Chapter 22
Evaluation of Cache Coherence Mechanisms for Multicore Processors

Malik Al-Manasia and Zenon Chaczko

Abstract Multiple core designs have become commonplace in the processor marketplace, and are therefore a major focus in modern computer architecture research. Thus, for both product development and research, multiple core processor performance evaluation is a mandatory step in marketplace. Multicore computing has presented many challenges for system designers; one of which is data consistency between a shared cache or memory and the local caches of the chip. This is also known as cache coherency. The cache coherence mechanisms are a key component in the direction of accomplishing the goal of continuing exponential performance growth through widespread thread-level parallelism. In the scope of this research, we have studied the available efficient methods and protocols used to achieve cache coherence in multicore architectures. These protocols were further modeled and evaluated utilizing Simics simulator for multicore architectures. We also explored the weaknesses and strengths of different protocols and discussed the way of improving them.

22.1 Introduction

Enhancement of microprocessors is guided greatly by Moores Law, which forecasts that the number of transistors per silicon area is doubled every eighteen months [38]. Computer architects are embarking on a fundamental shift in how the transistor bounty is used to increase performance, while Moores Law is expected to continue at least into the next decade [36].

There is a great correlation between power and frequency. When the frequency is enhanced, the power will also be enhanced and after that the temperatures will also increase [41]. Multicore processors take advantage of this relationship by combining

M. Al-Manasia (✉) · Z. Chaczko
Faculty of Engineering, University of Technology Sydney, Broadway,
NSW 2007, Australia
e-mail: malik.a.al-Manasia@student.uts.edu.au

Z. Chaczko
e-mail: zenon@eng.uts.edu.au

G. Borowik et al. (eds.), *Computational Intelligence and Efficiency in Engineering Systems*, Studies in Computational Intelligence 595,
DOI 10.1007/978-3-319-15720-7_22

multiple cores. Each core is able to run at a lower frequency, by splitting up the power provided to a single core normally between all cores [18]. The performance will enhance whereas the power and temperatures are still under control.

Numerous new applications are becoming multithreaded and computer architecture is turning its focus towards parallelism. This is because it's very hard to enhance the performance of single core processors by increasing the clock frequencies, let alone the difficulties probable such as heating or speed of light if the frequency exceeds certain ranges, the design and verification needs a large team as well.

Multicore design is faced with many challenges like any new design. These challenges require identification and understanding. One of the main challenges facing Chip Multiprocessors (CMPs) is the competition for shared resources, this challenge forms a restriction bottleneck [11, 23, 24]. Some of the shared resources are: main memory bandwidth and capacity, cache bandwidth and capacity, memory subsystem interconnection bandwidth and system power. Memory system scalability is the big issue the multicore future will face and challenge.

The first on-chip multiprocessor for the computing market of the general purpose was presented in 2000 by IBM. Followed in 2005 by AMD that presented the two processor version for the server market, also in the end of 2007 and 2008 kickoff quad-core and triple-core processors were introduced, whereas an 8-core chip for computer-farm application was produced by Sun in 2006. Tilera however, on the 20th of August 2007 gave off its 64-core processor. Intel also released its two-processor versions in 2005 for the server market; its quad core processor was presented on Dec 13, 2006. Within the coming years Intel is expected to release its 80-core processor prototype each running at 3.16 GHz.

Cache memory is defined as a specific memory subsystem whereas data of persistent usage is stored for fast access. Caching is used to increase the speed of large amount of slow, affordable memory by using a small amount of expensive memory. Regarding multicore processors, cache coherency stands for the credibility of data stored in each cores cache. Multicore processors may have distributed and shared caches on the chip, so we should account for coherence protocols to assure that when a core reads from memory, it reads the current piece of data and not a value that has been updated by a different core.

The cache coherence problem is illustrated in Fig. 22.1. Figure 22.1 illustrates a shared L2 cache by four cores with private caches via a bus. The cores access location X sequentially. At the beginning core 1, brings a copy to its cache by reading X from L2 cache. After that, core 4 brings a copy to its cache by reading X from L2 cache. Thereafter core 4 changes X's location value from 8 to 5. With a write-through cache, causing the L2 cache location to be updated, but in (action 4) when core 1 read location X again, it will read the old value 8 from its own cache rather than reading its correct value 5 from L2 cache.

The writeback caches complicate the situation even further. Core 4's write would not update L2 cache right away; instead it would barely set the dirty (or modified) bit concerned with the cache block holding location X. Contents of cache block would be written back to L2 cache solely when this cache block is subsequently replaced from the cache of core 4. The reading of the old value is not inclusive to core 1;

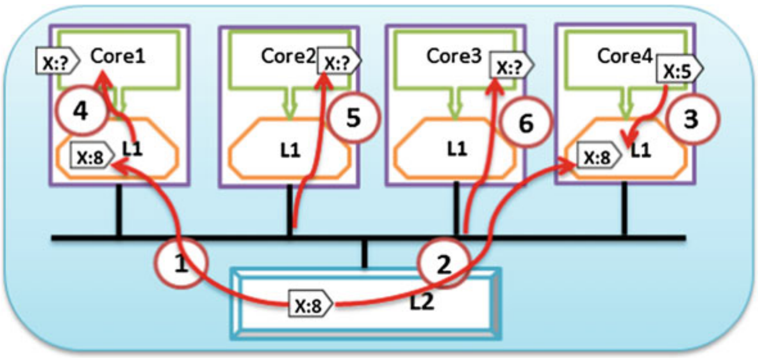

Fig. 22.1 Cache coherence problem

furthermore core 2 and core 3 will miss in their caches when reading location X (actions 5 and 6) by reading the old value 8 instead of 5 from L2 cache. Last but not least, if more than one core write different values in their write-back caches to location X, The end value that will reach the L2 cache will not be related to the sequence in which the writes to X occurred instead it will be determined by the sequence in which the cache blocks that contain X are replaced.

Coherence between the caches has to be enforced in order for correct execution. This process is affected by two main factors: Performance and implementation cost.

In general in hardware-based, there are two methods for cache coherence, a snooping protocol and a directory-based protocol. The Snoopy cache-coherence methods require sending information to all of cache controllers. However, if the number of cores increased the cache messages will increase, also then the required bus which connect the caches and all messages pass through it bandwidth will be bigger than available one, then total saturation of bus bandwidth occurs. These techniques can be used in small-scale systems due to this limitation. However, the Directory-based protocol, scales to larger numbers of processors or cores than snoopy-based coherence protocol, since it enables multiple coherence actions to take place at the same time.

There are three critical attributes that have an impact on the performance of any cache coherence protocol, thus being:

Low-latency Cache-to-Cache Misses: A cache-to-cache miss is a miss frequently caused by accessing shared data that requires another processors cache to provide the data. To decrease the latency of cache-to-cache misses, a coherence protocol should ideally support direct cache-to-cache misses [33]. To efficiently support the frequent communication and synchronization in these workloads, systems are required to optimize the latency of cache-to-cache misses [6].

Bandwidth Efficiency: A cache coherence protocol should conserve bandwidth to decrease the cost and avoid interconnect contention (since contention reduces performance).

Scalability Challenges: Hardware resources must scale efficiently, as the future is multithreaded. Memory hierarchy is the main hardware scalability dilemma. CMPs depend on large, multi-level cache hierarchies to diminish the high cost, limited bandwidth and high latency of main memory. Caches generally use nearly the half of chip area and a substantial fraction of system energy. The main problems limit cache hierarchies scalability are cache associativity, partitioning and coherency. Caches currently spend significant energy and latency to implement associative lookups, making them inefficient and have a considerable impact on system performance. The previously proposed partitioning techniques suffer from two main weaknesses: they are not scalable and limited to a small number of coarse-grain partitions, and partitioning often reduces performance. Also, traditional coherence schemes do not scale beyond a few tens of cores.

The objective of this paper is to explain the relative behavior of different coherence protocols (traditional protocols and Token Coherence based protocols). We aim not to (1) Generate ultimate execution times or throughput rates for our simulated systems or (2) Evaluating such protocols on all of the future's system configurations. We use an approximation of a chip multiprocessor system in order to accurately obtain relative comparisons and evaluations instead. Full system simulation and modeling of the first-order timing effects for approximating an aggressive multicore system operating commercial loads are the means to reach such an aim. Our goal is to get the first-order effects, although similar to most architectural simulations—Capturing all system's aspects in precise detail is not what we try to do.

22.2 Background and Related Work

In this section, we describe the traditional protocols (Snooping based protocols and Directory based protocols) in addition to the Token coherence protocol: how they are working, what the advantages and disadvantages of each of them and how the improvement of these protocols can be done.

22.2.1 Snooping Protocols

In Snooping protocols the processor cores snoop every bus transaction and respond with appropriate state changes for the corresponding cache lines depending on two elements: the cache line status and bus transaction type. Two primary policies classify the snoop-based coherence protocols: the invalidation-based protocols (e.g., the write-once [21], the Synapse [16], the Berkeley [25] and the Illinois [13]) and the update-based protocols (e.g., the Firefly [45] and the Dragon [37]). Hence invalidation-based coherence has been favored over update-based coherence protocols in most up-to-date systems (e.g., [9, 12, 13, 22, 29, 39, 44]), this paper considers only invalidation-based cache coherence protocols.

The main present advantage of snoop-based multiprocessors is the low average miss latency, especially for cache-to-cache misses. The responder knows fast that it has to send a response because a request is sent directly to all the other processors and memory modules in the system. Low cache-to-cache miss latency is of great importance for workloads with considerable amounts of data sharing. Replying with data from processor caches when possible can reduce the average miss latency if cache-to-cache misses have lower latency than fetching data from memory (i.e., a memory-to-cache miss). Low-latency memory access is a result of the tightly-coupled nature of these systems [33].

Formerly, snooping has had two additional advantages. First, shared-wire buses used to be cost-effective interconnects for numerous systems and bus-based coherence offered a complexity-effective approach to applying cache coherence. Second, bus-based snooping protocols were comparatively simple. This advantage that was of great importance in the past is now much less importance; New snooping protocols that use virtual buses are often as complex or more complex than alternative approaches to coherence.

The first primary disadvantage of snooping is that even though system designers have evolved beyond shared-wire buses snooping designers are still bound to choosing interconnects that can provide virtual-bus behavior (i.e., a total order of requests) when they choose interconnect. These virtual-bus interconnects could be more expensive (e.g., by requiring switch chips), may obtain lower bandwidth (e.g., due to a bottleneck at the root), or might acquire higher latency (since all requests need to reach the root). On the contrary, an unordered interconnect (such as a directly connected grid or torus of processors) might have more attractive latency, bandwidth and cost attributes [33].

The second primary disadvantage is that snooping protocols are still naturally broadcast-based protocols; i.e., protocols whose bandwidth requirements increase with the number of processors. This broadcast requirement limits system scalability even after eliminating the bottleneck of a shared-wire bus or virtual bus. To control this limitation, recent proposals [8, 34, 42] aim at reducing the bandwidth requirements of snooping by using destination-set prediction (also known as predictive multicast) instead of broadcasting all requests. These proposals suffer from snoopings other disadvantage: They rely on a totally-ordered interconnect, Even though they reduce request traffic.

Some of the best techniques that can be used to improve the Snoopy Cache Coherency protocols:

1. Three wired OR signals: In this technique, when any other cache has a copy of block besides the requester the first signal is asserted, and when any cache has exclusive copy of block the second signal given. The third signal is asserted when all snoop actions are finished on the bus [18]. When the third signal is asserted, the other two signals are safely examined by the requesting L1 and the L2. Performance can be improved by implementing these signals using low-latency L-Wires since all of them are on the critical path.
2. Voting wires is another technique used to enhance snoopy based protocol with low latencies. Generally cache to cache transfers occur from the data in the modified state, whereas there is a single supplier [17]. Although, a block can be retrieved from other cache rather than memory in MESI protocol. Multiple caches share copy voting mechanism is generally used to provide data therefore voting mechanism works with low latencies and enhances processor performance.

22.2.1.1 Directory-Based Protocols

Memory is distributed among different processors in directory based protocols and directory is maintained for each such memory. Currently, several Chip Multiprocessors, as Piranha [7] also use directory protocols in order to maintain cache coherency. L1 cache misses are sent to L2 caches and a directory which store the status of block is maintained across each L2 caches. The request goes to home node where the original data is stored to check whether it has. When request comes from requester node from another cache, if it is not available the request goes to remote node by home node and first fetches data from remote node and sends it later on to requester node. Also write-invalidate-direct based protocol is employed in one of the most common chip multiprocessors technology we are using, which is core2duo.

Directory protocols target the avoidance of the scalability and interconnection limitations of snooping protocols. Directory protocols predate snooping protocols for a fact, with Censier and Feautrier [10] and Tang [43] performing early work on directory protocols in the late 1970s. Systems that use these protocols also known as distributed shared memory (DSM) or cache-coherent non-uniform memory access (CC-NUMA) systems are preferred when scalability (in the number of processors or cores) is a first-order design constraint. These protocols often sacrifice fast cache-to-cache misses in exchange for this scalability, even though these protocols are significantly more scalable than snooping protocols, Examples of systems that use directory protocols include Stanfords DASH [27, 28] and FLASH [26], MITs Alewife [2], SGIs Origin [29], the AlphaServer GS320 [20] and GS1280 [15], Sequents NUMA-Q [14], Crays X1 [1], and Piranha [7].

Directory protocols better scalability than snooping protocols and avoidance of snoopings virtual bus interconnect are the two primary advantages of directory protocols. The most discussed and studied advantage is perhaps the significantly improved scalability of directory protocols. By only contacting those processors that might have copies of a cache block (or a small number of additional processors when using an approximate directory implementation), the traffic in the system grows linearly with the number of processors. In contrast, the endpoint traffic of broadcasts used in snooping protocols grows quadratically [33]. Combined with a scalable interconnect (one whose bandwidth grows linearly with the number of processors), Using directory protocol the system is permitted to scale to hundreds or thousands of processors.

Two scalability dilemmas are encountered when using large system sizes:

First, the amount of directory state required becomes great concern.

Second, interconnect of reasonable is not truly scalable.

A deep study of these two problems have been applied extensively, and actual systems supporting hundreds of processors exist (e.g., the SGI Origin 2000 [29]).

The ability to exploit arbitrary point-to-point interconnects is the second and maybe the more important advantage of directory protocols. The point-to-point interconnects are generally have high-bandwidth and low-latency [33].

Directory protocols are of two main disadvantages. First, the extra interconnect traversal and directory access is on the critical path of cache-to-cache misses. Hence the memory lookup is normally performed simultaneously with the directory lookup memory-to-cache, misses do not incur a penalty. Directory lookup latency is similar to that of main memory DRAM in numerous systems, and thus locating this lookup on the critical path of cache-to-cache misses increases cache-to-cache miss latency considerably. Although the directory latency can be decreased by using fast SRAM in order to hold or cache directory information, the extra latency presented by the additional interconnect traversal is harder to mitigate. A combination of these two latencies enhances cache-to-cache miss latency significantly. With the prevalence of cache-to-cache misses in many important commercial workloads, these higher-latency cache-to-cache misses might have a dramatical impact on system performance.

The storage and manipulation of directory state could be considered the second disadvantage of directory protocols. This disadvantage was present on earlier systems, ones that used dedicated directory storage (SRAM or DRAM) thus adding to the total system cost. On the other hand, to save directory state while eliminating additional storage capacity overhead; numerous modern directory protocols have used the main system DRAM and reinterpretation of bits used for error correction codes (ECC) (e.g., the S3mp [40], Alpha 21364 [39], UltraSparc III [22], and Piranha [7, 20]) by increasing the number of memory reads and writes [19] storing these bits in main memory enhances the memory traffic [33].

Some of the best techniques that can be used to improve the Directory based Cache Coherency protocols:

1. Exclusive Read Request for a block in a shared state.
The L2 cache is a clean copy and upon receiving a request from the L1 cache the L2 cache is going to invalidate every L1 cache in this approach. Before sending the data to the processor, The L1 cache will receive an invalidate acknowledgement from the other L1 caches. Normally the L2 cache requires a hop for the reply and where as it requires 2 hops for an acknowledgment. So the latencies of the Hop and the latencys of reply and acknowledgement messages should be of the same value [4]. In this approach both the acknowledgment and reply messages are sent at the same time via the corresponding low latency L-wires and low power PW-wires. This approach enhances the performance and lowers the consumption of power.

2. Read request for block in exclusive state.
The exclusive owner will receive a read request for the L2 cache sends as the requesting L1 receives a copy of data from the L2 cache. The exclusive owner will send a reply message to the requesting L1 cache pointing out that the data sent by the L2 cache is valid if the exclusive copy is a clean one. If the requesting node requests for data the exclusive owner will send a copy of data to the requesting L1 cache while updating the data in the L2 cache. The requesting cache will not go on until it receives a message from the exclusive owner. Simultaneously the data the L2 cache sends the data through slow PW wires [4].

An acknowledgement message should be sent to the requester from the exclusive owner through low bandwidth L-wires if the owner copy is a clean copy, whereas if the owner copy is a dirty copy then the message should be sent through the B-wires and the write back to the L2 is done via PW-wires. This approach mainly follows the ways of improving the performance by sending the prioritized data through the L-wires and the least prioritized one through PW-wires.

3. The methods of which the Proximity Aware coherence protocols enhances the performance of a multicore thus by lowering the unnecessary access to the off-chip memory are described below.

22.2.1.2 Token-Based Protocols

The constraint of directory indirection are removed without sacrificing either decoupling of the interconnect from the coherence protocol or decoupling of coherence from consistency thus is done by using the recently-proposed token coherence protocol [31, 33, 34]. Token coherence take on token counting to resolve races without the need to require home node or an ordered interconnect. Token coherence contains even further levels of decoupling by separating the correctness substrate from the systems performance policy. The correctness substrate is decoupled further on invoking safety and avoiding starvation.

Token counting does not ensure that a request is satisfied in the end even though it ensures safety. Thus, the correctness substrate gives persistent requests to prevent starvation. The processor initiates a persistent request when it detects possible starvation. Using a fair arbitration mechanism, the substrate then activates at most one persistent request per block. 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 are present at the time being and received in the future to the request initiator. Finally, the initiator performs a memory operation (a load or store instruction) and deactivates its persistent request when it has the necessary tokens.

Token coherence performance policies have been developed [33] 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 [34].

TokenB protocol focuses on both avoiding indirection latency for cache-to-cache misses (like snooping protocols) and not requiring any interconnect ordering (like directory protocols). One seemingly clear approach is to directly send broadcasts on an unordered interconnect.

22.3 Evaluation Methodology

The objective of evaluation is to explain the relative behavior of different coherence protocols (traditional protocols and Token Coherence based protocols). The Simics full-system multiprocessor simulator (WindRiver) extended with the Wisconsin GEMS simulation environment [35] is used in order to perform the analysis [5].

Table 22.1 Simulation parameters

Private L1 caches	64 KB, 4-way set associative, split D/I, 1 ns latency (2-cycle latency)
L2 cache	Unified, 4 MB, 6 ns latency (12 cycles)
Main memory	4 GB, 80 ns (160 cycles)
Coherence protocol	MOESI protocol
Interconnect link	4 GB/second or unbounded bandwidth 15 ns latency (30 cycles)

Throughout this research, we demonstrate how the overall performance of different protocols by the runtime i.e. measuring the time necessary to complete certain amount of work. The metric instructions-per-cycle has been used by other works instead of runtime in judging performance improvements. However; system timing effects of multiprocessor workloads may alter the number of instructions executed therefore, Instruction Per Cycle (IPC) is not a suitable metric for evaluating the coherence protocols and systems.

The measurement is started at the parallel phase so that we avoid measuring thread forking. Until now, a full system checkpoint (to provide a well-defined starting point) is used to initialize the system state and to simulate the execution until the end of the parallel phase. The number of cycles is recorded and referred to as application time in order to complete the parallel phase.

Endpoint traffic (in messages per miss) and interconnect traffic (in terms of bytes on interconnect links per miss) are other ways besides reporting runtime that measure and report the traffic [4]. The endpoint traffic shows the amount of controller bandwidth needed for handling incoming messages. The amount of link bandwidth used by the messages are indicated in the interconnect traffic as they traverse the interconnection.

We used three multi-threaded commercial workloads from the Wisconsin Commercial Workload Suite [3]: an online transaction processing workload (OLTP), a Java middleware workload (SPECjbb), and a static web serving workload (Apache). The previously mentioned workloads operate on a simulated 16-core SPARC processor that runs Solaris 9. The simulated system has 4 GBs of main memory.

Following are details of how the main components of the system are modeled (Table 22.1).

22.4 Analysis and Evaluations

In order to evaluate the demand system; full-system simulation is used. Using full system simulation allows for evaluating the proposed systems when running realistic scientific applications on top of actual operating systems. As well as capturing the subtle timing effect that can't be captured with trace-based evaluation. We use cycles per transaction to be our only metric of performance. Figure 22.2 shows normalized runtime (smaller is better) that TokenB is faster than Snooping with unlimited bandwidth links by (21–34).

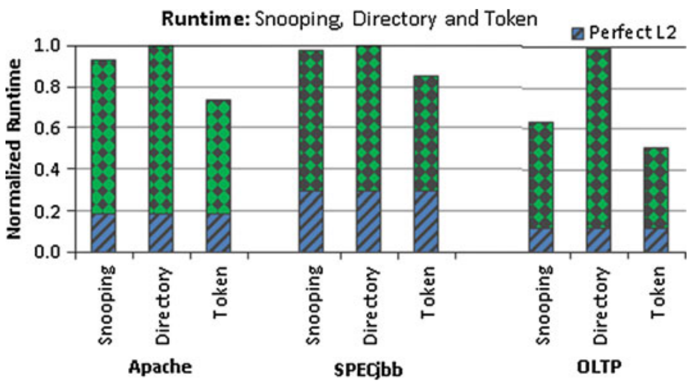

Fig. 22.2 Runtime of Snooping, Directory, and TokenB. The runtime of Snooping, Directory, and TokenB with unbounded link bandwidth

TokenBs endpoint traffic is similar to or less than Snooping while has more interconnect traffic than Snooping. Figure 22.3 illustrates the endpoint traffic (in normalized messages per miss received at each endpoint coherence controller), and Fig. 22.4 exhibits the interconnect traffic (in normalized bytes per miss). When examining only data and non-reissued request traffic, TokenB and Snooping are practically identical. TokenB adds some additional traffic overhead (comes from reissued and persistent requests), but the overhead is small for all three of our workloads. Snooping and TokenB both use extra traffic for writeback control messages, but because of the detailed implementation decisions in Snooping involving writeback acknowledgment messages, Snooping uses more traffic for writebacks than TokenB. Snooping sends a writeback request on the ordered interconnect to both the memory and to itself as a marker message. If it is still the owner of the block, it receives the marker message and sends the data back to the memory. Ignoring this precise implementation overhead leads us to conclude that these protocols generate the same amounts of traffic.

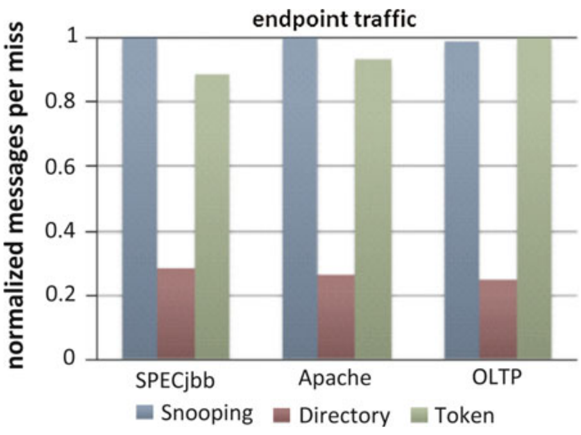

Fig. 22.3 Endpoint Traffic of Snooping, Directory, and TokenB. The endpoint traffic (in normalized messages per miss) of Snooping, Directory, and TokenB

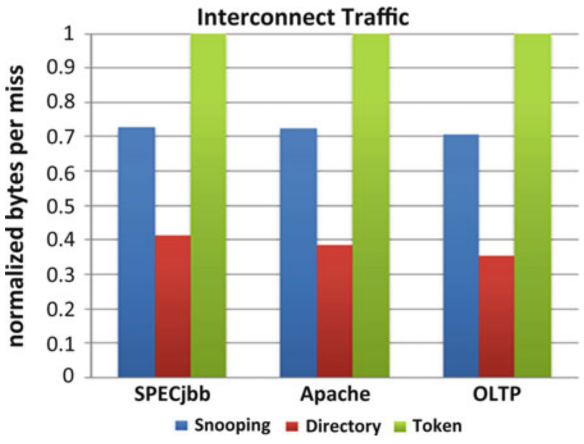

Fig. 22.4 Interconnect Traffic of Snooping, Directory, and TokenB. The interconnect traffic (in normalized bytes per miss) of Snooping, Directory, and TokenB

TokenB generates bigger endpoint traffic and interconnect traffic than Directory. These figures show that TokenB generates more traffic than Directory about three times more endpoint traffic and interconnect traffic). Thus results in, incredibly higher bandwidth coherence controllers are required by TokenB.

TokenB depends on broadcast, which limits its scalability. TokenB is less scalable than Directory, hence DIRECTORY avoids broadcast. However, as the number of processors increases, TokenB endpoint bandwidth improves linearly. The interconnect traffic difference between TokenB and Directory enhances slowly. Thus, TokenB can operate well for almost up to 64 processors if bandwidth is rich (by using high-bandwidth links and high-throughput coherence controllers). On the other hand, TokenB is not a good choice for larger or more bandwidth-limited systems.

22.5 Conclusions and Recommendations for Future Work

The cache coherence mechanisms are a key element aiming at accomplishing the goal of proceeding exponential performance growth through widespread thread-level parallelism. The available efficient methods and protocols were studied in this paper, which were used to achieve cache coherent in multicore architectures. These protocols (Snooping-based protocols, Directory-based protocols and TokenB-based protocols) modeled and evaluated on the Simics/GEMS simulator. The weaknesses and strengths of each protocol were demonstrated and we discussed how the improvement of them can be done.

TokenB is both (1) better than Snooping and (2) faster than Directory when bandwidth is plentiful. TokenB is better than SNOOPING because it uses the same amounts of traffic and can outperform Snooping by exploiting a faster, unordered interconnect. TokenB is of a higher speed than Directory in bandwidth-rich situations by avoiding placing directory lookup latency and a third interconnect traversal on the critical path of common cache-to-cache misses. On the other hand, TokenB uses a moderate amount of additional interconnect traffic and considerable more endpoint message bandwidth than Directory for small systems. Thus, Directory performs better than TokenB in a bandwidth-constrained situation. Although TokenB is a message-intensive protocol, it is only one of many possible high performance policies.

The choice of coherence protocol is as complicated and subtle today as it has ever been. CMPs will enable even more cost-effective multicore processors by reducing the number of discrete components in the system. For the future work, we recommend further modeling for each of the snooping, Directory, and Token protocols by testing additional benchmarks rather than the three benchmarks that we use in this paper, in the same architecture for more accuracy results.

References

1. Abts, D., Scott, S., Lilja, D.J.: So many states, so little time: verifying memory coherence in the Cray X1. In: IEEE Proceedings of International Parallel and Distributed Processing Symposium, p. 10(2003)
2. Agarwal, A., Bianchini, R., Chaiken, D., Johnson, K.L., Kranz, D., Kubiatowicz, J., Lim, B.-H., Mackenzie, K., Yeung, D.: The MIT Alewife machine: architecture and performance. In: IEEE Proceedings of 22nd Annual International Symposium on Computer Architecture, pp. 2–13 (1995)
3. Alameldeen, A.R., Martin, M.M., Mauer, C.J., Moore, K.E., Xu, M., Hill, M.D., Wood, D.A., Sorin, D.J.: Simulating a 2M Commercial Server on a 2K PC. Computer **36**(2), 50–57 (2003)
4. Al-Manasia, M., Al-Omari, F., Al-Jarrah, M.: Modeling and evaluation of cache coherence mechanisms for multicore processors. Masters Thesis, Yarmouk University (2011). http://repository.yu.edu.jo/handle/123456789/1505
5. Al-Manasia, M., Chaczko, Z.: A survey of computer system architecture simulators, case study: sniper. In: Proceedings of the 2nd Asia-Pacific Conference on Computer-Aided System Engineering, APCASE 2014, 10–12 February 2014, South Kuta, Indonesia, pp. 14–15 (2014). ISBN: 978-0-9924518-0-6
6. Barroso, L.A., Gharachorloo, K., Bugnion, E.: Memory system characterization of commercial workloads. In: ACM SIGARCH Computer Architecture News **26**(3), 3–14 (1998)
7. Barroso, L.A., Gharachorloo, K., McNamara, R., Nowatzyk, A., Qadeer, S., Sano, B., Smith, S., Stets, R., Verghese, B.: Piranha: a scalable architecture based on single-chip multiprocessing. In: ACM SIGARCH Computer Architecture News **28**(2), 93–282 (2000)
8. Bilir, E.E., Dickson, R.M., Hu, Y., Plakal, M., Sorin, D.J., Hill, M.D., Wood, D.A.: Multicast snooping: a new coherence method using a multicast address network. In: IEEE Proceedings of the 26th International Symposium on Computer Architecture, pp. 294–304 (1999)
9. Borkenhagen, J.M., Hoover, R.D., Valk, K.M.: EXA cache/scalability controllers. In: IBM Enterprise X-Architecture Technology: Reaching the Summit, pp. 37–50 (2002)
10. Censier, L.M., Feautrier, P.: A new solution to coherence problems in multicache systems. IEEE Trans. Comput. **100**(12), 1112–1118 (1978)
11. Chandra, D., Guo, F., Kim, S., Solihin, Y.: Predicting inter-thread cache contention on a chip multi-processor architecture. In: IEEE 11th International Symposium on High-Performance Computer Architecture, pp. 340–351 (2005)
12. Charlesworth, A.: Starfire: extending the SMP envelope. IEEE Micro **18**(1), 39–49 (1998)
13. Charlesworth, A.: The sun fireplane interconnect. IEEE Micro **22**(1), 36–45 (2002)
14. Clapp, R., Lovett, T.: STiNG: A CC-NUMA computer system for the commercial marketplace. In: IEEE 23rd Annual International Symposium on Computer Architecture (1996)

15. Cvetanovic, Z.: Performance analysis of the alpha 21364-based HP GS1280 multiprocessor. In: IEEE Proceedings of 30th Annual International Symposium on Computer Architecture, pp. 218–228 (2003)
16. Frank, S.J.: Tightly coupled multiprocessor system speeds memory-access times, vol. 1. Electronics, United States (1984)
17. Galles, M., Williams, E.: Performance optimizations, implementation, and verification of the SGI challenge multiprocessor. In: Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, vol. 1, pp. 134–143. IEEE (1994)
18. Geer, D.: Chip makers turn to multicore processors. Computer **38**(5), 11–13 (2005)
19. Gharachorloo, K., Barroso, L.A., Nowatzyk, A.: Efficient ECC-based directory implementations for scalable multiprocessors. In: Proceedings of the 12th Symposium on Computer Architecture and High-Performance Computing (SBAC-PAD 2000) (2000)
20. Gharachorloo, K., Sharma, M., Steely, S., Van Doren, S.: Architecture and design of AlphaServer GS320. In: ACM SIGARCH Computer Architecture News **28**, 13–24. ACM (2000)
21. Goodman, J.R.: Using cache memory to reduce processor-memory traffic. In: 25 Years of the International Symposia on Computer Architecture (selected papers), pp. 255–262. ACM (1998)
22. Horel, T., Lauterbach, G.: UltraSPARC-III: designing third-generation 64-bit performance. IEEE Micro **19**(3), 73–85 (1999)
23. Hsu, L.R., Reinhardt, S.K., Iyer, R., Makineni, S.: Communist, utilitarian, and capitalist cache policies on CMPs: caches as a shared resource. In: Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques, pp. 13–22. ACM (2006)
24. Iyer, R.: CQoS: a framework for enabling QoS in shared caches of CMP platforms. In: Proceedings of the 18th Annual International Conference on Supercomputing, pp. 257–266. ACM (2004)
25. Katz, R.H., Eggers, S.J, Wood, D.A., Perkins, C, Sheldon, R.G.: Implementing a cache consistency protocol, vol. 13. IEEE Computer Society Press (1985)
26. Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, J., Nakahira, D., Baxter, J., Horowitz, M.: The stanford flash multiprocessor. In: IEEE Proceedings the 21st Annual International Symposium on Computer Architecture, pp. 302–313 (1994)
27. Lenoski, D., Laudon, J., Gharachorloo, K., Gupta, A., Hennessy, J.: The directory-based cache coherence protocol for the DASH multiprocessor, vol. 18. ACM (1990)
28. Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J., Horowitz, M., Lam, M.S.: The stanford dash multiprocessor. Computer **25**(3), 63–79 (1992)
29. Loudon, J., Lenoski, D.: The SGI origin: a ccNUMA highly scalable server. In: Proceedings of the 24th International Symposium on Computer Architecture. Silicon Graphics Inc. (1997)
30. Martin, M.M., Harper, P.J., Sorin, D.J., Hill, M.D., Wood, D.A.: Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors. In: IEEE Proceedings of the 30th Annual International Symposium on Computer Architecture, pp. 206–217 (2003)
31. Martin, M.M., Hill, M.D., Wood, D.A.: Token coherence: decoupling performance and correctness. In: IEEE Proceedings of the 30th Annual International Symposium on Computer Architecture, pp. 182–193 (2003)
32. Martin, M.M., Sorin, D.J., Ailamaki, A., Alameldeen, A.R., Dickson, R.M., Mauer, C.J., Moore, K.E., Plakal, M., Hill, M.D., Wood, D.A.: Timestamp snooping: an approach for extending SMPs. In: ACM SIGARCH Computer Architecture News. **28**, pp. 25–36. ACM (2000)
33. Martin, M.M.: Token Coherence, University of Wisconsin (2003)
34. Martin, M.M., Hill, M.D., Wood, D.A.: Token coherence: a new framework for shared-memory multiprocessors. IEEE Micro **23**(6), 108–116 (2003)
35. Martin, M.M., Sorin, D.J., Beckmann, B.M., Marty, M.R., Xu, M., Alameldeen, A.R., Moore, K.E., Hill, M.D., Wood, D.A.: Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. In: ACM SIGARCH Computer Architecture News **33**(4), 92–99 (2005)
36. Marty, M.R.: Cache coherence techniques for multicore processors. PhD thesis, University of Wisconsin (2008)

37. McCreight, E.M.: The dragon computer system. In: Microarchitecture of VLSI Computers, pp. 83–101. Springer (1985)
38. Moore, G.E.: Cramming more components onto integrated circuits. Reprinted from Electronics **38**(8), 114 (1965) IEEE Solid-State Circuits Newslett. **11**(5), 33–35 (2006)
39. Mukherjee, S.S., Bannon, P., Lang, S., Spink, A., Webb, D.: The alpha 21364 network architecture. IEEE Hot Interconnects **9**, 113–117 (2001)
40. Nowatzyk, A., Aybay, G., Browne, M., Kelly, E., Lee, D., Parkin, M.: The S3. mp scalable shared memory multiprocessor. In: IEEE Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences, vol. 1, pp. 144–153 (1994)
41. Rao, W.: Multi processors, their memory organizations and implementations by Intel and AMD (2009) http://ece.uic.edu/~wenjing/courses/fa08ECE569/ECE569/w21.pdf
42. Sorin, D.J., Plakal, M., Condon, A.E., Hill, M.D., Martin, M.M.K., Wood, D.A.: Specifying and verifying a broadcast and a multicast snooping cache coherence protocol. IEEE Trans. Parallel Distrib. Syst. **13**(6), 556–578 (2002)
43. Tang, C.: Cache system design in the tightly coupled multiprocessor system. In: Proceedings of the National Computer Conference and Exposition, 7–10 June 1976, pp. 749–753. ACM (1976)
44. Tendler, J.M., Dodson, J.S., Fields, J., Le, H., Sinharoy, B.: POWER4 system microarchitecture. IBM J. Res. Dev. **46**(1), 5–25 (2002)
45. Thacker, C.P., Stewart, L.C., Satterthwaite Jr, E.H.: Firefly: a multiprocessor workstation. IEEE Trans. Comput. **37**(8), 909–920 (1988)
46. WindRiver, Wind River Simics “Full System Simulation”. www.windriver.com/products/simics/