This article has been accepted for inclusion in a future issue.
IEEE Computer Architecture Letters Vol. 5, 2006

In-Network Cache Coherence

‡Noel Eisley, ‡Li-Shiuan Peh, and ⋆Li Shang
‡Department of Electrical Engineering, Princeton University, Princeton
⋆Department of Electrical and Computer Engineering, Queens University, Canada
{eisley,peh}@princeton.edu, li.shang@queensu.ca

Abstract— **We propose implementing cache coherence protocols within the network, demonstrating how an in-network implementation of the MSI directory-based protocol allows for in-transit optimizations of read and write delay. Our results show 15% and 24% savings on average in memory access latency for SPLASH-2 parallel benchmarks running on a 4x4 and a 16x16 multiprocessor respectively.**

Keywords— **cache coherence, interconnection network**

I. Introduction

With Moore's law furnishing chip designers with billions of transistors, architects are increasingly moving towards multi-core architectures as an effective way of dealing with escalating design complexity and power constraints. Commercial designs with moderate numbers of cores have been announced with shared memory architectures maintained with snoopy cache coherence protocols. In future generations, as the number of cores scales beyond tens, more scalable directory-based coherence protocols will be needed. However, there are well-known problems with the overhead of directory-based protocols: each access needs to first go to the directory node to discover where data is currently cached, or to uncover the sharers so they can be invalidated. These traversals to and from the directory node become increasingly costly as technology scales.

There have been a plethora of protocol optimizations proposed to alleviate the overheads of directory-based protocols (see Section IV). Specifically, there has been prior work exploring network optimizations for cache coherence protocols. However, to date, these protocols maintain a firm abstraction of the interconnection network fabric as a communication medium – the protocol consists of a series of end-to-end messages between requestor nodes, directory nodes and sharer nodes. In this paper, we propose removing this conventional abstraction of the network as solely a communication medium. Specifically, we propose implementing coherence protocols *within* the network, at each router node. This opens up the possibility of optimizing a protocol with in-transit actions.

Here, we target the classic MSI (Modified, Shared, Invalid) directory-based protocol [7] as a first illustration of how implementing the protocol within the network permits in-transit optimizations that were not otherwise possible. Figure 1(a) sketches a scenario where node B issues a read request to the home directory node H, which then proceeds to instruct a current sharer, node A, to forward its data to node B. It consists of three end-to-end messages, B to H, H to A, and A to B. Moving this protocol into the network allows node B to "bump" into node A while in-transit to the directory node H, realize that it is in the shared state and thus obtain the data directly from A, reducing the communication to just a round-trip between B and A. Figure 1(b) illustrates the potential corresponding savings for a new write request from node C. In the original MSI protocol, a write request message needs to go from C to H, followed by invalidations from H to A and B, corresponding acknowledgments from A and B to H, before the data can be forwarded to node C. An in-network implementation allows A and B to start percolating invalidations and the accompanying acknowledgments once C "bumps" into them enroute to H. This in-transit optimization reduces write communication from two round-trips to a single round-trip from C to H and back.

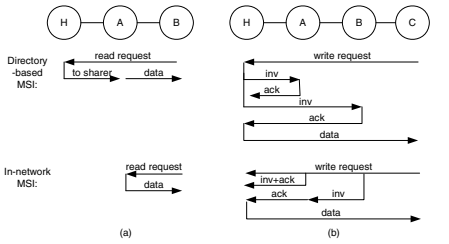

Fig. 1. Optimization of the MSI protocol through an in-network implementation for (a) reads and (b) writes.

In the rest of the paper, we will describe the in-network optimization of the MSI protocol in detail and discuss how we formally verify its sequential consistency in Section II. Next, in Section III, we present our preliminary simulation results showing 15% and 24% reduction in average memory access time when running a series of SPLASH-2 parallel benchmarks running on a 4x4 and a 16x16 multiprocessor respectively. Section IV discusses and contrasts against prior related work while Section V concludes the paper.

II. In-network cache coherence

A. MSI directory-based protocol

A simple, classic directory-based cache coherence protocol is the directory-based MSI protocol [7]. Each cache line is either *Invalid*, i.e. the local cache does not have a valid copy of this data; *Modified*, i.e. the local cache has the only copy of the cached data in the system and it is dirty; or *Shared*, i.e. the local cache contains a valid, read-only copy of the data, and furthermore other caches may also have a read-only copy. A *home* node that houses the directory is statically assigned to each memory address.

B. In-network MSI protocol (base)

Here, we outline the specifics of a base in-network implementation of the MSI protocol.

Virtual trees as directories. First, we propose the notion of virtual trees maintained in the network – one for

Manuscript submitted: 16 Nov. 2005. Manuscript accepted: 28 Feb. 2005. Final manuscript received: 17 Mar. 2006.

Posted to IEEE & CSDL on 3/22/2006
DOI 10.1109/L-CA.2006.9

each cache line. The virtual tree consists of one *root* node which is the node that first loads a cache line from off-chip memory, all nodes that are currently sharing this line, as well as the intermediate nodes between the root and the sharers. The home node that used to house the directory no longer stores the state of the cache line nor the sharers; instead, it points in the direction of the root node of the virtual tree. The nodes of the tree are connected by virtual links with each link between two nodes always pointing towards the *root* node.

Read accesses. When a node reads a cache line that is not locally cached, it sends a read request message. If the requesting node is not part of the virtual tree associated with the read address, the read request message is simply routed towards the static home node, just as it was originally done in the MSI protocol. Should the message encounter a node that is a part of the virtual tree enroute, it detours and starts following the virtual links towards the root node instead, i.e. each router directs the message along the corresponding physical link towards the root.

A read request message terminates when it encounters a node that has valid data cached for the address contained in the message, or when it reaches the home node. If it reaches a node containing valid data, a read reply message is generated and sent back towards the original requester node, along with the data. This *in-transit* optimization of read latency is the result of our proposed in-network implementation. If the read request message reaches the home node, then one of two things may happen. First, if the home node has an outgoing link pointing towards the root node, then the message is forwarded in that direction, in the same way as if it had intercepted the tree at any other point, as above. Second, if the home node does not have any outgoing links, then that means that no cache in the network contains a valid copy of the data, and therefore it must be loaded from off-chip memory. The data is then read, placed in a read reply message and returned towards the original requester.

When a read reply message routes back to the original requesting node, it makes the following decision at each hop along the way. If there exists a virtual link which connects to a node that is one hop closer to the requester node, it routes along the corresponding physical link. If, however, there is no such virtual link, then the message *constructs* a virtual link towards the requesting node, pointing in the direction of the root node. The message is then routed along the physical link corresponding to the newly created virtual link.

Write accesses. A write begins similarly to a read in that a write request message is sent towards the root node if the requestor node is part of the virtual tree. Otherwise, it is routed towards the home node. As in the original MSI protocol, all write requests are handled by the home node, i.e. the home node arbitrates between multiple write requests and ensures sequential consistency by delaying the updating of a cache line until all invalidations have been sent and acknowledged.

In the simplest case, there are no virtual links for this address at the home node, and the home node does not hold a valid copy of the cache line, i.e., no virtual tree exists. In this case, the home node sends a write reply message back to the original requesting node, granting permission to modify the data. Similarly to a read reply message, a write reply message constructs virtual links as it routes towards the requesting node that now becomes the root node. Once the write request reaches the original requesting node, the cache line is written and the dirty bit is set. As in the original MSI protocol, if a read request or teardown message comes by later and the dirty bit is set, the cache line is written back to memory.

If, however, a virtual tree exists upon arrival of the write request, then the tree will first be torn down or invalidated. Here, the in-network implementation again enables in-transit optimization of invalidation delay. Enroute to the home node, the first virtual tree node that the write request message encounters will result in spawning of teardown messages along all virtual links off that node. These teardown messages recursively propagate through the tree until they reach the leaves of the tree. At a leaf, a teardown message is turned into an acknowledgment message and sent back up the tree towards the home node. An acknowledgement message removes the virtual link connecting it and its parent as it moves up the tree.

In this way, when the home node has no more virtual links, it knows that the entire virtual tree has been successfully torn down, and it is safe to send out a write reply message to the original requesting node. From this point everything proceeds as discussed above for the case where no tree was found to exist for the given address.

B.1 Implementation

The only difference in the router microarchitecture that implements the proposed protocol versus a typical interconnection network router is the addition of the *virtual tree cache*. This virtual tree cache is accessed by the first flit of a message (head flit) to determine its route, or the output port this message should request for. It is accessed in the same way as the regular data cache with the address contained in each message's header – if the tag matches, and there is a hit in the tree cache, its prescribed direction is used as the desired output port; else, the default routing algorithm determines the desired output port. The virtual tree cache can be accessed in parallel with the routing pipeline stage and sized appropriately so the router pipeline delay remains unchanged. For subsequent body and tail flits, there is no change to the router pipeline; they follow the route that is reserved by the head flit, linked via the virtual channel ID stored in each flit header.

Each entry of the virtual tree cache consists of 8 bits (in addition to the tag bits which depends on its size and associativity) for a 2-dimensional network: a virtual link field with 1 bit per north, south, east, west (NSEW) direction (4 bits); two bits to describe which link leads to the owner (2 bits); a busy bit (1); and an outstanding request bit (1).

The virtual link bit field has a bit set for each physical link which is also a virtual link for the given address. Since

a node can by definition have only one virtual link leading *to* the root node, we only need two bits to encode which link it is. The busy and outstanding request bits are both used to maintain sequential consistency and not specific to our proposed in-network MSI protocol. The busy bit only applies to the home node; when set, it means that the home node has sent out a teardown message and is waiting for the return of the acknowledgments; this is used so that request messages which arrive during the teardown of a tree will queue up at the home node. The outstanding request bit is used to ensure atomic (per address) memory operations.

In the event of a capacity miss in the virtual tree cache, the network needs to evict an existing entry as well as other virtual links of that address in the tree, i.e. it needs to tear down the tree of which the victim entry is a member. Once the existing tree has been removed, the construction of the new, conflicting, tree can proceed.

Formal verification of sequential consistency. We use Murϕ [4], a model checking tool, to verify the sequential consistency of our proposed in-network protocol. Due to the exhaustive nature of the search, Murϕ confines its applicability to finite-state machines. Using Murϕ, we identified and verified the micro operations of data access, i.e., read/write, and corresponding node/network operations. To each memory address, multiple concurrent reads and up to two concurrent writes are allowed. Murϕ verified that our in-network protocol is sequentially consistent.

III. Simulation Results

We implemented a trace-driven architectural network simulator, driven by memory traces gathered by running a set of SPLASH-2 benchmarks [12] on Bochs [9], a multiprocessor simulator with embedded Linux 2.4 kernel. Each benchmark was run in Bochs [9] and the memory trace captured once, using sixteen threads. The trace for each benchmark is then applied to 4-by-4, 8-by-8, and 16-by-16 systems (all 2D meshes). The sixteen threads are placed randomly onto 16 nodes in the upper-left corner of the system, mimicking the program being allocated a subset of the system when multiprogrammed. For the 4-by-4 system, the home node is in the middle, and for the larger systems, the home node is in the lower-right corner; i.e. in both cases, the directory is sited in our favor. As the simulator does not model network contention, having a single home node does not create congestion delay. The results assume infinite data and virtual tree caches, i.e. no evictions. This lowers the potential savings of our approach since it reduces the number of cache accesses. Average memory access latency is measured in terms of hop count. No off-chip latency is included since that will affect both the original and in-network MSI protocols similarly.

Effect on average memory latency. The reduction in average read and write access latencies for six SPLASH-2 [12] benchmarks with the proposed in-network implementation are presented in Figure 2. We see that for a 4-by-4 system, average read latency is reduced by an average of 21.5% across all benchmarks. For a 16-by-16 system, this enlarges to an average of 30.2% lower read latency. The same trend holds for average write latency, though lower

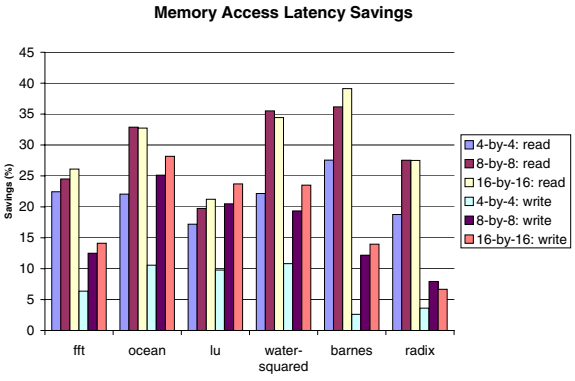

Fig. 2. Reduction in read and write access latency for our protocol as compared to typical directory-based MSI protocol.

savings is realized: an average of 7.4% reduction in write latency for 4-by-4, and 18.4% for 16-by-16. As the results show, a larger system yields higher savings in memory access latency since the directory node is further away, presenting greater potential for in-transit savings. The example in Figure 1 explains why the potential savings in write latency is lower: a read can potentially return immediately once a sharer is encountered with a best-case latency of 2 hops, regardless of the system size; a write always needs to go to the home node, with a best-case latency of a round-trip traversal between the requestor and home node vs. two round-trips, a 50% savings.

Storage overhead. There are two key differences in our in-network implementation of the MSI protocol that affect the storage overhead. First, in our in-network implementation, the home node no longer stores a list of sharers, but only points to the root node. Now, though, each intermediate node needs storage for the virtual tree cache bits. Note however that the in-network implementation enables the storage of *directions to sharers* rather than *actual sharer addresses*, with each node only knowing about itself and its immediate neighbors. As a result, the virtual tree cache line size grows with the *dimension* of the system (the number of immediate neighbors), rather than the total number of nodes.

To calculate the storage overhead of our protocol, we multiply the number of bits per virtual tree cache line by the average number of active virtual tree entries, as tracked by the simulator. We then calculate the storage overhead of the MSI protocol (the sharer bit vector plus the busy and outstanding request bits). In the interest of space, we do not report per-benchmark results, but only the averages here. For the 4-by-4 system, our protocol uses 16% more storage; For the 8-by-8, this increases to 40% more overhead. However, for the 16-by-16 system, our protocol uses 22% *less* storage space. This highlights the poor storage scalability of the directory-based protocol whose sharer bit vector scales with the size of the system.

IV. Related Work

It has been shown in [1] that it is expensive to enforce sequential consistency, as it requires ensuring a strict ordering between read/write accesses that mandates multiple round-trips of request-invalidate-acknowledgment-reply communications. As a result, extensive prior research has focused on optimizing sequentially consistent systems.

Network optimizations. Prior work has demonstrated the impact of the interconnection network on overall shared-memory system performance and proposed network designs that can better the performance of cache coherence protocols. Stets *et al.* [13] demonstrated that network features, such as network ordering, multicasting, can greatly simplify coherence protocol implementation and improve system performance. In [5], Fillo *et al.* presented a crossbar-based cluster interconnect that provides virtual shared memory supported by internodal address space mapping. In [3], Dai and Panda presented block correlated FIFO channels to tackle the memory ordering issue of network interface design. These protocols still use the network solely for communications though.

The closest work to our proposed in-network cache coherence protocol is that by Kaxiras and Goodman [8], where the benefits of a network-mapped protocol was first proposed. They proposed a tree that is similar to the virtual trees of our proposed protocol – the tree connects sharers of a line and is mapped onto the network topology, so that a sharer who is closer by than the directory can be reached to reduce latency. However, they stop short of embedding the protocol fully into the network, instead implementing the protocol at network interface cards which constantly snoop traffic and redirect traffic when needed by sinking network packets and regenerating new ones with new destinations. The underlying network thus remains purely for communications. By moving the implementation of cache coherence protocols into the network routers, we can realize very-low-latency steering of the coherence traffic. In addition, by moving the directories into the network routers, information needs no longer be kept on a per-node basis, but can be kept on a per-direction basis, significantly improving storage scalability.

Protocol optimizations. Substantial work has gone into optimizing the deficiencies of the vanilla MSI protocol through new protocol states, with protocols supporting relaxed consistency the most prevalent [1]. While relaxing the consistency model lowers communication overhead, it also complicates the programming model. Our proposed in-network implementation of the MSI protocol reduces communication overhead while ensuring sequential consistency. Besides, end-to-end protocol optimizations are orthogonal to in-transit optimizations; they can be further optimized through an in-network implementation with the proposed in-network virtual trees. For instance, our virtual trees can be used to percolate tokens of the TokenB coherence protocol [10] to minimize communication overhead. Various coherence protocols which use pointers to maintain abstract linked lists (such as SCI [6]) or trees (such as [11]) can also be moved into the network, with the virtual links of our virtual trees forming the hardware pointers between the sharers. For instance, in [11], each node in the tree holds pointers to its parent and children. The tree is optimally balanced such that the distance from the root to any leaf is bounded from above by $log_2(N)$, where N is the number of nodes in the network. However, a parent-child relationship in the tree does not necessarily map to a neighboring tile relationship in the physical network. Therefore, the number of actual hops required for a message to traverse the tree from the root to any of the leaves is bounded from above by $Dlog_2(N)$, where D is the diameter of the network. By implementing these pointers within the network, using the virtual links of our routers, physical network locality can be leveraged in an efficient manner since the pointers are at each router, rather than at the data caches of the nodes themselves.

V. Conclusions

This paper serves as a proof-of-concept of the potential benefits of in-network cache coherence. Next, we will explore the implementation of more sophisticated protocols, such as relaxed consistency models, within the network, leveraging the natural locality exposed by the network to further optimize their overhead towards the ideal. Ultimately, We see in-network cache coherence's virtual trees providing a scalable, distributed solution for capturing data affinity in parallel applications, enabling efficient run-time affinity-aware thread placement [2].

Acknowledgments

We would like to thank Kevin Ko for his help in generating the trace files which we used in this work; Margaret Martonosi and the rest of the Network Driven Processor (NDP) group at Princeton, David Wood and the anonymous reviewers of the paper for their helpful feedback towards this paper and future work. This work was partially funded by the MARCO Gigascale Systems Research Center and NSF EHS-0509402.

References

[1] S.V. Adve and K. Gharachorloo, "Shared Memory Consistency Models: A Tutorial," *IEEE computer*, vol. 29, no. 12, pp. 66-76, 1996.

[2] J. Chen *et al.*, "Hardware-Modulated Parallelism in Chip Multiprocessors," in *Proc. of Workshop on Design, Arch., and Simulation of CMPs, held in conjunction with MICRO*, November, 2005.

[3] D. Dai and D.K. Panda, "Exploiting the Benefits of Multiple-Path Network in DSM Systems: Architectural Alternatives and Performance Evaluation," *IEEE Trans. Comput.*, vol. 48, no. 2, pp. 236-244, 1999.

[4] D.L. Dill, "The Murϕ Verification System." in *CAV*, 1996, pp. 390-393.

[5] M. Fillo and R.B. Gillett, "Architecture and Implementation of Memory Channel 2," *DEC Technical Journal*, vol. 9, no. 1, 1997.

[6] S. Gjessing, *et al.*, "The SCI Cache Coherence Protocol," Kluwer Academic Publishers, 1992.

[7] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc., 2003.

[8] S. Kaxiras and J.R. Goodman, "The GLOW Cache Coherence Protocol Extensions for Widely Shared Data," in *ICS '96: Proceedings of the 10th International Conference on Supercomputing*. New York, NY, USA: ACM Press, 1996, pp. 35-43.

[9] K.P. Lawton, "Bochs: A Portable PC Emulator for Unix/x," *Linux J.*, vol. 1996, no. 29, p. 7, 1996.

[10] M.M.K. Martin, M.D. Hill, and D.A. Wood, "Token Coherence: Decoupling Performance and Correctness," in *Proc. Int. Symp. Computer Architecture*, Jun. 2003.

[11] H. Nilsson and P. Stenström, "The Scalable Tree Protocol - A Cache Coherence Approach for Large-Scale Multiprocessors," in *Proc. of the Fourth IEEE Symp. on Par. and Dist. Proc.* IEEE Computer Society Press, 1992, pp. 498-506.

[12] http://www.flash.stanford.edu/apps/SPLASH/

[13] R. Stets *et al.*, "The Effect of Network Total Order, Broadcast, and Remote-Write Capability on Network-Based Shared Memory Computing," in *Proc. Int. Symp. High Perf. Comp. Arch.*, Feb. 2000.