Techniques for Improving the Performance and Scalability of Directory-based Shared-Memory Multiprocessors: A Survey

Manuel E. Acacio and **José M. García**
Dept. Ingeniería y Tecnología de Computadores
Universidad de Murcia, SPAIN
E-mail: {meacacio, jmgarcia}@ditec.um.es

ABSTRACT

Cache-coherent, nonuniform memory access or cc-NUMA is an attractive architecture for building a spectrum of shared memory multiprocessors (which are seeing widespread use in commercial, technical and scientific applications). Unfortunately, there are some factors which limit the maximum number of processors that can be offered at a good price/performance ratio. This paper presents a survey of some of the proposals that have recently appeared focusing on two of these factors: the increased cost in terms of hardware overhead that the use of directories entails, and the long cache miss latencies observed in these designs as a consequence of the indirection introduced by the access to the directory.

Keywords: cc-NUMA Multiprocessors, Scalability, Cache Coherence, Directory Memory Overhead, Cache Miss Latency

1. INTRODUCTION

The key property of shared-memory multiprocessors is that communication occurs implicitly as a result of conventional memory access instructions (i.e., loads and stores) which makes them easier to program and thus, more popular than message-passing machines.

Shared-memory multiprocessors cover a wide range of prices and features, from commodity SMPs to large high-performance cc-NUMA machines. Most shared-memory multiprocessors employ the cache hierarchy to reduce the time needed to access memory by keeping data values as close as possible to the processor that uses them. Since multiple copies of a data value may co-exist in different caches, these machines implement a coherence protocol (generally, a write-invalidate coherence protocol) to ensure consistency among these copies.

The adopted solutions to the coherence problem are quite different depending on the total number of processors. For systems with small processor counts, a common bus is usually utilized along with snooping cache coherence protocols. Snooping protocols [20] solve the cache coherence problem using a network with a completely ordered message delivery (traditionally a bus) to broadcast coherence transactions directly to all processors and memory. Unfortunately, the broadcast medium becomes a bottleneck (due to the limited bandwidth that it provides and to the limited number of processors that can be attached) preventing them from being scalable.

Instead, scalable shared memory multiprocessors are constructed based on scalable point-to-point interconnection networks, such as a mesh or a torus [18]. Besides, main memory is physically distributed in order to ensure that the bandwidth needed to access main memory scales with the number of processors. In these designs, accessing main memory has nonuniform access costs to a processor and in this way, architectures of this type are often called *cache-coherent, nonuniform memory access* or cc-NUMA architectures. The best-known example of a commercial multiprocessor using this approach is the SGI Origin 2000 [37].

In these organizations, totally ordered message delivery becomes infeasible and cache coherence is based on the concept of a *directory* [12], which is an structure used to keep explicitly the state of every memory line. Directory entries are distributed along with the memory, so that different directory accesses can go to different locations, just as different memory requests go to different memories. Each memory line is assigned to a directory (the *home directory*), which keeps a directory entry for the memory line. A natural way to organize directories is to maintain the directory information for each memory line together with the line in main memory at the home node. Each directory entry is comprised of two main fields: a state field, used to store the state of the line, and a *sharing code* [45] field, used to identify those caches currently holding a copy of the line. The majority of the bits of each directory entry are devoted to codifying the sharing code and, therefore, its election directly affects the memory required for the directory. A simple organization for the sharing code is as a bit-vector of N presence bits, which indicate for each of the N nodes whether that node has a

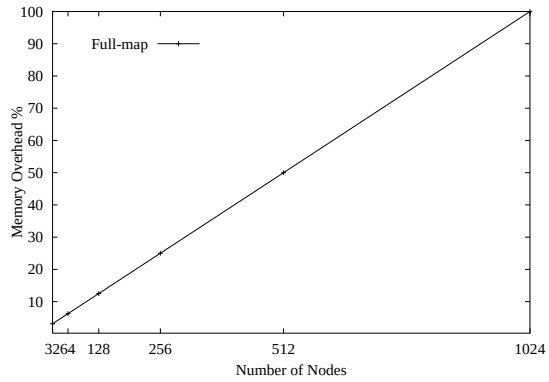

Figure 1. Memory Overhead for Full-Map

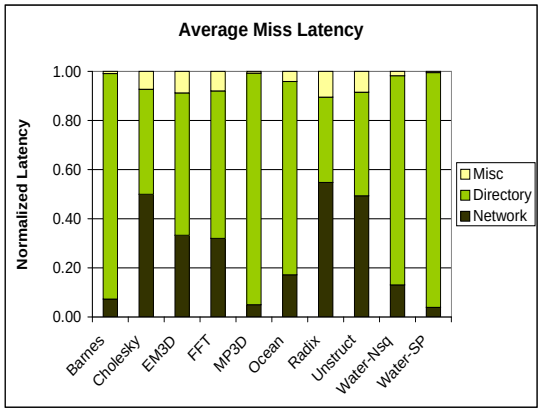

Figure 2. Average cache miss latency

cached copy of the line or not (note that each node may be a uniprocessor or a multiprocessor system). This organization of the sharing code is called *full-map*, Dir_N or *bit-vector* [17] and some current multiprocessors, such as the SGI Origin 2000 [37] or the Stanford DASH Multiprocessor [39], directly use it or some variations on it.

Although the use of directory-based coherence protocols allows multiprocessor designers to orchestrate shared memory multiprocessors with hundreds of nodes, the implementations of the shared memory paradigm have limited scalability, then becoming infeasible for very large-scale systems, which use the message-passing paradigm. Examples of such machines are the ASCI Red, the ASCI Blue Pacific and the ASCI White multiprocessors.

There are several factors limiting the scalability of cc-NUMA designs. Two of the most important of these issues are, first, the cost in terms of the hardware overhead that the use of directories implies, and, second, the increased distance to memory, which is the reason for the higher cache miss latencies that are currently being observed in cc-NUMA architectures.

The most important component of the hardware overhead is the amount of memory required to store the directory information, particularly the sharing code. Depending on how the sharing code is organized, memory overhead for large-scale configurations of a parallel machine could be intolerable. For example, for a simple full-map sharing code and for a 128-byte line size, Figure 1 draws directory memory overhead (measured as sharing code size divided by memory line size) as a function of the number of system nodes. As observed, directory memory overhead for a system with 128 nodes is 12.50%, which is not too bad. However, when the node count reaches 1024, this overhead becomes 100%, which is definitely prohibitive.

On the other hand, long cache miss latencies of directory protocols are caused by the inefficiencies that the distributed nature of the protocols and the underlying scalable network imply. Currently, and as a consequence of the increased distance to memory, the most important of such inefficiencies is the indirection introduced by the access to the directory, which is usually stored in main memory. This represents a unique feature of directory protocols, not present in SMPs. The consequences of such indirection are particularly serious for cache-to-cache transfer misses and upgrade misses, which constitute the most important fraction of the total cache miss rate [3, 9]. Figure 2 presents the normalized average miss latency obtained when running several applications on top of a simulated 64-node cc-NUMA multiprocessor using RSIM [29]. Average miss latency is split into network latency, directory latency and miscellaneous latency (buses, cache accesses...). As can be observed, the most important fraction of the time needed to satisfy a certain request is spent at the directory in almost every application. Therefore, techniques for reducing the directory component of the miss latency will be rewarding.

In this paper, we present a revision of the proposals that have recently appeared facing these two important issues in cc-NUMA multiprocessors. First of all, in Section 2 we summarize several organizations for the directory aimed at reducing directory memory overhead. Then, in Section 3 we depict some important research efforts for accelerating cache misses in these architectures. Finally, Section 4 concludes the paper and presents some future ways.

2. REDUCING DIRECTORY MEMORY OVERHEAD

There are two main alternatives for storing directory information [17]: *flat, memory-based* directory schemes and *flat, cache-based* directory schemes. In flat, memory-based directory schemes the home node maintains the identity of all the sharers and the state, for every one of its memory lines. On the contrary, flat, cache-

based directory protocols (also known as chained directory protocols), such as the IEEE Standard Scalable Coherent Interface (SCI) [23], rely on distributing the sharing code among the nodes of the system. For every one of its memory lines, the home node contains only a pointer to the first sharer in the list plus a few state bits. The remaining nodes caching the line are joined together in a distributed, doubly linked list, using additional pointers that are associated with each cache line in a node (which are known as forward and backward pointers). The locations of the copies are therefore determined by traversing this list via network transactions.

The most important advantage of flat, cache-based directory protocols is their ability to significantly reduce directory memory overhead. In these protocols, every line in main memory only has a single head pointer. The number of forward and backward pointers is proportional to the number of cache lines in the machine, which is much smaller than the number of memory lines. Although some optimizations to the initial proposal have been studied (for example, in [14] and [49]) and several commercial machines have been implemented using this kind of protocols, such as the Sequent NUMA-Q [40] and Convex Exemplar [16] multiprocessors, the tendency seems to have changed these days and from the SGI Origin 2000 [37] onwards, most designs use memory-based directory protocols, such as Piranha [10], the AlphaServer GS320 [19] or the Cenju-4 [28]. Other examples of shared memory multiprocessors that have been constructed based on such a directory organization are the Stanford FLASH and DASH multiprocessors [34, 39], the MIT Alewife [6] and the HAL-S1 [60], among others. The decreased popularity of cache-based directory protocols is a consequence of some important drawbacks they introduce: the increased latency of coherence transactions as well as occupancy of cache controllers, and complex protocol implementations [17]. A comparison between flat, memory-based directory protocols and flat, cache-based ones can be found in [27].

On the other hand, memory overhead in flat, memory-based directory schemes is usually managed from two orthogonal points of view: reducing directory *width* and reducing directory *height*.

Reducing directory *width*

Some authors propose to reduce the width of directory entries by having a limited number of pointers per entry to keep track of sharers [7, 13, 55]. The differences between them are mainly found in the way they handle overflow situations, that is to say, when the number of copies of the line exceeds the number of available pointers [17]. As an example, $\mathrm{Dir}_i\mathrm{B}$ sharing code [7] provides i pointers to codify up to i potential sharers. It was inspired by experimental data suggesting that, in many cache invalidation patterns, the number of sharers is very low and a few number of pointers may be sufficient for the majority of cases [21]. When the number of available pointers i is exceeded, a broadcast bit in the directory entry is set. On a subsequent write operation, invalidation messages will be sent to all the nodes in the system, regardless of whether or not they are caching the line. Two interesting instances of this sharing code are $\mathrm{Dir}_1\mathrm{B}$ and $\mathrm{Dir}_0\mathrm{B}$. Whereas the former needs $1+\log_2 N$ bits, the latter does not use any bit and always sends *N-1* coherence messages (invalidations or cache-to-cache transfer requests) when the home node cannot directly satisfy a certain cache miss (i.e., on a coherence event), for a N-node system.

More recently, the segment directory has been proposed as an alternative to the limited pointer schemes [15]. The segment directory is a hybrid of the full-map and limited pointers schemes. Each entry of a segment directory consists of two components: a segment vector and a segment pointer. The segment vector is a K-bit segment of a full-map vector whereas the segment pointer is a $log_2(N/K)$-bit field keeping the position of the segment vector within the full-map vector, aligned in K-bit boundary. Using directory's bits in this way results in a reduction of the number of directory overflows suffered by limited pointer schemes.

Other proposals reduce directory width by using *compressed sharing* codes. Unlike the well-known full-map sharing code, compressed sharing codes require a lower number of bits and achieve better scalability by storing an *in-excess* representation of the nodes that hold a memory line. Since more sharers than necessary are usually codified, this kind of sharing codes, which are also known as *multicast* protocols [45] and *limited broadcast* protocols [7], leads to the appearance of *unnecessary coherence messages*. An excessive number of these messages has dramatic consequences on the final performance. Several compressed sharing code schemes have been proposed in the literature with a variety of sizes and precisions. Some of the them are *coarse vector* [22], which is currently employed in the SGI Origin 2000 multiprocessor, *tristate* [7], *gray-tristate* [45], and *binary-tree with subtrees* [2].

Unlike full-map sharing code, in coarse vector each bit of the sharing code stands for a group of K processors. The bit is set if any of the processors in the group (or some of them) cached the memory line. Thus, for a N-node system, the size of the sharing code is N/K bits. Tris-

tate, also called the superset scheme by Gupta *et al.* [22], stores a word of d digits where each digit takes on one of three values: 0, 1 and *both*. If each digit in the word is either 0 or 1, then the word is the pointer to exactly one sharer. If any digit is coded *both*, then the word denotes sharers whose identifier may either be 0 or 1 in that digit, but match the rest of the word. If i digits are coded *both*, then 2^i sharers are codified. In this way, it is possible to construct a superset of current sharers. Each digit can be coded in 2 bits, thus requiring $2\log_2 N$ bits for a N-node system. Gray-tristate improves tristate in some cases by using Gray code to number the nodes. Finally, binary-tree with subtrees requires approximately $\log_2 N$ bits for a N-node system, and it is based on the distinction between the logical structure of the system (i.e, how the system *seems* to be organized) and the physical one (how the system actually is). The codification of the sharers is performed based on the logical structure of the system. In this case, the logical system is a binary tree with the nodes located at the leaves, and the sharing code codifies the two minimal subtrees that include all the sharers. One of them is computed from the home node. For the other one, one of the symmetric nodes of the home node is employed. Additionally, this sharing code solves the common case of a single sharer by directly encoding the identifier of that sharer.

Reducing directory *height*

Other schemes try to decrease directory memory overhead by reducing the total number of directory entries available. This can be achieved either by combining several directory entries in a single entry (*directory entry combining*) [54] or by organizing the directory as a cache (*sparse directory*) [22, 51]. The first approach tends to increase the number of coherence messages per coherence event as well as the number of cache misses in those cases in which several memory lines share a directory entry. On the other hand, the use of sparse directories also increases the number of cache misses as a result of *premature invalidations* that are sent each time a directory entry is evicted.

Everest [48] is an architecture for high performance cache coherence and message passing in partitionable distributed shared memory systems that use commodity SMPs as building blocks. In order to maintain cache coherence between shared caches included into every SMP, Everest uses a new directory design called Complete and Concise Remote (CCR) directory. In this design each directory maintains a *shadow* of the tags array of each remote shared cache. In this way, each directory consists of $N-1$ shadows for a N-node system, which prevents CCR directories from being a solution for scalable systems.

Reducing directory *width* and *height*

Finally, *two-level directories* are proposed in [1, 2] as a solution to the scalability problem which combines the benefits of compressed sharing codes and sparse directories in order to significantly reduce directory memory overhead without degrading performance. A two-level directory architecture consists of a small full-map first-level directory (which stores precise information for a small subset of the memory lines) and a compressed and complete second-level directory (which provides in-excess information for every line). Due to the temporal locality found for directory accesses, results show that a system with this directory architecture achieves the same performance as a multiprocessor with a big and non-scalable full-map directory, with a very significant reduction of the memory overhead. This approach can be generalized to a *multilevel directory* organization.

3. REDUCING CACHE MISS LATENCY

In most designs directory entries are stored in main memory, together with the memory lines they are associated with, which puts the cycles needed to access main memory into the critical path of cache misses. Unfortunately, a well-known industry trend is that microprocessor speed is increasing much faster than memory speed [25]. The increased distance to memory (the *memory wall* problem) raises the necessity of storing directory information out of main memory, in more efficient structures, or even more, of completely removing the access to the directory from the critical path of the cache misses.

Caching-based techniques

Caching directory information was originally proposed in [22] and [51] as a means to reduce the memory overhead entailed by directories. More recently, directory caches have also been used to reduce directory access times [34, 44, 47]. For example, Michael and Nanda [47] proposed the integration of directory caches inside the coherence controllers to minimize directory access time and, consequently, to reduce cache miss latencies. The Everest architecture proposed in [48] also uses directory caches to reduce directory access time. In addition, remote data caches (RDCs) have also been used in several designs (as [39], [40] and [50]) to accelerate access to remote data. A RDC is used to hold those lines that are fetched to the node from remote memory and acts as backup for the processor caches.

In [30], the remote memory access latency is reduced by placing caches in the crossbar switches of the interconnect to capture and store shared data as they flow from the memory module to the requesting processor. Subsequently, in [31] the same idea is applied to reduce the latency of cache-to-cache transfer misses. In this case, small directory caches are implemented in the crossbar switches of the interconnect medium to capture and store ownership information as the data flows from the memory module to the requesting processor. In both cases, special network topologies are needed to keep coherent the information stored in these switch caches.

Exploiting on-chip integration

Recent technology improvements have enlarged the number of components that can be included into a single chip. System-on-a-chip techniques allow for integration of all system functions including compute processor, caches, communications processor, interconnection networks and coherence hardware onto a single die. Directly connecting these highly-integrated nodes leads to a high-bandwidth, low-cost, low-latency "glueless" interconnect. Some proposals exploiting system-on-a-chip techniques are the Compaq Alpha 21364 EV7 [24], the IBM BlueGene/L supercomputer [58], and the AMD Hammer [8]. Even more, as more transistors can be placed on a chip, a sizeable fraction of the main memory is likely to be also integrated on chip (processor-in-memory or PIM chips) [59]. Other designs go further and use semiconductor technology trends to implement a complete multiprocessor into a single chip (multiprocessor-on-a-chip), for example the Compaq Piranha CMP [10], the Stanford Hydra [26] or the IBM POWER4 [57]. In [3], cache misses found in cc-NUMA multiprocessors are firstly classified in terms of the actions performed by directories to satisfy them, and then, it is proposed a novel node architecture that makes extensive use of on-processor-chip integration in order to reduce the latency of each one of the types of the classification.

Approaching snooping behavior

Other proposals have focused on using snooping protocols with unordered networks. In [42], Martin *et al.* propose a technique that allows SMPs to utilize unordered networks (with some modifications to support snooping). However, scalability to larger systems is still compromised since coherence transactions must be sent to all the processors in the system which, in turn, must process them. In addition, Bandwidth Adaptive Snooping Hybrid (BASH) has been proposed to reduce the requirements that snooping protocols put on the interconnect [43]. BASH is a hybrid protocol that ranges from behaving like snooping (by broadcasting coherence transactions) when excess bandwidth is available to behaving like a directory protocol (by unicasting coherence transactions) when bandwidth is limited. Additionally, Token Coherence is presented in [41] as a new coherence framework that tries to avoid the accesses to the directory in a system by directly sending coherence messages from the requesting node. To this end, Token Coherence has two parts: a correctness substrate which ensures safety (i.e., all reads and writes are coherent) and starvation avoidance, and a performance protocol.

The new Compaq AlphaServer GS320 [19] constitutes an example of cc-NUMA architecture specifically targeted at medium-scale multiprocessing (up to 64 processors). The hierarchical nature of its design and its limited scale make it feasible to use simple interconnects, such as a crossbar switch, to connect the handful of nodes, allowing a more efficient handling of certain cache misses than traditional directory-based multiprocessors by exploiting the extra ordering properties of the switch.

Prediction-based techniques

Prediction has a long history in computer architecture and it has proved useful for improving microprocessor performance. Prediction in the context of shared memory multiprocessors was first studied by Mukherjee and Hill, who showed that it is possible to use address-based[1] 2-level predictors at the directories and caches to track and predict coherence messages [46]. Subsequently, Lai and Falfasi modified these predictors to improve their accuracy and to reduce their size (that is, their implementation costs). Additionally, the authors presented the first design for a speculative coherent cc-NUMA using pattern-based predictors by executing coherence operations speculatively to hide the remote read latency [35]. In particular, the proposed scheme tries to predict when the producer of a particular memory line has finished writing to it. Then, the writable copy is speculatively invalidated and forwarded to the consumers. Finally, Kaxiras and Young [33] presented a taxonomy of all prediction schemes in a uniform space, and provided simulation results on the accuracy of a practical subset of them. Through this process, the authors derived prediction schemes that are more accurate than those previously proposed.

Alternatively, Kaxiras and Goodman [32] proposed and evaluated prediction-based optimizations of migratory sharing patterns (converting some load misses that are predicted to

[1] Address-based stands for predictors whose table is accessed using the effective memory address.

be followed by a store-write fault to coherent writes), wide sharing patterns (to be handled by scalable extensions to the SCI base protocol) and producer-consumer sharing patterns (presending a newly created value to the predicted consumers).

Bilir *et al.* [11] investigated a hybrid protocol that tried to achieve the performance of snooping protocols and the scalability of directory-based ones. The protocol is based on predicting which nodes must receive each coherence transaction. If the prediction hits, the protocol approximates the snooping behavior (although the directory must be accessed in order to verify the prediction). Performance results in terms of execution time were not reported and the design was based on a network with a completely ordered message delivery, in particular an Isotach-like Fat Tree Network [53], which could restrict its scalability.

Recently, cache-to-cache transfer misses has been reported as constituting a significant fraction of the cache miss rate in commercial applications [9]. For these misses, the home node has an obsolete copy of the memory line and the corresponding directory entry stores the identity of the node holding the single valid copy of the memory line (this node is said to be the *owner* node). Therefore, the role of the directory in these cases is to *redirect* the misses to the corresponding owner nodes (of course, the directory also serves as the serialization point for all accesses to a particular memory line and therefore, to ensure correctness). If on suffering a cache-to-cache transfer miss, the missing node were able to know the identity of the owner of the memory line, it could send the request directly to it, completely removing the access to the directory from the critical path of the miss. This idea is developed in [4] and requires to extend the original coherence protocol to deal with some emerging situations, as well as the development of an effective prediction scheme. Additionally, in [5] it is shown how a similar technique can be applied for accelerating upgrade misses, that is to say, write misses for which the requesting node already has the valid data and only needs exclusive ownership, for which invalidation messages are sent to the rest of the sharers of the memory line.

Another related technique to reduce coherence overhead resulting from the invalidation of shared lines was originally presented in [38]. In this work, some heuristics are applied to allow each processor to detect and *self-invalidate* those shared lines that will probably not be accessed in the future. Subsequently, Lai and Falsafi [36] proposed Last-Touch Predictors (LTPs) to improve self-invalidation accuracy.

4. CONCLUSIONS

Cache-coherent, nonuniform memory access or cc-NUMA is an attractive architecture for building a spectrum of shared memory multiprocessors (which are seeing widespread use in commercial, technical and scientific applications). Based on a directory-based cache coherence protocol, cc-NUMA designs offer a scalable performance path beyond symmetric multiprocessors (SMPs) by maintaining a compatible programming interface and allowing a large number of processors to share a single global address space over physically distributed memory.

Unfortunately, there are some factors which limit the maximum number of processors that can be offered at a good price/performance ratio. This paper surveys some of the proposals that have recently appeared focusing on two of these factors: the increased cost in terms of hardware overhead that the use of directories entails, and the long cache miss latencies observed in these designs as a consequence of the indirection introduced by the access to the directory.

Apart from the two issues studied in this paper, availability has become increasingly important as cache-coherent shared memory multiprocessors has seen widespread use in commercial, technical and scientific applications. Recently, techniques for providing fault-tolerance in these architectures has been the focus of several works, such as ReVive [52] and SafetyNet [56].

5. REFERENCES

[1] M. E. Acacio. "Improving the Performance and Scalability of Directory-based Shared-Memory Multiprocessors". *Ph.D. Thesis, University of Murcia*, 2003.

[2] M. E. Acacio, J. González, J. M. García and J. Duato. "A New Scalable Directory Architecture for Large-Scale Multiprocessors". *Proc. of the 7th Int'l Symposium on High Performance Computer Architecture (HPCA-7)*, pp. 97–106, January 2001.

[3] M. E. Acacio, J. González, J. M. García and J. Duato. "A Novel Approach to Reduce L2 Miss Latency in Shared-Memory Multiprocessors". *Proc. of the 16th Int'l Parallel and Distributed Processing Symposium (IPDPS'02)*, April 2002.

[4] M. E. Acacio, J. González, J. M. García and J. Duato. "Owner Prediction for Accelerating Cache-to-Cache Transfer Misses in cc-NUMA Multiprocessors". *Proc. of the Int'l SC2002 High Performance Networking and Computing*, November 2002.

[5] M. E. Acacio, J. González, J. M. García and J. Duato. "The Use of Prediction for Accelerating Upgrade Misses in cc-NUMA Multiprocessors". *Proc. of the 2002 Int'l Conference on Parallel Architectures and Compilation Techniques (PACT 2002)*, pp. 155–154, September 2002.

[6] A. Agarwal, R. Bianchini, D. Chaiken, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie and D. Yeung. "The MIT Alewife

Machine: Architecture and Performance". *Proc. of the 22nd Int'l Symposium on Computer Architecture (ISCA'95)*, pp. 2–13, May/June 1995.

[7] A. Agarwal, R. Simoni, J. Hennessy and M. Horowitz. "An Evaluation of Directory Schemes for Cache Coherence". *Proc. of the 15th Int'l Symposium on Computer Architecture (ISCA'88)*, pp. 280–289, May 1988.

[8] A. Ahmed, P. Conway, B. Hughes and F. Weber. "AMD Opteron^{TM} Shared Memory MP Systems". *Proc. of the 14th HotChips Symposium*, August 2002.

[9] L. A. Barroso, K. Gharachorloo and E. Bugnion. "Memory System Characterization of Commercial Workloads". *In Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA'98)*, pp. 3–14, June 1998.

[10] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets and B. Verghese. "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing". *Proc. of the 27th Int'l Symposium on Computer Architecture (ISCA'00)*, pp. 282–293, June 2000.

[11] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill and D. A. Wood. "Multicast Snooping: A New Coherence Method Using a Multicast Address Network". *Proc. of the 26th Int'l Symposium on Computer Architecture (ISCA'99)*, pp. 294–304, May 1999.

[12] L. Censier and P. Feautrier. "A New Solution to Coherence Problems in Multicache Systems". *IEEE Transactions on Computers*, 27(12):1112–1118, December 1978.

[13] D. Chaiken, J. Kubiatowicz and A. Agarwal. "LimitLESS Directories: A Scalable Cache Coherence Scheme". *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV)*, pp. 224–234, April 1991.

[14] Y. Chang and L. Bhuyan. "An Efficient Hybrid Cache Coherence Protocol for Shared Memory Multiprocessors". *IEEE Transactions on Computers*, pp. 352–360, March 1999.

[15] J. H. Choi and K. H. Park. "Segment Directory Enhancing the Limited Directory Cache Coherence Schemes". *Proc. of the 13th Int'l Parallel and Distributed Processing Symposium (IPDPS'99)*, pp. 258–267, April 1999.

[16] Convex Computer Corp. "Convex Exemplar Architecture". *dhw-014 edition*, November 1993.

[17] D. E. Culler, J. P. Singh and A. Gupta. *"Parallel Computer Architecture: A Hardware/Software Approach"*. Morgan Kaufmann Publishers, Inc., 1999.

[18] J. Duato, S. Yalamanchili and L. Ni. *"Interconnection Networks: An Engineering Approach"*. Morgan Kaufmann Publishers, Inc., 2002.

[19] K. Gharachorloo, M. Sharma, S. Steely and S. V. Doren. "Architecture and Design of AlphaServer GS320". *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pp. 13–24, November 2000.

[20] J. Goodman. "Using Cache Memories to Reduce Processor-Memory Traffic". *Proc. of the Int'l Symposium on Computer Architecture (ISCA'83)*, June 1983.

[21] A. Gupta and W.-D. Weber. "Cache Invalidation Patterns in Shared-Memory Multiprocessors". *IEEE Transactions on Computers*, 41(7):794–810, July 1992.

[22] A. Gupta, W.-D. Weber and T. Mowry. "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes". *Proc. Int'l Conference on Parallel Processing (ICPP'90)*, pp. 312–321, August 1990.

[23] D. Gustavson. "The Scalable Coherent Interface and Related Standards Projects". *IEEE Micro*, 12(1):10–22, Jan/Feb 1992.

[24] L. Gwennap. "Alpha 21364 to Ease Memory Bottleneck". *Microprocessor Report*, 12(14):12–15, October 1998.

[25] H. Hadimioglu, D. Kaeli and F. Lombardi. "Introduction to the Special Issue on High Performance Memory Systems". *IEEE Transactions on Computers*, 50(11):1103–1105, November 2001.

[26] L. Hammond, B. Hubbert, M. Siu, M. Prabhu, M. Chen and K. Olukotun. "The Stanford Hydra CMP". *IEEE Micro*, 20(2):71–84, March/April 2000.

[27] M. A. Heinrich. "The Performance and Scalability of Distributed Shared Memory Cache Coherence Protocols". *Ph.D. Thesis, Stanford University*, 1998.

[28] T. Hosomi, Y. Kanoh, M. Nakamura and T. Hirose. "A DSM Architecture for a Parallel Computer Cenju-4". *Proc. of the 6th Int'l Symposium on High Performance Computer Architecture (HPCA-6)*, pp. 287–298, January 2000.

[29] C. J. Hughes, V. S. Pai, P. Ranganathan and S. V. Adve. "RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors". *IEEE Computer*, 35(2):40–49, February 2002.

[30] R. Iyer and L. N. Bhuyan. "Switch Cache: A Framework for Improving the Remote Memory Access Latency of CC-NUMA Multiprocessors". *Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA-5)*, pp. 152–160, January 1999.

[31] R. Iyer, L. N. Bhuyan and A. Nanda. "Using Switch Directories to Speed Up Cache-to-Cache Transfers in CC-NUMA Multiprocessors". *Proc. of the 14th Int'l Parallel and Distributed Processing Symposium (IPDPS'00)*, pp. 721–728, May 2000.

[32] S. Kaxiras and J. R. Goodman. "Improving CC-NUMA Performance Using Instruction-Based Prediction". *Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA-5)*, pp. 161–170, January 1999.

[33] S. Kaxiras and C. Young. "Coherence Communication Prediction in Shared-Memory Multiprocessors". *Proc. of the 6th Int'l Symposium on High Performance Computer Architecture (HPCA-6)*, pp. 156–167, January 2000.

[34] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum and J. Hennessy. "The Stanford FLASH Multiprocessor". *Proc. of the 21st Int'l Symposium on Computer Architecture (ISCA'94)*, pp. 302–313, April 1994.

[35] A. C. Lai and B. Falsafi. "Memory Sharing Predictor: The Key to a Speculative DSM". *Proc. of the 26th Int'l Symposium on Computer Architecture (ISCA'99)*, pp. 172–183, May 1999.

[36] A. C. Lai and B. Falsafi. "Selective, Accurate, and Timely Self-Invalidation Using Last-Touch Prediction". *Proc. of the 27th Int'l Symposium on Computer Architecture (ISCA'00)*, pp. 139–148, May 2000.

[37] J. Laudon and D. Lenoski. "The SGI Origin: A ccNUMA Highly Scalable Server". *Proc. of the 24th Int'l Symposium on Computer Architecture (ISCA'97)*, pp. 241–251, June 1997.

[38] A. R. Lebeck and D. A. Wood. "Dynamic Self-Invalidation: Reducing Coherence Overhead in Shared-Memory Multiprocessors". *Proc. of the 22nd Int'l Symposium on Computer Architecture (ISCA'95)*, pp. 48–59, June 1995.

[39] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz and M. S. Lam. "The Stanford DASH Multiprocessor". *IEEE Computer*, 25(3):63–79, March 1992.

[40] T. Lovett and R. Clapp. "STiNG: A CC-NUMA Computer System for the Commercial Marketplace". *Proc. of the 23rd Int'l Symposium on Computer Architecture (ISCA'96)*, pp. 308–317, 1996.

[41] M. M. Martin, M. D. Hill and D. A. Wood. "Token Coherence: Decoupling Performance and Correctness". *Proc. of the 30th Int'l Symposium Computer Architecture (ISCA'03)*, June 2003.

[42] M. M. Martin, D. J. Sorin, A. Ailamaki, A. R. Alameldeen, R. M. Dickson, C. J. Mauer, K. E. Moore, M. Plakal, M. D. Hill and D. A. Wood. "Timestamp Snooping: An Approach for Extending SMPs". *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*, pp. 25–36, November 2000.

[43] M. M. Martin, D. J. Sorin, M. D. Hill and D. A. Wood. "Bandwidth Adaptive Snooping". *Proc. of the 8th Int'l Symposium on High Performance Computer Architecture (HPCA-8)*, pp. 251–262, February 2002.

[44] M. M. Michael and A. K. Nanda. "Design and Performance of Directory Caches for Scalable Shared Memory Multiprocessors". *Proc. of the 5th Int'l Symposium on High Performance Computer Architecture (HPCA-5)*, pp. 142–151, January 1999.

[45] S. S. Mukherjee and M. D. Hill. "An Evaluation of Directory Protocols for Medium-Scale Shared-Memory Multiprocessors". *Proc. of the 8th Int'l Conference on Supercomputing (ICS'94)*, pp. 64–74, July 1994.

[46] S. S. Mukherjee and M. D. Hill. "Using Prediction to Accelerate Coherence Protocols". *Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA'98)*, pp. 179–190, July 1998.

[47] A. K. Nanda, A.-T. Nguyen, M. M. Michael and D. J. Joseph. "High-Throughput Coherence Controllers". *Proc. of the 6th Int'l Symposium on High Performance Computer Architecture (HPCA-6)*, pp. 145–155, January 2000.

[48] A. K. Nanda, A.-T. Nguyen, M. M. Michael and D. J. Joseph. "High-Throughput Coherence Control and Hardware Messaging in Everest". *IBM Journal of Research and Development*, 45(2):229–244, March 2001.

[49] H. Nilsson and P. Stenström. "The Scalable Tree Protocol – A Cache Coherence Approach for Large-Scale Multiprocessors". *Proc. of 4th Int'l Symposium on Parallel and Distributed Processing (SPDP'92)*, pp. 498–506, December 1992.

[50] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, M. Parkin, W. Radke and S. Vishin. "The S3.mp Scalable Shared Memory Multiprocessor". *Proc. of the Int'l Conference on Parallel Processing (ICPP'95)*, pp. I:1–10, July 1995.

[51] B. O'Krafka and A. Newton. "An Empirical Evaluation of Two Memory-Efficient Directory Methods". *Proc. of the 17th Int'l Symposium on Computer Architecture (ISCA'90)*, pp. 138–147, May 1990.

[52] M. Prvulovic, Z. Zhang and J. Torrellas. "ReVive: Cost-Effective Architectural Support for Rollback Recovery in Shared-Memory Multiprocessors". *Proc. of the 29th Int'l Symposium Computer Architecture (ISCA'02)*, pp. 111–122, May 2002.

[53] P. F. Reynolds, C. Williams and R. R. Wagner. "Isotach Networks". *IEEE Transactions on Parallel and Distributed Systems*, 8(4):337–348, April 1997.

[54] R. Simoni. "Cache Coherence Directories for Scalable Multiprocessors". *Ph.D. Thesis, Stanford University*, 1992.

[55] R. Simoni and M. Horowitz. "Dynamic Pointer Allocation for Scalable Cache Coherence Directories". *Proc. Int'l Symposium on Shared Memory Multiprocessing*, pp. 72–81, April 1991.

[56] D. J. Sorin, M. M. Martin, M. D. Hill and D. A. Wood. "SafetyNet: Improving the Availability of Shared Memory Multiprocessors with Global Checkpoint/Recovery". *Proc. of the 29th Int'l Symposium Computer Architecture (ISCA'02)*, pp. 123–134, May 2002.

[57] J. Tendler, J. Dodson, J. Fields, H. Le and B. Sinharoy. "POWER4 System Microarchitecture". *IBM Journal of Research and Development*, 46(1):5–25, January 2002.

[58] The BlueGene/L Team. "An Overview of the BlueGene/L Supercomputer". *Proc. of the Int'l SC2002 High Performance Networking and Computing*, November 2002.

[59] J. Torrellas, L. Yang and A. T. Nguyen. "Toward A Cost-Effective DSM Organization that Exploits Processor-Memory Integration". *Proc. of the 6th Int'l Symposium on High Performance Computer Architecture (HPCA-6)*, pp. 15–25, January 2000.

[60] W.-D. Weber, S. Gold, P. Helland, T. Shimizu, T. Wicki and W. Wilcke. "The Mercury Interconnect Architecture: A Cost-Effective Infrastructure for High-Performance Servers". *Proc. of the 24th Int'l Symposium on Computer Architecture (ISCA'97)*, pp. 98–107, June 1997.