Formal Verification and its Impact on the Snooping versus Directory Protocol Debate

Milo M. K. Martin
Department of Computer and Information Science
University of Pennsylvania
milom@cis.upenn.edu

Abstract

This invited paper argues that to facilitate formal verification, multiprocessor systems should (1) decouple enforcing coherence from enforcing a memory consistency model and (2) decouple the interconnection network from the cache coherence protocol (by not relying on any specific interconnect ordering or synchronicity properties). Of the two dominant classes of cache coherence protocols—directory protocols and snooping protocols—these two desirable properties favor use of directory protocols over snooping protocols. Although the conceptual simplicity of snooping protocols is seductive, aggressive implementations of snooping protocols lack these decoupling properties, making them perhaps more difficult in practice to reason about, verify, and implement correctly. Conversely, directory protocols may seem more complicated, but they are more amenable to these decoupling properties, which simplify protocol design and verification. Finally, this paper describes the recently-proposed token coherence protocol's adherence to these properties and discusses some of its implications for future multiprocessor systems.

1 Introduction

After years as an exotic and exclusively high-end technology, shared-memory multiprocessors are now solidly mainstream. Even low-end servers routinely have at least two processors, and high-end systems often have dozens of processors. With the performance and power advantages of multi-core chips, multiprocessors are expanding beyond servers into desktops, laptops, handhelds, game consoles, and embedded devices. If current trends continue, multi-core systems will become so ubiquitous that in a few years it may not be possible to purchase a uniprocessor system.

Cache coherence protocols. These shared-memory multiprocessor systems use a cache coherence protocol to coordinate the caches and memories as part of providing the processors with a consistent view of the contents of a single shared address space. The exact definition of this "consistent view" of memory is defined by the system's *memory consistency model* [2], which is commonly specified as part of the instruction set architecture of the system. As part of enforcing a consistency model, invalidation-based cache coherence protocols conceptually maintain a global invariant: for each block of shared memory either (1) zero or more processors are allowed to read the block or (2) exactly one processor is allowed to write and read the block.

Snooping versus directory protocol debate. Although multiprocessors are now common, there is not a clear consensus on the design of cache coherence protocols. Whereas other areas of computer architecture have slowly gravitated to canonical designs (*e.g.*, processor microarchitecture), multiprocessor system designs are still widely varied. In fact, a debate on designing such systems has raged for decades. Much of the debate has centered around two classes of cache coherence protocols: snooping protocols and directory protocols. Snooping protocols use a totally-ordered interconnect (*e.g.*, a bus) to broadcast request to all processors, allowing all system components to transition between coherence states in a consistent fashion. In contrast, directory protocols send all requests to a directory at the memory, which forwards requests to other processors as needed. Directory protocols avoid broadcast and a totally-ordered interconnect in exchange for adding indirection latency to some misses.

The debate revisited. This invited paper contains thoughts and opinions on this debate based on experience with developing and enhancing protocols in an academic research environment, limited industrial experience, conversations with system designers, and dabbling in formal verification of such protocols. Instead of focusing exclusively on the performance or scalability of cache coherence protocols, this paper revisits the snooping versus directory protocol debate by explicitly considering the impact of formal verification on design complexity and design verification of multiprocessor systems. This paper describes the impact of formal verification on the cache coherence protocol design process (Section 2), distills two desirable decoupling properties for formal modeling and implementing coherence protocols (Section 3), revisits the snooping versus directory debate in this context (Section 4), and reflects on the creation of *token coherence* [16, 18, 19], a recently-proposed alternative approach to cache coherence (Section 5). This paper concludes that formal verification

Proceedings of the 2005 International Conference on Computer Design (ICCD'05)
0-7695-2451-6/05 $20.00 © 2005 IEEE

considerations favor choosing either a directory protocol or a token coherence protocol over all but the simplest implementations of snooping-based cache coherence protocols.

2 Impact of Formal Verification

In addition to simulation and other forms of traditional design verification, formal verification plays as increasingly important role in the design of today's digital systems. Formal verification includes techniques such as model checking via state space exploration and automated theorem proving techniques. Using formal verification methods has become almost routine for enhancing confidence (and finding bugs) in cache coherence protocols [1, 5, 6, 8, 12, 25, 26].

2.1 Explicit Role of Formal Verification

The explicit role of formal verification is finding bugs and improving confidence in the correctness of a design. These methods can either be applied after the high-level design has been completed or during the high-level design phase.

Post-design verification. If formal verification is thought of as part of traditional design verification efforts (*e.g.*, simulation-based techniques), formal verification will likely be applied later in the design and implementation of a system. Although late application of formal verification is still effective at finding bugs, it suffers from first finding many "false bugs" caused by incomplete documentation, design specification errors, and real bugs that were already fixed by the designers [12]. Bugs that are encountered late may also be more difficult to fix cleanly or the fix may have a significant impact on the performance or complexity of the design.

During-design verification. In contrast, if formal verification is thought of as an integral part of the design process, formal verification will likely be applied early in the design process and have a more pervasive impact on the actual design. Formal verification is most effective when used during the initial design phases [8, 12]. Instead of relying on incomplete specifications, the modeling processes can be part of the specification process (or the model may even be part of the actual specification). Bugs found early can be fixed more easily and at the appropriate level of the design. For example, instead of adding a localized timing-sensitive kludge to fix a subtle coherence protocol race, early detection of such a bug could have resulted in a higher-level design change that would have been simpler, cleaner, and easier. In addition, a set of bugs discovered early may actually cause the designers to rethink or adjust the high-level design.

2.2 Implicit Role of Formal Verification

If verification is an integral part of the high-level design process, formal verification has several implicit benefits—benefits that arise not from the actual computational verification of the model, but from the creation of the model itself. Creating a model requires one or more engineers to think systematically about the correctness of the design and the accuracy of its specification.[1] In addition, the designers themselves may be influenced by the knowledge that whatever the designers propose, someone will need to model it.[2] This subtle effect can influence the designers to create a more modular design with clearer abstraction layers and better specification and documentation. Knowing that a design is going to be formally verified may encourage designers to actually document their designs and employ "principles of good design" that they otherwise might not be disciplined enough to self-enforce. In essence, placing the extra design constraint of "verifiability" on the designers in some cases may produce a better design.

3 Two Desirable Properties of Cache Coherence Protocols

This section describes two properties found in some coherence protocols that give them advantages in terms of tractability of formal verification, design complexity, and ability to be modified for future design iterations.

3.1 Decouple Coherence from Consistency

The definition of correctness of a multiprocessor memory system is its memory consistency model [2]. Informally, *coherence* refers only to the behavior of a single memory location, whereas the *memory consistency model* encompasses the ordering and interaction of memory operations to multiple memory locations. Because individually verifying either a consistency model or simple coherence properties is a difficult task, we advocate clearly decoupling these two aspects of a multiprocessor memory system to simplify verification. To enable this decoupling, the *coherence protocol* should provide a coherence interface sufficient to provide a serializable view of memory. The *processor core* should interact with that interface to provide whatever memory consistency model the processor design team

[1] For example, when talking with a verification engineer about one specific verification effort, I was told that the only critical bug found during the formal verification process was encountered during *construction* of the model; the automated checking of the model did verify the bug and its fix, but it did not discover any other significant bugs.

[2] For example, after suggesting a specific cache coherence protocol proposal to one industrial designer, the designer's face went pale, and he said something like "the verification team would never let us get away with that".

deems necessary (using prefetching and various speculative techniques to aggressively implement the consistency model [3, 9]).

In such a system, the only role of the coherence protocol is to inform the processor when it can read a block and when it can write a block. This notion closely corresponds with the multiple-reader-single-writer coherence invariant described in the introduction. Such a coherence protocol (1) provides a simple interface to the processor and (2) allows the processor to aggressively implement the desired consistency model. To clarify these points, consider the implementation of memory barrier operations (used to establish memory ordering). A system that follows the above decoupling property can efficiently handle memory barriers entirely within the processor core, freeing the coherence protocol from providing special-purpose operations with complicated semantics.

Although some systems have at least partially adopted a decoupled consistency and coherence philosophy (*e.g.*, [4, 7, 13]), many systems have not (*e.g.*, [10, 27]). Those systems intertwine coherence and consistency throughout the system, often relying on a patchwork of special ordering properties of processor queues and a total ordering of requests via a totally-ordered interconnect to carefully and delicately orchestrate the desired consistency model (discussed further below).

3.2 Decouple Interconnect from Protocol

The interconnect is the communication mechanism that reliably delivers messages between the caches and memories of a multiprocessor system. A cache coherence protocol might rely on an interconnect that provides (1) point-to-point ordering, (2) totally-ordered message delivery, (3) no special ordering properties. We contend that to ease verification, the coherence protocol should not depend upon any special ordering properties of the interconnect. To support this position, this section discusses some of the difficulties of designing and verifying coherence protocols that rely on these two types of interconnect orderings.

Avoid point-to-point ordered interconnects. Of the two types of interconnect ordering properties, point-to-point ordering is less problematic than totally-ordered interconnects. However, point-to-point ordering is undesirable because (1) it constrains interconnect design (*e.g.*, by precluding or limiting adaptive routing) and (2) it complicates formal verification by encumbering the model and increasing the state space by reducing symmetry (as compared with modeling an unordered interconnect) [12]. Fortunately, many protocols—especially directory protocols—do not rely on any sort of interconnect ordering (*e.g.*, [24]). However, some protocols require at least point-to-point ordering (*e.g.*, [28]), and many protocols require the even more onerous totally-ordered interconnect (as discussed next).

Avoid totally-ordered interconnects. Relying on a totally-ordered interconnect has pervasive design ramifications. Depending on such an interconnect is undesirable because it intertwines the processing of different addresses. An interconnect provides a *total ordering of messages* if all messages are delivered to all destinations in some order. A total ordering requires an ordering among all the messages (even those from different sources or sent to different destinations). For example, if any processor receives message *A* before message *B*, then no processor receives message *B* before *A*. Many protocols (*e.g.*, most snooping protocols and some directory protocols [10]) require an interconnect that provides a total ordering of requests. Unfortunately, establishing a total ordering of requests can add complexity, increase cost, and increase latency. For example, totally-ordered interconnects commonly use some centralized root switch or arbitration mechanism, and such mechanisms are not a good match for direct interconnects.

In protocols that do not rely on a total ordering of requests—*e.g.*, traditional directory protocols [14, 24, 28] and AMD's Opteron protocol [4]—requests and responses for different blocks have no need to unduly influence each other. Only when a processor decides to commit a read or write to a block does the processor need to consider the interactions of reads and writes to other blocks. In contrast, in systems that rely on a total ordering of requests, requests (and sometimes responses) for different addresses must be kept in order with respect to each other throughout the system.[3] This requirement forces cache controllers to process all requests in order, making banking of coherence controllers to increase bandwidth difficult. To use a uniprocessor analogy, relying on a total ordering of requests introduces "dependences" (both true and false dependencies) between requests for various blocks.

3.3 Discussion and Implications

The decoupling of coherence and consistency and the decoupling of protocol and interconnect results in a much more modular design. Such a design has simpler interfaces between the various system components, which should substantially simplify verification (both formal verification and traditional verification). For example, each component can be designed and verified in

[3] For further discussion of the subtle implications and interactions of memory consistency models and totally-ordered interconnects, see chapters 2 and 10 of Martin [16].

isolation, and a high-level model can be used to verify many of their interactions.

In addition, such a design is more amenable to incremental improvements for future product generations based on the same basic design. Because of the huge effort to fully design, verify, and performance debug a high-performance design, a design will often be used for several product generations. For example, the initial "P6" core design of the Pentium Pro has been improved and adapted over many years to be used in desktop chips (Pentium II and Pentium III), server chips (Xeon), and most recently in low-power mobile chips (Pentium M). Similar derivations occur with multiprocessor systems. A multiprocessor that adheres to these two decoupling properties will be easier to adapt over time, because an individual component (*e.g.*, the interconnect or cache controller) can be improved with fewer global interactions than in a less modular design.

As an example of non-localized cascading changes, consider a first-generation design with an interconnect that does not implement adaptive routing (and thus implicitly provided point-to-point interconnect ordering); if the coherence protocol exploits that point-to-point ordering, the second-generation system cannot implement adaptive routing in the interconnect unless the coherence protocol is also redesigned. However, if the system designers had decoupled the interconnect ordering from the protocol, such a change to the interconnect would be a purely local change.

Although these two decoupling properties seem like laudable and desirable properties, they also may be controversial, as some proposals have explicitly argued the opposite approach [10], and systems such as IBM's Power4 [27] and AlphaServer GS320 [10] designs have neither property. Both of these systems use an ordered interconnect and intermingle coherence and consistency. For example, Power4 relies on ordering in the interconnect and requires broadcasting of certain memory-ordering barrier operations. In contrast, the Alpha 21364/GS1280 and AMD Opteron protocols both appear to exhibit both decoupling properties described above (based on the limited published descriptions of the protocols [4, 7, 24]).

4 Revisiting Snooping vs. Directory Protocols

The choice of coherence protocol is as subtle and controversial today as it has ever been. Instead of focusing primarily on performance or scalability, this section revisits the snooping versus directory protocol debate by considering the impact of formal verification and the two desirable decoupling properties described in the last section. Although any high-performance implementation of a multiprocessor memory system is complicated, we contend that directory protocols are preferable from a complexity and formal verification viewpoint, because directory protocols are more amenable to decoupling coherence from consistency and decoupling the interconnect from the coherence protocol.

Snooping protocols. Although snooping is seductively simple from a conceptual viewpoint, as snooping implementations have evolved over time they have become anything but simple. Real-world high-performance implementations are not simple because they use many advanced techniques such as snoop response combining, split-transaction protocols, split request/response interconnects, and multiple totally-ordered switched interconnects. These enhancements tightly couple the timing and ordering of the interconnect with the protocol, and they couple coherence and consistency. In essence, aggressive snooping protocols are complex and difficult to verify because they do not exhibit the two desirable decoupling properties described in the last section.

Directory protocols. In contrast, even the simplest directory protocol may seem complicated (*e.g.*, because of the number of request and writeback races that can occur). However, these situations are exactly the sort of high-level protocol issues that formal verification methods can help overcome, especially if applied early in the design. Directory protocols are more amenable to formal verification techniques because they often exhibit these two decoupling properties. These properties also confer such protocols better "complexity scalability" over time. That is, as a directory protocol implementation becomes more aggressive—faster interconnects, more outstanding requests, more protocol states and optimizations—the changes are more localized.

Why then have directory protocols not been used more frequently? One possible explanation is that directory protocols were promoted—both in academic research and in industrial designs—primarily for being scalable (*i.e.*, support hundreds or thousands of processors).[4] In fact, the term "scalable cache coherence" is synonymous with a directory protocol. Such emphasis on scalability and their first-glance complexity may have given directory protocols a reputation as an exotic and expensive technology. Perhaps this reputation has led designers to be more comfortable evolving their existing snooping-based system over the years, especially as most systems sold—even of "scalable" systems—have a modest number of processors [23]. Transitioning to a directory protocol represents a more radical design change

[4] See Hill [11] for an early skeptical view of scalability.

than just evolving an existing snooping system.[5] In addition, many early directory systems were hierarchical multiprocessors built from snooping-based multiprocessor building blocks (*e.g.*, [14, 15, 28]), resulting in a reputation of substantial complexity and large latency overhead from bridging between the two protocols. Finally, performance issues may have played a role (discussed briefly in the next section).

5 A New Alternative: Token Coherence

Although we encourage designers to consider directory protocols over snooping protocols for complexity and design verification reasons, directory protocols have an important performance disadvantage that must be addressed: directory protocols add indirection latency to cache-to-cache misses. To resolve races, a directory protocol sends all requests to a home node that then forwards the request (if needed) or responds with the data from memory. In contrast, a snooping protocol avoids indirection by broadcasting all requests to all nodes and relying on interconnect ordering to help resolve races. Although the additional indirection latency of a directory protocol can be partially mitigated by using a directory cache, an extra interconnect traversal remains on the critical path of some cache misses.

The recently-proposed token coherence protocol [16, 18, 19] can eliminate the constraint of directory indirection without sacrificing either decoupling of the the interconnect from the coherence protocol or decoupling of coherence from consistency. Token coherence uses token counting to resolve races without requiring a home node or an ordered interconnect. Token coherence embraces even further levels of decoupling by separating the *correctness substrate* from the system's *performance policy*. The correctness substrate is further decoupled into *enforcing safety* and *avoiding starvation*.

Enforcing safety. Token coherence's correctness substrate provides safety by counting tokens for each block of memory in the system. Each block in the system has a fixed number of tokens (T). If a processor's cache has all T tokens for the block, it is allowed to read and write the block. If a processor's cache has at least one token, it can read the block (but not write it). If a processor's cache holds no tokens, it can neither read nor write the block. These token counting rules directly ensure that while one processor is writing the block, no other processor is reading or writing it. In essence, it directly enforces the multiple-reader-single-writer coherence invariant suitable for allowing the processor to enforce the desired memory consistency model. Such simple rules allow for reasoning about protocol safety in a much simpler fashion, and by its nature, token coherence does not rely upon complicated ordering properties of the interconnect or the use of a directory home node to resolve races.

Avoiding starvation. Although token counting ensures safety, it does not ensure that a request is eventually satisfied. Thus the correctness substrate provides persistent requests to prevent starvation. When a processor detects possible starvation (such as via a time-out), it initiates a persistent request. The substrate then activates at most one persistent request per block, using a fair arbitration mechanism. Each system node remembers all activated persistent requests (for example, in a table at each node) and forwards all tokens for the block—those tokens currently present and received in the future—to the request initiator. Finally, when the initiator has sufficient tokens, it performs a memory operation (a load or store instruction) and deactivates its persistent request.

Performance policies. The correctness substrate provides a foundation for implementing many performance policies. These performance policies focus on making the system fast and bandwidth-efficient, but have no correctness responsibilities, because the substrate is responsible for correctness. This decoupling of responsibility between the correctness substrate and performance policy enables the development of performance policies that capture many of the desirable attributes of snooping and directory protocols. For example, token coherence performance policies have been developed [16] to approximate an unordered broadcast-based protocol (inspired by snooping protocols), a bandwidth-efficient performance policy that emulates a directory protocol, and a predictive hybrid protocol that uses destination-set prediction [17].

Ramifications on design verification. As token coherence's use of decoupling goes beyond the two decoupling properties described in Section 3, it perhaps has some additional verification advantages. For example, Marty *et al.* [22] used a single-level token coherence protocol to approximate the performance characteristics of a two-level hierarchical coherence protocol. That is, the protocol is *flat for correctness* but *hierarchical for performance*. Using token coherence in this fashion allows for the performance benefits of a hierarchical protocol combined with the ease of verification of a single-level protocol. Marty *et al.* [22] also show that the difficulty of verifying the token coherence correctness substrate is comparable to verifying a single-level directory protocol. In addition, the flexibility provided

[5] In fact, some of my earlier research took this evolutionary approach to trying to enhance snooping protocols to mitigate their disadvantages (*e.g.*, [20, 21]). However, I now believe the disadvantages of directory protocols are perhaps more easily mitigated than the disadvantages of snooping protocols.

by the performance policy should allow a system using a token coherence protocol to be enhanced over time without substantial changes to the correctness substrate.

6 Conclusions

In this paper, we reflected on the impact of the increasing use of formal design verification methods during the early stages of multiprocessor system design. We identified two desirable decoupling properties for reducing the complexity and enhancing the verifiability of the system: decoupling coherence from consistency and decoupling the interconnect ordering properties from the cache coherence protocol. We used these properties to revisit the snooping versus directory protocol debate, and we argued that these decoupling properties point to directory protocols as being more attractive that snooping protocols from a verifiability and modifyability point of view. Finally, we identified token coherence as a possible approach for (1) further simplifying the verifiability of coherence protocols and (2) overcoming the indirection performance penalty found in directory protocols. As multi-core designs are becoming ubiquitous, we encourage the designers of these systems to look beyond simple bus-based snooping-based designs and to consider directory protocols and token coherence as approaches to creating more verifiable and adaptable designs.

Acknowledgments

The author thanks Rajeev Alur, Sebastian Burckhardt, Jesse Bingham, Mark Hill, Alan Hu, and David Wood for helpful discussions leading up to this paper. This work is funded in part by a gift from Intel Corporation.

References

[1] D. Abts, D. J. Lilja, and S. Scott. Toward Complexity-Effective Verification: A Case Study of the Cray SV2 Cache Coherence Protocol. In *1st Workshop on Complexity-Effective Design held in conjunction with the 27th International Symposium on Computer Architecture*, June 2000.

[2] S. V. Adve and K. Gharachorloo. Shared Memory Consistency Models: A Tutorial. *IEEE Computer*, 29(12): 66–76, Dec. 1996.

[3] S. V. Adve, V. S. Pai, and P. Ranganathan. Recent Advances in Memory Consistency Models for Hardware Shared Memory Systems. *Proceedings of the IEEE*, 87 (3):445–455, Mar. 1999.

[4] A. Ahmed, P. Conway, B. Hughes, and F. Weber. AMD Opteron Shared Memory MP Systems. In *Proceedings of the 14th HotChips Symposium*, Aug. 2002.

[5] H. Akhiani, D. Doligez, P. Harter, L. Lamport, J. Scheid, M. Tuttle, and Y. Yu. Cache Coherence Verification with TLA+. In *FM'99—Formal Methods, Volume II*, volume 1709 of *Lecture Notes in Computer Science*, page 1871. Springer Verlag, 1999.

[6] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, Dec. 1996.

[7] Z. Cvetanovic. Performance analysis of the Alpha 21364-based HP GS1280 multiprocessor. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 218–229, June 2003.

[8] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol Verification as a Hardware Design Aid. In *1992 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525, 1992.

[9] K. Gharachorloo, A. Gupta, and J. Hennessy. Two Techniques to Enhance the Performance of Memory Consistency Models. In *Proceedings of the International Conference on Parallel Processing*, volume I, pages 355–364, Aug. 1991.

[10] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren. Architecture and Design of AlphaServer GS320. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 13–24, Nov. 2000.

[11] M. D. Hill. What is Scalability? *Computer Architecture News*, 18(4):18–21, 1990.

[12] A. J. Hu, M. Fujita, and C. Wilson. Formal Verification of the HAL S1 System Cache Coherence Protocol. In *Proceedings of the International Conference on Computer Design*, pages 438–444, Oct. 1997.

[13] J. Laudon and D. Lenoski. The SGI Origin: A cc-NUMA Highly Scalable Server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241–251, June 1997.

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

[15] T. D. Lovett and R. M. Clapp. STiNG: A CC-NUMA Computer System for the Commercial Marketplace. In *Proceedings of the 23th Annual International Symposium on Computer Architecture*, May 1996.

[16] M. M. K. Martin. *Token Coherence*. PhD thesis, University of Wisconsin, 2003.

[17] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood. Using Destination-Set Prediction to Improve the Latency/Bandwidth Tradeoff in Shared Memory Multiprocessors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 206–217, June 2003.

[18] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: A New Framework for Shared-Memory Multiprocessors. *IEEE Micro*, November-December 2003.

[19] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 182–193, June 2003.

[20] M. M. K. 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. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 25–36, Nov. 2000.

[21] M. M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood. Bandwidth Adaptive Snooping. In *Proceedings of the Eighth IEEE Symposium on High-Performance Computer Architecture*, pages 251–262, Feb. 2002.

[22] 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 *Proceedings of the 11th IEEE Symposium on High-Performance Computer Architecture*, Feb. 2005.

[23] J. R. Mashey. NUMAflex Modular Design Approach: A Revolution in Evolution. Posted on comp.arch news group, Aug. 2000.

[24] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. In *Proceedings of the 9th Hot Interconnects Symposium*, Aug. 2001.

[25] F. Pong, M. Browne, A. Nowatzyk, and M. Dubois. Design Verification of the S3.mp Cache-Coherent Shared-Memory System. *IEEE Transactions on Computers*, 47 (1):135–140, Jan. 1998.

[26] F. Pong and M. Dubois. Verification Techniques for Cache Coherence Protocols. *ACM Computing Surveys*, 29(1):82–126, Mar. 1997.

[27] J. M. Tendler, S. Dodson, S. Fields, H. Le, and B. Sinharoy. POWER4 System Microarchitecture. *IBM Journal of Research and Development*, 46(1), 2002.

[28] 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. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.