Improving Coherence Protocol Reactiveness by Trading Bandwidth for Latency

Lucía G. Menezo Valentin Puente Pablo Abad José Ángel Gregorio

University of Cantabria

Los Castros Ave. s/n 39005 Santander (Spain)

{gregoriol, vpuente, abadp, monaster}@unican.es

ABSTRACT

This paper describes how on-chip network particularities could be used to improve coherence protocol responsiveness. In order to achieve this, a new coherence protocol, named LOCKE, is proposed. LOCKE successfully exploits large on-chip bandwidth availability to improve cache-coherent chip multiprocessor performance and energy efficiency. Provided that the interconnection network is designed to support multicast traffic and the protocol maximizes the potential advantages that direct coherence brings, we demonstrate that a multicast-based coherence protocol could reduce energy requirements in the CMP memory hierarchy. The key idea presented is to establish a suitable level of on-chip network throughput to accelerate synchronization by two means: avoiding the protocol serialization, inherent to directory-based coherence protocol, and reducing average access time more than in other snoop-based coherence protocols, when shared data is truly contended. LOCKE is developed on top of a Token coherence performance substrate, with a new set of simple proactive policies that speeds up data synchronization and eliminates the passive token starvation avoidance mechanism. Using a full-system simulator that faithfully models on-chip interconnection, aggressive core architecture and precise memory hierarchy details, while running a broad spectrum of workloads, our proposal can improve both directory-based and token-based coherence protocols both in terms of energy and performance, at least in systems with up to 16 aggressive out-of-order processors in the chip.

Categories and Subject Descriptors

B.3.2 [**Memory structures**]: Design Styles – *cache memories*

Keywords CMP, coherence protocol, memory hierarchy.

1. INTRODUCTION

Chip multiprocessors (CMPs) represent a major milestone in computing system evolution. Adding more processors per chip seems to be the most reasonable approach to keep translating the continuous enhancement in technological integration into performance improvements. Given the challenge involved in parallel software development, the hardware has to assist the programmer's productivity [4] as much as possible. The consensus is that it is much easier to perform this task by providing all chip cores with a unified memory view. From the hardware point of view, in CMP systems one of the major challenges is the off-chip bandwidth wall. Among other solutions, it is essential to provide complex on-chip cache hierarchies to minimize off-chip interface pressure.

If we combine the above-mentioned facts, it seems that cache coherent CMP will become the dominant class of systems, at least in general purpose computing. Today, many commercial products that target this market implement this approach [28][18][7]. However, this statement does not negate the suitability of non-cache coherent CMPs, such as [13], in some specialized markets.

In a CMP system, the computing elements are so intricate that hardware-enforced cache coherence is the easiest way to support the shared memory model and so, the coherence protocol has a fundamental role to play. Many architectural solutions used in CMP systems are borrowed from the off-chip realm without substantial alterations. In particular, many of the cache coherence protocols used or proposed take advantage of premises from System-Multiprocessors. Some of them are very cautious about bandwidth utilization at the expense of increasing latency. In an off-chip interconnection network, bandwidth is scarce because of the discrete nature of the communication system elements. In contrast, in on-chip interconnection networks bandwidth availability is greater. In this type of systems, communication link width is much greater and the delay allows much faster data rates with lower energy cost. 3D stacked systems [37] and utilization of low-swing links [20] substantially increase the excess in bandwidth and reduce the energy cost of moving data.

The coherence protocol should, at all costs, use on-chip network bandwidth availability to avoid adding extra latency in the form of indirections. Currently there are a substantial number of CMP coherence protocol proposals that share our view [2][22][26][30]. Most of these ideas use broadcasting as the mechanism to overcome indirection at intermediate ordering points. Nevertheless, bandwidth demand is still a concern in most of these works and they allow some performance to be lost in exchange for saving bandwidth consumption.

With a suitable interconnection network design it could be possible to increase the whole system performance by improving the coherence protocol behavior. Following this premise, we introduce the LOCKE Coherence Protocol in this work. This protocol uses the token coherence framework [22] as its starting point, but enhancing responsiveness and stability in several ways. First, token coherence deals with concurrent requests coming from different processors to shared blocks using a passive approach called "Persistent Request". This mechanism uses a time-out-based triggering policy to address the aforementioned situation. Consequently, critical operations, such as contended synchronizations, could be artificially delayed. This negatively affects system performance. Second, the mechanism could overreact when the network is heavily loaded, potentially turning most of the processor memory accesses into persistent requests. Contention is hard to manage and adds unpredictable and non-depreciable delays in latency. On top of that, persistent request increases contention due to the extra traffic generated. In extreme situations, some applications could render the system useless due to the chain-reaction produced by persistent request explosion and contention. Neither static nor dynamic time-out estimation is sufficient to avoid such unstable behavior, because they cannot capture the diverse and complex situation that contention produces.

In order to identify where each token position is, LOCKE uses explicit acknowledgments for each token movement. Thus, each memory request will locate either tokens or pending acknowledgments. In this way, we can quickly forward requests to in-flight tokens' destinations, which would improve latency when accessing contended data. Applying a correct ordering between true racing requests, eventually any pending operation will locate the data and all the tokens needed to complete the transaction. No starvation avoidance mechanism, such as persistent request, is required. It might appear that acknowledgment traffic will increase bandwidth utilization and added contention could potentially increase network latency or network energy consumption, however, using state-of-the-art network design we will demonstrate that this is not the case. The effectiveness of the token location mechanism compensates for its extra bandwidth consumption, improving the energy-performance tradeoff of both token coherence and directory-based coherence protocols.

We have evaluated the effectiveness of the idea using a state-of-the-art full-system simulator which includes a very precise interconnection network simulator with a wide variety of workloads ranging from multithreaded server applications, through multithreaded numerical applications to multi-programmed workloads. LOCKE outperforms, on average, a conventional Directory and a Token Coherence protocol by 16% and 28% respectively for 16-core CMP with Nehalem-like cores. Additionally, LOCKE exhibits lower susceptibility to workload characteristics, having six times less performance variance than Token Coherence.

The rest of the paper is organized as follows: Section 2 explains the motivation for the introduction of LOCKE. Section 3 describes the proposal itself, explaining the foundations of the coherence protocol. Section 4 describes the experimental methodology employed. Section 5 presents performance results and provides insight into LOCKE responsiveness. Section 6 summarizes the related work and, finally, Section 7 states the main conclusions of the paper.

2. MOTIVATION

2.1 Trading Bandwidth for Latency

Bandwidth availability is profuse in CMP environments because of the utilization of scalable point-to-point interconnection networks, scalable cache hierarchies such as NUCA, and ultra-wide short links. In contrast, the portion of the chip reachable per clock cycle is shrinking as the technology advances. Under this scenario, it seems inadequate to maintain coherence in a CMP using protocols originally conceived for off-chip systems, such as directory-based ones [21], especially if their utilization increases data access latency due to the burden of multiple indirections across the chip. Therefore, taking advantage of bandwidth availability to avoid adding extra delay makes sense. As stated before, snoop-based protocols running on top of scalable interconnection networks provide the best design choice for CMP systems. Currently most commercial aggressive CMPs, such as [28][18], use this approach. Nevertheless, it is commonly accepted that those protocols are not free of shortcomings, namely: 1) The multicast traffic required for on-chip cache requests will increase power consumption; 2) An excessive network and cache bandwidth consumption could increase contention and increase on-chip latency, potentially ruining the rationale of snoop-based coherence protocols, and 3) The extra cache tag lookups produced in such protocols will increase cache energy consumption.

Although these considerations are pertinent, their impact can be much smaller than is commonly believed. First, power consumption is affected by this multicast traffic in a different way depending on the network characteristics. If the network has hardware support for multicast [10], its impact is highly reduced. In this case, each network resource is used at most once per request instead of many times as occurs when no support is provided. According to [1], using multicast support could save up to 70% in the Energy Delay Square Product (ED2P)[1] . Second, a correctly dimensioned design for the cache hierarchy capable of decoupling the number of cores and the on-chip cache bandwidth will oblige the use of NUCAs [14], as [28][18] are already doing. Under these circumstances, on-chip communication bandwidth will scale in proportion to core count. Third and finally, if we take into account the growing leakage in each technological advance [15], the area devoted to cache, and the substantial benefit in terms of performance obtained by snoop-based coherence, increased tag snoop energy is quickly amortized by the reduction in static energy.

2.2 Token Coherence Responsiveness

Conceptually the Token Coherence protocol deals with racing requests by counting tokens. This way, data races are avoided by forcing different ongoing memory operations to require an incompatible number of tokens in order to be performed. For example, performing a simultaneous read (GETS) and write (GETX) over the same cache block requires more than the maximum number of tokens available in the system. In starvation-prone circumstances, each contending processor eventually issues a persistent request, which will statically determine the winner and force the losers to return the tokens to the frontrunner processor. When this one finishes its operation, the next processor obtains the tokens required to perform its pending memory transaction. Assuming that under realistic working conditions racing requests are not frequent, this serialization will have a negligible impact on performance.

However, synchronization is a key operation in multithreaded workloads [35], which in most cases will involve racing requests. The passive approach used by token coherence to resolve that situation, which is bounded by the time established to issue the persistent request, could delay synchronization resolution unnecessarily. Additionally, persistent requests not only serialize potential data races, but also address temporary lack of knowledge about token location. This problem situation arises when some of the tokens required to perform a specific memory transaction are unavailable at the end point of all multicast messages issued by the request. For example, this happens when a block is evicted

[1] ED2P is the most suitable energy-performance tradeoff for high-performance systems [38], such as the scenario assumed in this paper.

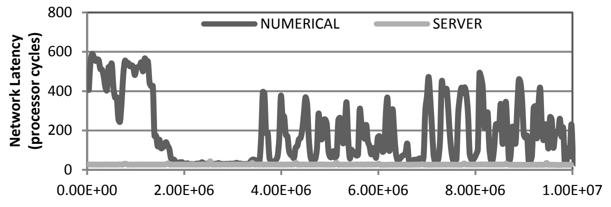
from the cache and the request overtakes the in-flight data block in the interconnection network. In these circumstances, the request will not be fulfilled because the destination of the tokens being evicted will never be located by the original request. The outcome of this situation is similar to a temporary racing request, denoted as "false racing request". By contraposition, we denote concurrent and simultaneously incompatible operations issued over the same block by different processors as "true racing requests". When interconnection network contention is considered, this situation might not be as negligible as it was with true racing requests, especially in highly contended situations.

2.3 Token Coherence Stability

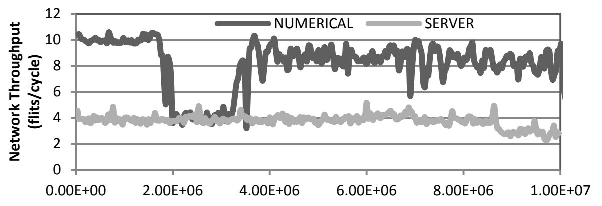
The persistent request method is a starvation avoidance mechanism that solves true or false request races by keeping track of the time involved in each pending memory request. If the time is greater than a fixed time threshold, a persistent request is sent. In order to maintain the scalability of the hardware, structures are required to perform persistent requests and to provide a distributed and fair arbitration scheme. Token coherence assumes that only one ongoing persistent request per core is supported. To minimize performance impact in processors with multiple outstanding memory operations, the original request is reissued one or more times before sending a persistent request.

The timeout chosen to trigger this process can be established statically, looking at the on-chip miss access latency, or dynamically, averaging the average latency of recent memory transactions. If the time of a particular ongoing memory transaction is above this limit, it seems reasonable to suppose that there might be another core accessing the same block. The request is reissued and if the timeout is once again exceeded then a persistent request is sent. Although this mechanism seems to be very simple, contention effects are ignored. For example, when the load applied over the network is significant, the communication latency of each individual message increases as a result of the unavailability of resources in use by other messages. At medium loads the total latency could increase by a few cycles, but when the load is higher this variation could be substantially larger and, worst of all, highly dependent on the applied load and on the interconnection network implementation.

In a low contention situation, network latency is closer to the base latency and persistent requests work as expected. Nevertheless, if a spike of traffic suddenly appears, contention increases and so does the latency of all pending memory transactions. If the effect of the contention is over the persistent request timeout, a chain reaction is triggered. The positive feedback between reissues and persistent request and network contention creates a storm of these types of requests in which almost any memory operation is reissued or even resolved by a persistent request. Under this unstable situation, the system performance drops dramatically. To illustrate this phenomenon, we will focus our interest on two particular applications (NUMERICAL and SERVER) running in 16 aggressive cores in the CMP, described in subsection 4.1. All of the parameters of the system, including the network, are correctly dimensioned, i.e. they are chosen in order to obtain an optimal cost/performance ratio over a large set of applications. The sharing degree of the two applications is quite different, in the SERVER it is high and in the NUMERICAL it is low. However, for an optimal time-out threshold and one reissue before sending a persistent request, the proportion of memory transactions resolved by persistent request is less than 0.1% for SERVER and more than 10% for NUMERICAL.



(a)



(b)

Figure 1. Network dynamic evolution with a 16-processor system (a) Average latency (includes injection queue delay), (b) Throughput.

This behavior, which is apparently contradictory according to the sharing degree of each application, is easily explained looking at Figure 1. It shows the network latency (a) and the applied load (b) during 10 million processor cycles for both applications. In contrast to the SERVER, the NUMERICAL application is very interconnection network demanding during short intervals due to the access to highly contended blocks. During these phases, the latency spikes due to on-network contention effects. These effects are exacerbated by the one-to-all traffic pattern of the application. During these spikes, reissue and persistent request frequency increases, not because of true racing requests but because packets are delayed within the network. This triggers more reissues and persistent requests, which further increases contention. Even using dynamically predicted thresholds, we are unable to predict any sudden variations in latency. In fact, dynamic estimations could accelerate system instabilities even preventing the complete execution of the workload. The described effect is not a rare anomaly and similar behavior could also be observed if off-chip bandwidth is saturated. All in all, without a solution for this problem, employing this protocol in a general purpose machine would be highly risky.

3. LOCKE COHERENCE PROTOCOL

LOCKE will use token counting to maintain coherence invariants, but it introduces a smart mechanism to actively resolve true racing requests, making a passive starvation avoidance mechanism unnecessary. In order to do this, LOCKE is based on precise knowledge of where any token is or will be located in the near future. If the protocol is able to track all the tokens, no false racing requests are possible. True racing requests are solved with a starvation-free self-inhibition mechanism that serializes data access of simultaneous incompatible memory transactions. Next, we will detail how false racing requests are avoided and true racing requests are dealt with. For readers interested in a more detailed specification of the protocol, a table-based state-transition table of cache controllers can be seen at [27].

3.1 False Racing Requests: Token location

In order to determine token location, any block movement is monitored at the originating location, keeping a label of the destination of the block. The label information is kept until a message reception acknowledgement is received from that destination. Thus, when a coherence controller generates a request, all the tokens needed or the flag of some pending acknowledgement will be found. Note that, in contrast to directory-based protocols, LOCKE's acknowledgement messaging is outside the critical data access path.

If the request corresponds to a write operation (Get Exclusive or GETX) every token will be forwarded to the requestor. On the contrary, if the request corresponds to a read operation (Get not exclusive or GETS) only the controller with the owner token will reply. If a request arrives when the tokens required are in-flight the requestor is notified with the final destination of the tokens. In this way, the requestor may reissue a unicast request to the one holding the necessary tokens. The intermediate node always notifies the requestor if the transaction is a GETX, but only notifies that the owner token is in-flight when the request is a GETS. Note that this is the situation depicted in the example in Figure 2, where simultaneously, processors P0 and P2 try to perform a GETS operation for the same block, and P1 holds only the owner token for that block. This situation in Token Coherence Protocol implies a false racing request. The side effect of this mechanism is the generation of extra unicast traffic for acknowledgement packets and reissuing the GETS. As we said before, in contrast to directory-based coherence protocols, acknowledgments operate outside the critical path of any memory transaction. In this example, the hit latency of processor P2 will not be increased because of the mechanism.

Unfortunately, the previous scheme is starvation prone. To exemplify this, Figure 3 shows the same initial situation as in Figure 2. This time, P0's request is delayed long enough so that it arrives at P1 when the acknowledgement message from P2 has already been received. In this situation, P1 does not notify P0 that P2 has the block and the owner token. Moreover, P2 is unaware of P0 being interested in that block because P0's request arrived at P2 before this processor issued the GETS. If both of these things happen, P0's transaction starves.

In order to prevent this anomalous situation, we need an approach to order both requests on the interconnection network. The most scalable way to perform such ordering is to use the same multicast routing tree for each set of addresses. If we force all the requests to a specific address to follow that routing tree, then no request or acknowledgment race is possible because the messages involved cannot be overtaken. To balance network resource utilization we could define different multicast trees per address. Routers should include the mechanism to use the right routing tree according to the address accessed. Using the least significant bits in the address we could select which one to follow. Figure 4 shows a possible distribution in an 8-processor CMP with non-uniform cache architecture using a 4×4 mesh interconnection network and four multicast trees. We will denote the multicast trees as I-trees. To minimize base latency effects, each I-tree trunk can pass through the last level (LLC) slice where the address could be located. Note that one of the destinations for the request multicast will be an L2 slice. For example, addresses mapped in slice 0, 4, 8 and 12 will use the I-tree for addresses %XXX00.

[2] The shadowing in the lines reflects the block state: Inv (Invalid state), O (Owner State), OI (Owner to Invalid), IS (Invalid to Shared). The line in the arrows reflects the nature of the message: dashed lines correspond to multicast, solid lines correspond to unicast.

Any multicast-capable network requires a multicast routing tree [11]. For example, in Figure 4, if core 0 requests data that is located in core 1 L1 cache, it will take only one hop in the network to reach it. In the worst case, if data is located in core 4 L1 cache using the I-tree in the figure it will take 7 network hops to reach it when in an optimal multicast tree it will take 3 hops. Although the average impact on on-chip latency overhead will depend on data distribution and network contention, the average distance increment for multicast messages is less than 10%. Moreover, the rest of the traffic (no requests or acks) always follows minimal paths.

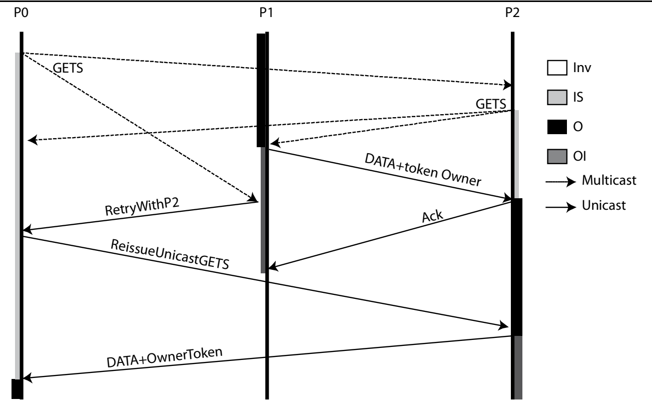


Figure 2. Token location with explicit acknowledgement: P0 issues a GETS operation transitioning the block to IS[2], P2 issues another GETS operation for the same block. The request from P2 arrives first at P1, which has only the owner token. P1 sends the data with the owner token to P2, transitioning its own block to OI. This state will be maintained until the explicit reception acknowledgement from P2 arrives at P1. When the block is received at P2, the block goes to the stable state O and the acknowledgement message is sent. In the meantime, the request from P0 arrives at P1 which informs it that P2 has the owner token. P0 reissues a unicast to P2 demanding a copy of the data.

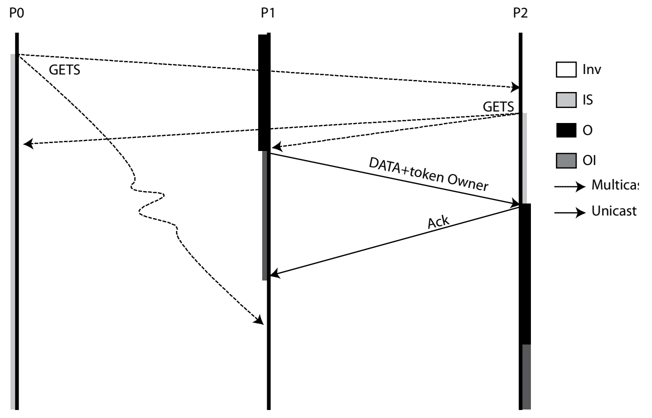


Figure 3. Starvation with request overtaking: With the same initial state depicted previously, the P0 multicast request message arrives at P2 before it issues its own GETS and arrives at P1 after the acknowledgement reception from P2. Both processors P1 and P2 ignore P0's request.

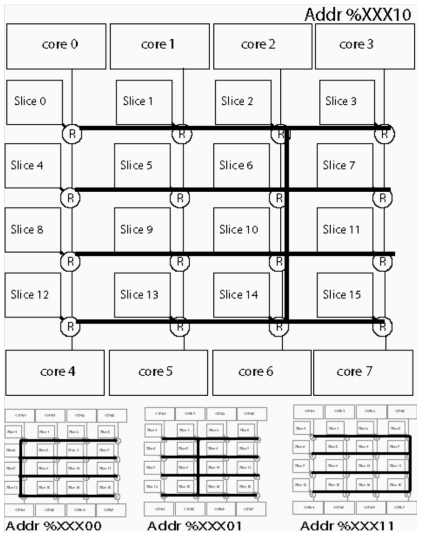


Figure 4. Ordering I-tree in an S-NUCA architecture.

3.2 True Racing Requests: Arbitration

3.2.1 *Self-inhibition*

If the location of all tokens needed to complete a transaction is known then only true racing requests have to be dealt with. When two or more processors are trying to perform simultaneous but incompatible operations, LOCKE solves the situation using scalable self-regulated arbitration. The solution adopted is to assign a priority order to each processor and operation and to allow the resolution of the race without breaking the coherency invariants. The different coherency controllers apply this policy in a fully distributed way, so guaranteeing system scalability.

Two or more simultaneous operations over the same block are incompatible if the total number of required tokens is greater than the number of processors P. If one coherence controller detects the possibility of such a situation arising, it must choose whether to keep going with the operation or to give up. For example if it wants to perform a write operation in a cache block and sees an incoming write request from another processor trying to write in the same cache block, it has to check each request priority. Initially and for the sake of simplicity we will assume that the priority is determined by the processor index. If the current controller has an index smaller than the incoming request, the controller goes ahead with its operation or, if not, it self-freezes the operation.

If the controller decides to temporarily inhibit the outgoing transaction, due to its inferior priority with respect to the remote incoming request, it changes the block state to "frozen" and annotates the winner controller for that block. When a block is frozen, any incoming token will be forwarded to the annotated winner controller. The block will remain in a frozen state until the winner notifies the completion of the operation, via a complete multicast message. If this happens, the inhibited operation is reissued from the beginning. Figure 5 presents an example of this situation.

[3] The shadowing in the lines reflects block state: Inv (Invalid state), M (Modified State), MI (Modified to Invalid), IM (Invalid to Modified), Frozen (frozen memory operation).

When a block is frozen, any other write request from another controller, no matter what its priority is, will be ignored. Thus, according to the timing of the reception of requests an implicit tree of pending operations is formed. This tree has a tendency to follow the address I-tree shape. Usually, independently of the number of controllers that are trying to perform the operation concurrently, the ordering tree shape is deep. Therefore, the request reissue after reordering is lazy; only one pending memory transaction is reissued after the completion of a write in most cases.

3.2.2 *Priority Ordering with Out-of-order processors*

Statically assigned priorities could provoke pathological situations because contended blocks will be obtained most often by the same processor. Nevertheless, assuming multiple outstanding requests per core, there is an easy and scalable solution to deal with this if we are capable of guaranteeing that: (a) Two different processors cannot issue an operation to the same block with the same priority; (b) The probability of having a different priority ordering at two contended blocks from two different processors has to be non zero.

The first condition guarantees that two different processors will never simultaneously grab a subset of tokens from the same block, i.e. avoiding starvation. The second condition guarantees that, on average, there will not be any memory operation favored over others. The most straightforward way to achieve this is to construct the priority of each request as the combination of the processor ID (LSB bits) used to achieve condition one, and a small random number (MSB bits) to achieve condition two. Experimentally, it is observed that this approach provides similar performance to an age-based priority (which requires a complex coordinated timestamp-based mechanism) at a fraction of the cost.

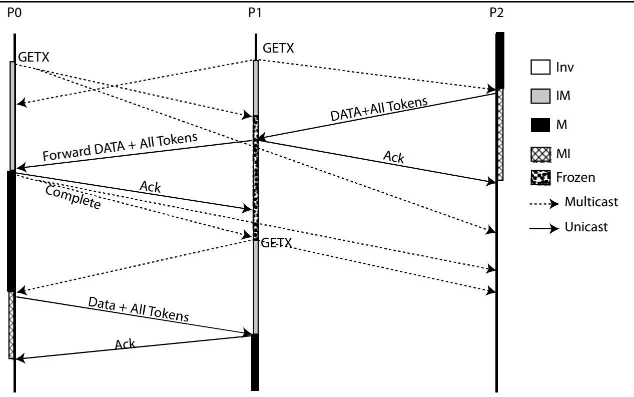


Figure 5. Example of write serialization: P0 and P1 simultaneously issue a GETX over a block in M[3] state at P2 (i.e., all the tokens are located there). Let's assume P0 has higher priority than P1. P1's request arrives at P2 first, so P2 sends data and all tokens, changing its state block to transitory state MI until the acknowledgement from P1 is received. Before receiving the data and the tokens, P1 sees a request from processor P0 which has more priority than its own priority, so it self-freezes its operation and annotates P0 as the winner at the MSHR. When data and tokens from P2 arrive, they are immediately forwarded to the winner P0, annotating the in-flight tokens. When P0 receives the data and tokens it sends an acknowledgement to P1 and finalizes its operation. When P0's GETX operation ends, it broadcasts a complete message. P1's MSHR hit unfreezes the operation and reissues it.

The performance reduction observed for a four-bit random number compared to an idealized fully age-based approach is less than 1%.

4. EVALUATION METHODOLOGY

4.1 Target System Configuration

In order to validate the advantages of our proposal, we have used two coherence protocols for the given system architecture with the configuration parameters shown in Table 1. The main parameters of the target system mimic state-of-the-art high-end CMPs such as [6][28][18]. The baseline coherence protocol used is an optimized directory protocol similar to the one used to compare the token coherence protocol in [25], but adapted to NUCA. Directory information is distributed across all slices and full mapping. Optimistically, null storage overhead is assumed for this protocol. Broadcast-based token coherence protocol variation [22] is considered, as a representative counterpart to snoop-based protocols. A fixed timeout to reissue the request is used. Only one reissue is tolerated before triggering a persistent request. This timeout has been selected measuring all the benchmarks with different timeouts and choosing the one with best average performance. Dynamically estimated time-out does not provide performance benefits.

For the cache hierarchy we will assume NUCA [14] for the last level cache. Although LOCKE is also applicable for a tiled system, NUCA architecture is a better approach because it decouples the number of LLC cache slices from the number of cores, providing much more flexibility to scale on-chip bandwidth. This cache architecture is currently a mainstream choice in high-end CMPs.

As far as the interconnection network is concerned, we will add the minimum variation over a commonly used router microarchitecture and network topology. As for the router micro-architecture used, it will be similar to the proposal described in [10], using on-network multicast support when required. We use dynamic buffering allocation per virtual channel and 1-cycle pipeline pass-through latency. In each protocol we use the required number of virtual channels to avoid message-dependent deadlock [31] and network deadlock [9].

In order to observe the scalability of the proposal we chose two different system sizes composed of 8 and 16 processors. The eight-processor system layout is similar to the one shown in Figure 4. For 16 cores, although the processor, L1 and router remain unchanged, the LLC capacity and bandwidth are scaled up in accordance with the larger number of processors [28].

In all configurations, instead of using in-order cores, we opted for aggressive out-of-order processors to mimic [18][28]. Although a large number of small cores could make sense for cloud-computing workloads, in general purpose computing, an aggressive processor microarchitecture still matters [12]. Additionally, medium size systems with a large number of outstanding memory transactions per processor will be much more demanding for the coherence protocol than many simple cores.

4.2 Simulation Stack

We work with a framework that allows us to perform full-system evaluation, i.e. user and system level code are accurately simulated. The main strategy of the proposal is that network bandwidth can be used cleverly to boost system performance; therefore, careful interconnection network modeling is essential. In order to achieve this requirement, SICOSYS [29] replaces the original interconnection network simulator of the full-system simulation tool GEMS [23]. SICOSYS models router micro-architecture very precisely. Therefore, network contention effects induced by extra traffic will be precisely modeled in the simulations. SICOSYS is augmented with Orion [17] in order to estimate the network energy consumption with each protocol. Power consumption in other components of the memory hierarchy is computed using CACTI [36]. The energy required by the cores will be assumed, pessimistically for performance leaders, to be constant across the different coherence protocols. The energy of each event is estimated assuming 32 nm technology.

Table 1. Basic system configuration, 32 nm. technology assumed for energy estimations

Processor Config.	Number of cores	8 @3GHz(config1) ***16 @3GHz(config2)***
	Functional Units	4xI-ALU/4xFP-ALU/ 4xD-MEM
	IWin size/Issue Width	128, 4-way
	Fetch-to-Dispatch	7 cycles
L1 Cache	Size/Associativity/ BlockSize/Access Time	128KB I/D, 4-way, 64B, 2 cycles
	Max. Outstanding Mem. Operations	16
L2 Cache	Size/Associativity/ BlockSize	8MB, 16×512KB, 16-way, 64B (config1) ***16MB, 32×512KB, 16-way, 64B (config2)***
	NUCA Mapping	Static, interleaved across slices
	Slice Access Time	5 cycles
Memory	Capacity/Access Time/ Memory Controllers/ BW	4GB, 240 cycles,2 centered / 32GBs (config 1) ***4GB, 240 cycles, 4 centered/64GBs (config 2)***
Network	Topology / Link Latency/ Link Width	4×4 Mesh, 1 cycle, 16B (config 1) ***6×6 Mesh,1cycle, 16B (config 2)***
	Router Latency/ Flow Control/ Buffering Size / Routing	1 cycle/ Wormhole/ 5.4KB/DOR

4.3 Workloads

Fifteen workloads (see Table 2) are considered in this study, including both multi-programmed and multi-threaded applications (numerical and server) running on top of the OpenSolaris 10 OS.

Table 2. Evaluated workloads

Multithreaded Workloads	Wisconsin Commercial Workload [3]	Apache (1000 Surge dynamic)	Zeus (1000 Surge Static)
		Jbb (4000 SpecJbb)	OLTP (500 TPC-C alike)
	NAS Parallel Bench. [16]	FT (Class W)	CG (Class A)
		LU (Class A)	IS (Class A)
	PARSEC [5]	blackscholes (native)	
		canneal (native)	
		fluidanimate (native)	
Multipro-grammed Workloads	Spec 2006 [32] (Rate Mode)	astar (reference)	hmmer (reference)
		lbm (reference)	ommetpp (reference)

The server benchmarks correspond to the whole Wisconsin Commercial Workload suite [3], released by the authors of GEMS in version 2.1. 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. Each application is simulated multiple times with random perturbations in memory access time in order to reach 95% confidence intervals. The number of applications enables the sweeping of a broad spectrum of usage scenarios, with diverse sharing degree, sharing contention, working set size, etc.

5. EVALUATION

5.1 Performance and Efficiency

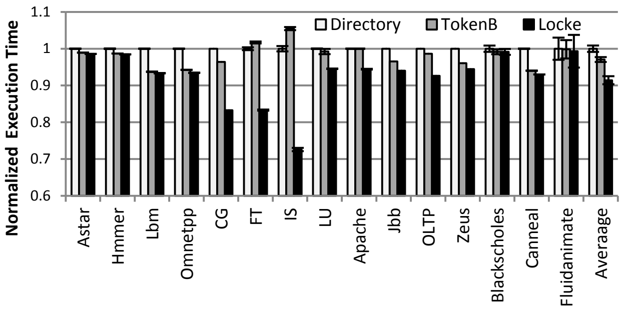
Figure 6 shows the performance with the basic 8-processor CMP (*config1* in Table 1). On average, DIRECTORY is outperformed by LOCKE and TOKEN. As expected, some workloads are insensitive, which attenuates the average performance impact of coherence protocol. In contrast, in applications with highly contended blocks, such as NAS benchmarks, coherence is very relevant. In those cases, LOCKE outperforms other protocols by up to almost 30%. In applications with high sharing degree but limited contention, such as server workloads, LOCKE outperforms the other counterparts by a smaller but noticeable margin. Although, on average, TOKEN performs better than DIRECTORY some noticeable results such as IS or FT, even in a modest size system like this one, show its performance is poor for the reasons explained in Section 2.3. In contrast, LOCKE exhibits a consistent performance across all the workloads.

End-point traffic comparison of different protocols may not reflect a direct impact in performance or energy profile. First, when the network uses capable routers, as in our case, a multicast packet with *n* destinations will not use the same effective bandwidth as *n* unicast packets for the same destination [10]. This issue will be considered again later in subsection 5.4. Second, network energy is only a part of the on-chip memory hierarchy which is dominated by cache. Third, Energy Delay Square Product (ED2P) is the most suitable metric to estimate energy-performance tradeoff in high-performance systems such as ours [38]. Therefore, we provide this metric, grouped for each suite of benchmarks and protocols in Figure 6 (b). As we can see, the cubic influence of performance in ED2P has a major effect, meaning that the ED2P of the network, in spite of producing more traffic, is even smaller for broadcast-based protocols. Additionally, for 32nm technology and a large cache footprint (8MB in this configuration), leakage power, which is constant across coherence protocols, causes the ED2P leakage proportion to grow significantly when the performance is worse. Therefore, and contrary to common belief, snoop-based broadcast coherence protocols have lower average ED2P than the one based on directories. Due to the more consistent LOCKE performance, on average it requires 19% less ED2P than DIRECTORY. In contrast, due to performance instabilities, TOKEN is only capable of saving 7%.

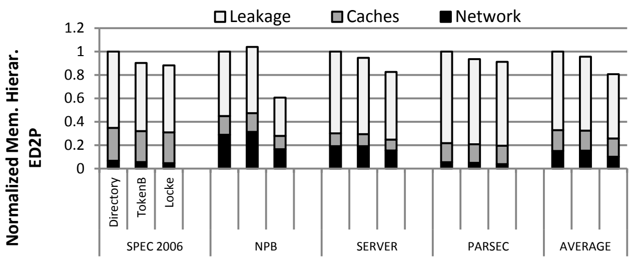
5.2 System Scalability

To explore the scalability of each protocol, system size has been increased to 16 cores (*config2* in Table 1). Using a higher number of processors, given their architectural complexity, would imply an unattainable computational cost for the simulations. Additionally, most of the benchmarks do not scale well beyond sixteen cores. In any case, given that each processor can have up to 16 pending memory transactions, the system could have up to 512 simultaneous coherence actions, including L1 misses and write-backs. This could be four times more demanding for the coherence protocol than having 64 in-order cores.

The performance observed in Figure 7 (a) indicates that LOCKE is able to increase its advantage in comparison to DIRECTORY. Scaling up the network size to accommodate NUCA slices would increase the cost of DIRECTORY indirections. Nevertheless, the

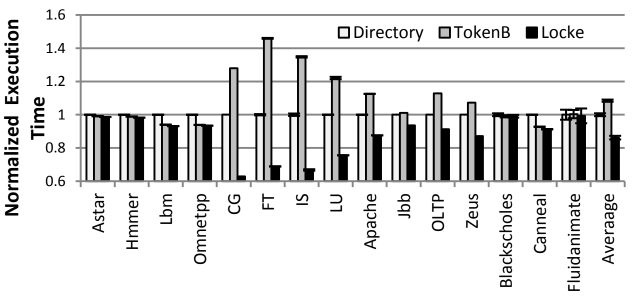


(a)

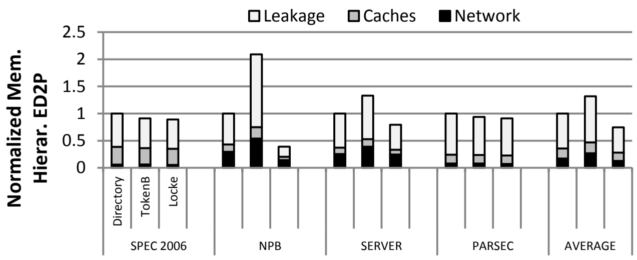


(b)

Figure 6. Directory normalized 8-processor CMP. (a) Execution Time. (b) Memory Hierarchy ED2P.



(a)



(b)

Figure 7. Directory Normalized 16-Processor CMP. (a) Execution Time. (b) Memory Hierarchy ED2P.

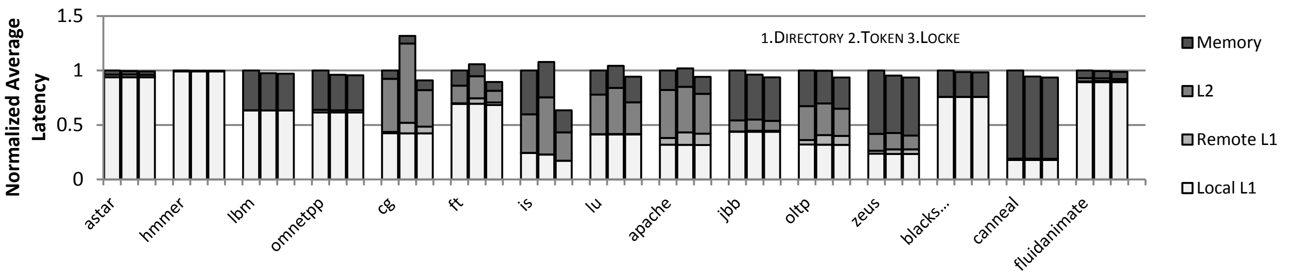


Figure 8. Directory Normalized Average Latency for 8-Processor system.

increased contention due to larger numbers of multicast destinations seems not to increase the latency in the network significantly for LOCKE. Therefore, the performance advantage of LOCKE over the directory is now greater than in 8-core CMP (16%). In contrast, TOKEN performs poorly, being noticeably slower than DIRECTORY.

5.3 Coherence Protocol Responsiveness

In order to provide insight into protocol effectiveness, Figure 8 shows the average latency perceived by the processor with each protocol for an 8-processor system.

As can be appreciated in both systems, the DIRECTORY-based protocol has a larger memory contribution in some applications. This is a direct consequence of inclusiveness. Whereas snoop-based protocols do not need inclusiveness to track on-chip block sharers, DIRECTORY requires an entry in LLC for all L1 cached blocks. Consequently, the effective cache capacity is larger in the former protocols, raising LLC miss rate. This problem is acknowledged as a serious drawback of directory coherence protocols [7]. Serialization with directory reduces on-chip hit latency in applications where network contention is not significant. TOKEN coherence introduces pressure on the network in some applications and the starvation avoidance mechanism increases the on-chip hit latency significantly, making the average access time up to 40% slower in applications such as IS. In contrast, LOCKE seems to consistently outperform other protocols in most applications.

Although, on-chip hit latency provides a good idea about protocol efficiency, it might be interesting to isolate how the protocol reacts when multiple coherence events arise simultaneously for the same block. In these situations, the effectiveness of the protocol is the key to prompt resolution of the situation. Figure 9 shows how effective each protocol is when dealing with true racing requests in eight processor systems. As we can see, in most cases LOCKE is the fastest one, being on average 10% faster than DIRECTORY and 60% faster than TOKEN. Token's persistent mechanism to resolve those situations makes it the slowest one, being on average 40% slower than DIRECTORY. Although not shown, for sixteen-core systems the advantage of LOCKE is even greater. With non-conflicting coherence events broadcast-based coherence protocols are faster than directory due to inclusiveness, which increases on-chip miss rate, as can be appreciated in the memory contribution in Figure 8.

5.4 Network Energy Impact of Multicast Traffic

As stated before, it is commonly assumed that multicast traffic has a large impact on network power consumption. This assumption is based on the large increment in control traffic observed at the end-point, i.e. consumers. Nevertheless, when a network has multicast support, i.e. on-network packet replication, this is completely wrong because multicast packets use network resources only once before replication [10][1]. Therefore, unlike unicast-only networks, energy consumption is not proportional to end-point traffic, but to average link utilization. For example, Figure 10 shows the directory normalized network link utilization for LOCKE and TOKEN for eight- and sixteen-processor CMPs. All the links in the interconnection networks have been considered, including the connections from routers to L1 caches, L2 slices and memory controllers.

As we can appreciate, and contrary to common belief, network activity in snoop-based protocols is not much higher than directory protocols. Multicast capable routers have an identical data-path to conventional ones [10][1], so normalized link utilization differences will be translated into energy consumption (and negligible implementation cost). In all cases, LOCKE has lower link activity because the multicast tree used is much deeper than the one used in TOKEN, which tries to reach all the destinations as soon as possible. As indicated in Section 3.1

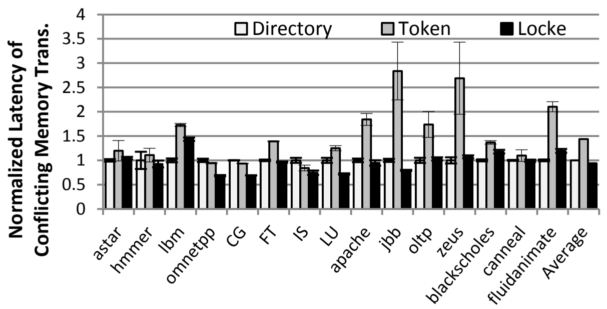


Figure 9. Normalized time to resolve conflicting memory accesses for an 8-processor CMP.

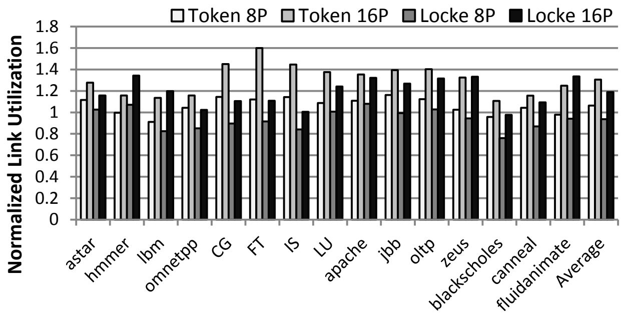


Figure 10. Directory Normalized Average Network link utilization.

LOCKE ordering I-trees delay packet replication, which increases request base latency, but reduces network activity. With particularly demanding applications, such as most NPB benchmarks, or bigger system sizes, TOKEN starvation avoidance increases the amount of activity. Even in the largest system, network system activity is only 15% greater in LOCKE than in DIRECTORY. Performance benefits offset this, making LOCKE the most efficient coherence protocol. With small size, the system DIRECTORY generates more network activity than snoop-based protocols due to protocol indirections and the larger number of on-cache misses.

6. RELATED WORK

There are numerous proposals for snoop-based on-chip coherence protocols over unordered networks in both academic [2][22][30][33] and commercial architectures [28][19]. In our proposal, two main characteristics can be seen. On the one hand, our method fulfills its main aim: to maintain all the advantages of token counting while eliminating the drawbacks of the persistent request method. Working in the same direction, the authors of the original token coherence protocol proposed PATCH [30]. This work extends a standard directory protocol to track tokens and uses token counting rules for enforcing coherence permissions. Token counting allows PATCH to support direct requests on an unordered interconnect, while a mechanism called "token tenure" uses local processor timeouts and the directory's per-block point of ordering at the home node to guarantee forward progress without relying on broadcasting. Nevertheless, this solution still depends on a suitable choice of the timeout used. Our proposal, offers the advantage of not needing fixed delays in order to take coherence decisions, which improves protocol responsiveness and stability. We believe that no quantitative performance comparison with PATCH is strictly necessary, because according to its authors, PATCH performance is between TOKEN and DIRECTORY, and LOCKE is better than both. It should be noted that the main motivation of PATCH is to reduce end-point traffic, which it successfully achieves. Nevertheless, as background motivation and according to the results presented in this paper, for on-chip systems it does not seem to be a fundamental issue.

On the other hand, to maintain system scalability, unordered networks are essential, but coherence protocols should provide a method to offer some kind of ordering of requests. There are many proposals where different solutions are provided. One of the possibilities is to establish this order by adding information to the requests sent to facilitate their ordering at their destination. In Timestamp Snooping [24], a coherence protocol is proposed in which a logical timestamp is added to all requests. These requests are sent out-of-order and put back into order at their destination.

Ordered requests may also be accomplished by using a physical structure through which requests may travel. One of the most simple and low-cost approaches is to embed a ring in the network and use it to transfer snoop messages. In order to address possible long response latencies or too many snoop operations, adaptive snooping algorithms are proposed in Flexible Snooping [34]. In these algorithms, depending on the chance of providing the line, a node receiving a snoop request will snoop first and then forward the request, or forward first and then snoop the operation. Moreover, if the node can prove that it will not be able to provide the line, it will skip the snoop operation, forwarding the request directly to the next node in the ring. However, any protocol using a ring as an interconnection network (either logical or physical) needs to send its snoop requests through it, forcing all requests to visit every node, which obviously means the loss of any possibility of parallelism with the snoop requests, thus increasing transaction latency. Trying to solve this problem, Unconstrained Snoop Request Delivery (UNCORQ) was proposed [33]. While requests are delivered using any network path, responses use the logical ring. A similar ring-approach, but inspired by token coherence, was used in [26].

Similarly to LOCKE, Virtual Tree Coherence [11] uses a tree to maintain ordering. This coherence protocol keeps track of sharers of a coarse-grained region, and multicasts them through a virtual tree to enforce ordering. Virtual trees are embedded inside a physical network of the topology. The root of the virtual tree provides an ordering point needed to order requests. Note that other works, such as [8], use trees with Token coherence, however, not for ordering but to avoid deactivations in persistent requests. Therefore, the lack of responsiveness and potential instabilities that persistent requests incur are still present.

7. CONCLUSIONS

This work presents a new coherence protocol suitable for CMP on-chip interconnection network characteristics. The protocol augments token coherence protocol, avoiding potential instabilities induced by workloads or system configuration. The increased robustness is accompanied by a performance benefit. LOCKE can proactively separate true data sharing and synchronization among cores from spurious data movements. Clearly LOCKE's benefits compensate its shortcomings, achieving a great scalability with aggressive out-of-order processors. In conclusion, LOCKE is competitive in terms of performance and energy footprint, not only with token but also with directory. LOCKE clearly demonstrates that the utilization of snoop-based coherence protocols is a suitable choice for managing CMP coherence, at least in systems with up to 16 aggressive out-of-order processors in the chip.

8. ACKNOWLEDGMENTS

The authors would like to thank José Ángel Herrero for his invaluable assistance with the computing environment, and the anonymous reviewers for many useful suggestions. This work has been supported by the MICCIN (Spain) under contract TIN2010-18159 and the HiPEAC European Network of Excellence.

9. REFERENCES

[1] P. Abad, V. Puente, and J.-A. Gregorio. MRR: Enabling fully adaptive multicast routing for CMP interconnection networks. In *15th Int S High Perf Comp (HPCA)*, 355-366, 2009.

[2] N. Agarwal, L.-S. Peh, and N. K. Jha. In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects. In *15th Int S High Perf Comp (HPCA)*, 67-78, 2009.

[3] A. R. Alameldeen et al. Simulating a $2M Commercial Server on a $2K PC. *Computer*, vol. 36, 50-57, 2003.

[4] K. Asanovic et al. *The Landscape of Parallel Computing Research: A View from Berkeley*. Technical Report. EECS Dept. U. of California Berkeley, vol. 18, no. UCB/EECS-2006-183, 2006.

[5] C. Bienia and K. Li. PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors. In *MoBS*, 2009.

[6] M. Butler. AMD 'Bulldozer' Core - a new approach to multithreaded compute. In *HOT Chips* 22, 2010.

[7] P. Conway et al. Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor. *IEEE Micro*, vol. 30, no. 2, 16-29, 2010.

[8] B. Cuesta, A. Robles, and J. Duato. An effective starvation avoidance mechanism to enhance the token coherence protocol. In *15th Euromicro Conf Proc*, 47-54, 2007.

[9] J. Duato. A theory of deadlock-free adaptive multicast routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 9, 976-987, 1995.

[10] N. D. Enright Jerger, L.-S. Peh, and M. Lipasti. Virtual Circuit Tree Multicasting: A Case for On-Chip Hardware Multicast Support. In *Int S Comp Arch (ISCA)*, 229-240, 2008.

[11] N. D. Enright Jerger, L.-S. Peh, and M. H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *41st Int Symp Microarch*, 35-46, Nov. 2008.

[12] M. D. Hill and M. R. Marty. Amdahl's Law in the Multicore Era. *Computer*, vol. 41, no. 7, 33-38, Jul. 2008.

[13] H. P. Hofstee. Power Efficient Processor Architecture and The Cell Processor. In *Int S High Perf Comp (HPCA)*, 258-262, 2005.

[14] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler. A NUCA substrate for flexible CMP cache sharing. In *19th Int Conf Supercomputing (ICS)*, 31-40, 2005.

[15] ITRS. Roadmap 2010.

[16] 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.

[17] A. B. Kahng et al. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Design, Automation & Test*, 423-428, 2009.

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

[19] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. The AMD Opteron processor for multiprocessor servers. *IEEE Micro*, vol. 23, no. 2, 66-76, 2003.

[20] K. Lee, S.-joong Lee, and H.-jun Yoo. Low-power network-on-chip for high-performance SoC design. In *IEEE Trans. on Very Large Scale Int. (VLSI) Systems*, vol. 14, no. 2, 148-160, 2006.

[21] D. Lenoski et al. The Stanford Dash multiprocessor. *Computer*, vol. 25, no. 3, 63-79, Mar. 1992.

[22] 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, 108-116, 2003.

[23] M. M. K. Martin et al. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News*, vol. 33, 4, Nov. 2005.

[24] M. M. K. Martin et al. Timestamp Snooping: An Approach for Extending SMPs. In *Architectural Support for Prog. Lang. and O. Systems (ASPLOS)*, vol. 1, no. 212, 1-12, 2000.

[25] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood. Improving Multiple-CMP Systems Using Token Coherence. In *11th Int S High Perf Comp (HPCA)*, 328-339, Feb 2005.

[26] M. Marty and M. Hill. Coherence Ordering for Ring-based Chip Multiprocessors. In *39th Int Symp Microarch (MICRO)*, 309-320, 2006.

[27] L.G. Menezo, V. Puente, JA. Gregorio. *Locke Formal Specification Tables*. Technical Report. Available online: http://sg.sg/GPGFef. 2011.

[28] C. Park et al. A 1.2 TB/s on-chip ring interconnect for 45nm 8-core enterprise Xeon® processor. In *2010 IEEE International SolidState Circuits Conference(ISSCC)*, 180-181, 2010.

[29] V. Puente, J. A. Gregorio, and R. Beivide. SICOSYS: An Integrated Framework for Studying Interconnection Network Performance in Multiprocessor Systems. *IEEE Comput. Soc*, pp. 15-22, 2002.

[30] A. Raghavan, C. Blundell, and M. M. K. Martin. Token tenure: PATCHing token counting using directory-based cache coherence. In *41st Intl S Microarch*, 47–58, Nov. 2008.

[31] Y. H. Song and T. M. Pinkston. Efficient handling of message-dependent deadlock. In *15th Int Parallel & Distributed Proc Symp (IPDPS)*, 2001.

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

[33] K. Strauss, X. Shen, and J. Torrellas. Uncorq: Unconstrained Snoop Request Delivery in Embedded-Ring Multiprocessors. In *40th Int S Microarch (MICRO)*, 327-342. 2007.

[34] K. Strauss, X. Shen, and J. Torrellas. Flexible Snooping: Adaptive Forwarding and Filtering of Snoops in Embedded-Ring Multiprocessors. In *33rd Int S Comp Arch (ISCA)*, 327-338, 2006.

[35] M. Suleman, O. Mutlu, M. Qureshi, and Y. N. Patt. Accelerating critical section execution with asymmetric multi-core architectures. In *14th Intl. Conf. on Architectural Support for Progr. Lang. and OS (ASPLOS)*, 253–264, 2009.

[36] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, *CACTI 4.0*. 2006.

[37] A. W. Topol et al. Three-dimensional integrated circuits. IBM J. of Research and Development, vol. 50, no. 4, 491-506, Jul. 2006.

[38] V. Zyuban, and P. Kogge. Optimization of high-performance superscalar architectures for energy efficiency. In *Intl S on Low Power Electronics & Design*, 84-89, 2000.