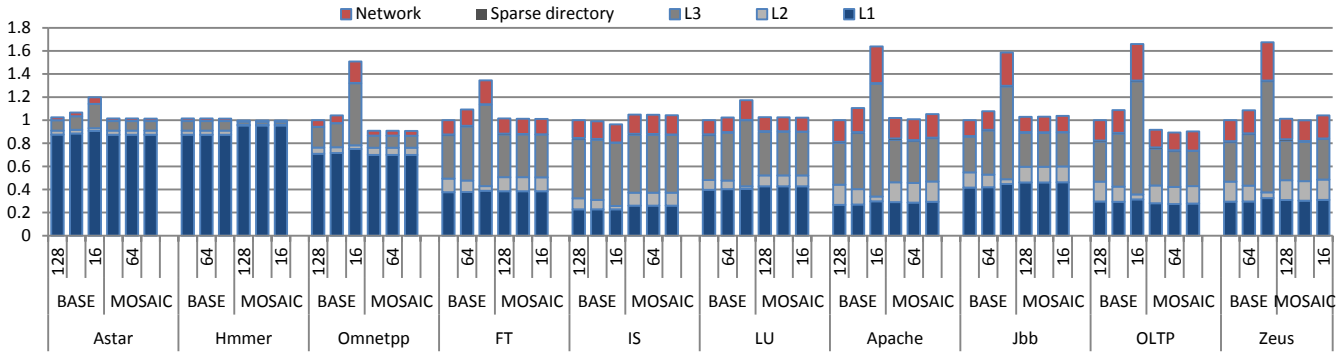

Fig. 7 Total dynamic energy used by caches and network normalized to the directory-based coherence protocol with an aggregate 128KB sparse directory. Different sizes: 128KB, 64KB and 16KB (8, 4 and 1 1KB per slice).

solutions such as [36], it seems irrelevant in most applications. The most relevant case is *IS*, which has a large MKPI. Even in these cases, the extra traffic is less than 10%. In server applications, shared blocks rarely change their state (from S) and they have the same probability to be evicted as private data blocks. Consequently, the number of invalidations of useful data in private caches is larger. The result is that the extra traffic required to deal with this situation is much greater than with MOSAIC.

The previous discussion partially addresses the potential added costs. To complete it, we need to look at the energy consumption, with emphasis on the cache hierarchy. Results of this analysis are shown for both protocols in Fig. 7 when using a 2-way associative sparse directory with three different sizes: 128KB, 64KB and 16KB. The results have been normalized to 128KB and a 2-way directory size of BASE protocol. The results are coherent with the traffic results: MOSAIC reacts in a more energy efficient way than the BASE protocol when the directory size is constrained. Therefore, we can conclude that the extra costs derived in the bandwidth-directory tradeoff overhead are favorable in our proposal.

VII. Scalability Analysis

To complete the cost analysis, we study how MOSAIC reacts in a CMP with 16 cores. In this system configuration we double the number of LLC banks and use a 6x6 mesh to connect them with private caches and four memory controllers. We maintain the remaining configuration parameters unchanged. To scale on-chip cache bandwidth, the number of banks and consequently the network has to be scaled up [28]. Comparing the results in Fig. 6(a) and Fig. 8, it can be seen that the differences are unnoticeable for most of the applications, even in extreme situations such as the one corresponding to a 2-way set associative directory with capacity to track an eighth of the private caches, which consumes only 7% more on average than a Duplicate Tag Directory. Like in the 8-core CMP, the server applications, due to their high sharing degree of read-only data, are the most sensitive to directory structure. Even in these cases, with a quarter of the directory capacity, the average extra traffic is below 10%. Although no performance or energy results comparison with a conventional protocol is provided for this configuration, it should be noted that these results are even better than the obtained in the 8-core CMP configuration. The rationale for this is that misses (due to directory invalidations) in private caches take longer to be resolved in LLC due to the larger size of the system.

Given the complexity of the evaluation environment and the architecture of the system evaluated, we cannot increase the number of cores simulated beyond this point. Nevertheless, comparing the evolution from 8 to 16-core CMP systems, we can deduce that the progression with larger number of cores might be similar. Since extra traffic will be proportional to the number of cores, the bandwidth overhead compared with an unfeasible Duplicate Tag Directory in bigger CMPs or with more realistic private cache hierarchies will be similar. Finally, note that to prevent on-chip and off-chip bandwidth impact on performance when increasing the number of cores in the chip, on-chip interconnection network bandwidth has to be extended [34]. In our particular case the bisection bandwidth has increased 50%, (from 4 to 6 bidirectional links), which is substantially larger than MOSAIC's traffic overhead in the most unfavorable directory configurations. Consequently, it seems reasonable to assume that MOSAIC will scale up for much larger systems.

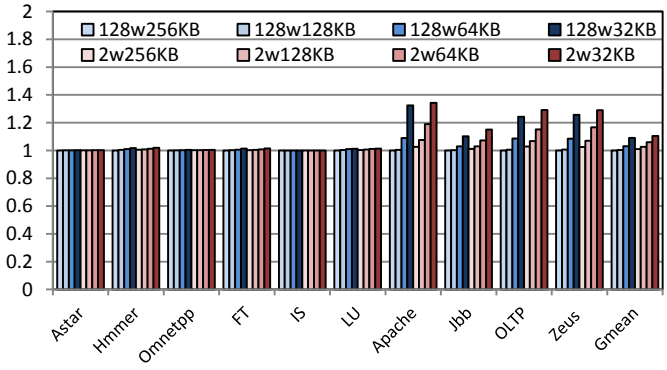

Fig. 8. Link utilization of MOSAIC normalized to a Duplicate Tag Directory (128-way associative, 256KB), varying directory capacity and associativity in a 16-core CMP.

VIII. Conclusions

The combination of complexity and scalability of our proposal suggests that it might be an interesting alternative for future many-core cache coherent CMPs. Since MOSAIC is quite insensitive to the directory configuration, the overhead of this structure in a large-scale system might be marginal. MOSAIC amalgamates the bandwidth scalability of a conventional directory with the elegancy of the Token Coherence correctness substrate, achieving a new approach capable of dealing with the problem of cache coherence in large-scale systems. All without incurring in noticeable

complexity or energy cost, which in our opinion, is a noteworthy finding.

ACKNOWLEDGMENTS

The authors would like to thank Jose-Angel Herrero for his valuable assistance with the computing environment CPD 3Mares, as well as Viji Srinivasan and the anonymous reviewers for all the useful comments and suggestions.

REFERENCES

[1] M. M. K. Martin, M. D. Hill, and D. J. Sorin, "Why on-chip cache coherence is here to stay," *Communications of the ACM*, vol. 55, no. 7, p. 78, Jul. 2012.

[2] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, no. 2, pp. 7–15, 2010.

[3] "Tilera. TILE-Gx 3000 Series Overview.," 2011.

[4] M. Butler, "AMD 'Bulldozer' Core - a new approach to multithreaded compute performance for maximum efficiency and throughput," in *IEEE HotChips Symposium on High-Performance Chips (HotChips 2010)*, 2010.

[5] N. Kurd, J. Douglas, P. Mosalikanti, and R. Kumar, "Next generation Intel® micro-architecture (Nehalem) clocking architecture," in *IEEE Symposium on VLSI Circuits*, 2008, pp. 62–63.

[6] J. L. Shin, H. Park, H. Li, A. Smith, Y. Choi, H. Sathianathan, S. Dash, S. Turullols, S. Kim, R. Masleid, G. Konstadinidis, R. Golla, M. J. Doherty, G. Grohoski, and C. McAllister, "The next-generation 64b SPARC core in a T4 SoC processor," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 82–90, Feb. 2013.

[7] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: challenges in and avenues for CMP scaling," in *36th International Symposium on Computer Architecture (ISCA'09)*, 2009, vol. 37, no. 3, pp. 371–382.

[8] F. Busaba, M. A. Blake, B. Curran, M. Fee, C. Jacobi, P.-K. Mak, B. R. Prasky, and C. R. Walters, "IBM zEnterprise 196 microprocessor and cache subsystem," *IBM Journal of Research and Development*, vol. 56, no. 1, pp. 1:1–1:12, Jan. 2012.

[9] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor," *IEEE Micro*, vol. 30, no. 2, pp. 16–29, 2010.

[10] A. Raghavan, C. Blundell, and M. M. K. Martin, "Token tenure: PATCHing token counting using directory-based cache coherence," in *41st IEEE/ACM International Symposium on Microarchitecture*, 2008, pp. 47–58.

[11] S. Przybylski, M. Horowitz, and J. Hennessy, "Characteristics Of Performance-Optimal Multi-level Cache Hierarchies," in *16th International Symposium on Computer Architecture (ISCA'89)*, 1989, pp. 114 – 121.

[12] A. Jaleel, E. Borch, M. Bhandaru, S. C. Steely Jr., and J. Emer, "Achieving Non-Inclusive Cache Performance with Inclusive Caches: Temporal Locality Aware (TLA) Cache Management Policies," in *43rd IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 151–162.

[13] A. Gupta, W. Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," *Springer US*, pp. 167–192, 1992.

[14] M. M. K. M. K. Martin, M. D. D. Hill, and D. a. A. Wood, "Token Coherence: Decoupling Performance and Correctness," in *30th International Symposium on Computer Architecture (ISCA'03)*, 2003, pp. 182–193.

[15] J.-L. Baer and W.-H. Wang, "On the inclusion properties for multi-level cache hierarchies," *ACM SIGARCH Computer Architecture News*, vol. 16, no. 2, pp. 73–80, May 1988.

[16] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *42nd IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 423–434.

[17] "OpenSPARC TM T2 system-on-chip (SoC) microarchitecture specification," 2008.

[18] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *18th IEEE International Symposium on High Performance Computer Architecture*, 2012, pp. 1–12.

[19] B. A. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. F. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *38th International Symposium on Computer Architecture (ISCA'11)*, 2011, pp. 93–104.

[20] D. Sanchez and C. Kozyrakis, "The ZCache: Decoupling Ways and Associativity," in *43rd IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 187–198.

[21] M. Ferdman, P. Lotfi-Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, 2011, pp. 169–180.

[22] H. Zhao, A. Shriraman, S. Dwarkadas, and V. Srinivasan, "SPATL: Honey, I Shrunk the Coherence Directory," in *20th International Conference on Parallel Architectures and Compilation Techniques (PACT'11)*, 2011, pp. 33–44.

[23] J. H. Kelm, M. R. Johnson, S. S. Lumetta, and S. J. Patel, "WayPoint : Scaling Coherence to 1000-core Architectures," in *19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10)*, 2010, pp. 99–110.

[24] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token Coherence: a new framework for shared-memory multiprocessors," *IEEE Micro*, vol. 23, no. 6, pp. 108–116, 2003.

[25] L. G. Menezo, V. Puente, P. Abad, and J. A. Gregorio, "Improving coherence protocol reactiveness by trading bandwidth for latency," in *9th ACM International Conference on Computing Frontiers (CF'12)*, 2012, pp. 143–152.

[26] D. J. Sorin, M. Plakal, A. E. Condon, M. D. Hill, M. M. K. Martin, and D. A. Wood, "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 6, pp. 556–578, Jun. 2002.

[27] "Mosaic Protocol Specification." [Online]. Available: http://www.atc.unican.es/galerna/mosaic.

[28] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA substrate for flexible CMP cache sharing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 1028–1040, 2007.

[29] N. E. Jerger, L. S. Peh, and M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *35th International Symposium on Computer Architecture (ISCA'08)*, 2008, pp. 229–240.

[30] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. D. Hill, D. A. Wood, and D. J. Sorin, "Simulating a $2M Commercial Server on a $2K PC," *Computer*, vol. 36, no. 2, pp. 50–57, Feb. 2003.

[31] H. Jin, M. Frumkin, and J. Yan, "The OpenMP Implementation of NAS Parallel Benchmarks and its Performance," *NAS Technical Report NAS-99-011, NASA Ames Research Center, Moffett Field, CA*, 1999.

[32] SPEC Standard Performance Evaluation Corporation, "SPEC 2006." [Online]. Available: http://www.spec.org.

[33] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset," *Computer Architecture News*, 2005.

[34] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *40th IEEE/ACM International Symposium on Microarchitecture*, 2007, pp. 3–14.

[35] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," *6th IEEE/ACM International Symposium on Networks-on-Chip*, pp. 201–210, 2012.

[36] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked DRAM caches," in *44th IEEE/ACM International Symposium on Microarchitecture*, 2011, pp. 454–464.