RING-DATA ORDER: A New Cache Coherence Protocol for Ring-based Multicores

Jin Young Park, Lynn Choi
Samsung Electronics, School of Electrical and Computer Engineering, Korea University
o2id@samsung.com, lchoi@korea.ac.kr

ABSTRACT

RING-DATA ORDER mechanism is motivated to solve cache coherence ordering for ring-based multicores. For point to point ring interconnects, the existing ORDERING-POINT mechanism is known for poor average performance due to its long response latency. GREEDY-ORDER mechanism is simple but the performance is still unacceptable because the requestor should often retry to complete a coherent operation. RING-ORDER mechanism is the best known algorithm so far but special storage and management overhead are expected due to its token management. This paper proposes a new mechanism called RING-DATA ORDER which is as simple as GREEDY-ORDER but more efficient than RING-ORDER. RING-DATA ORDER determines the order by data transfer sequence. The unbounded retries of GREEDY-ORDER can be eliminated by blocking incoming coherence request which contains data. Blocked request restarts to traverse the ring interconnect when the coherence operation for current node is completed. Blocking is also used in RING-ORDER mechanism. But RING-DATA ORDER also eliminates token and its additional management overhead without sacrificing performance. Using a custom-built multicore simulator with profiled SPEC 2000 integer benchmark suites we demonstrate that RING-DATA ORDER can achieve the same level of performance as RING-ORDER without the overhead of token management.

KEYWORDS: Cache, Coherence, Ordering, Multicore

1. INTRODUCTION

Continuous advancements of semiconductor process technology have made it possible to integrate multiple processing cores on a single silicon die at an acceptable cost. The primary advantage of chip multiprocessor architecture is that it can offer more scalable performance with less power consumption than the existing superscalar architecture by duplicating simple cores.

To interconnect multiple processing cores on a die, the bus interconnect is widely used. The bus-based interconnect can easily provide total ordering for coherent operations, but its multi-tap transmission line characteristics limit its maximum operating frequency. As the number of processing cores increases for bus-based multicores, the usable bandwidth per core decreases, which makes the bus interconnect inefficient except for small-scale multicores. Point to point interconnects such as ring network may become an alternative solution since point to point interconnects connects two cores directly without a centralized element. This can maximize usable bandwidth per core.

The advantage of the bus interconnect is that all bus transactions are visible to all nodes on the bus at the same time. But when point to point interconnects are used for multicores, each core has its own point of view for a coherent operation. Thus, coherence ordering mechanism must be employed. Existing solutions to coherence ordering for ring-based multicores are ORDERING-POINT[1]-[4], GREEDY-ORDER[5]-[7], and RING-ORDER[8] mechanisms.

ORDERING-POINT mechanism is the simplest approach by inserting ordering point in the ring. Ordering point plays a similar role as a centralized arbiter in bus-based interconnects. GREEDY-ORDER mechanism removes the ordering point but starvation may occur due to its distributed arbitration. In this context the starvation implies that a core may not complete its coherence request for an indefinite amount of time. RING-ORDER removes the starvation problem from GREEDY-ORDER and it also increases the effective bandwidth utilization per core but at the expense of extra token management. RING-DATA ORDER removes the token management overhead from RING-ORDER without sacrificing performance. RING-DATA ORDER determines the order of coherent requests by data transfer sequence. The closest neighbor node from current node which has the ownership for a specific memory block gets highest priority for coherent operations.

To verify and evaluate the performance of RING-DATA ORDER we have developed a custom multicore simulator. As an input to the simulator we also generate memory-related traces by profiling SPEC 2000 integer benchmark suites. Our experimentation data shows that the RING-DATA ORDER can achieve the same level of performance as RING-ORDER mechanism while it can successfully remove the extra overhead of token management in enforcing the coherence ordering

The rest of the paper is organized as follows. Section 2 surveys existing works related to RING-DATA ORDER. Section 3 describes the basic ideas and operations of RING-DATA ORDER mechanism with some coherence transaction examples. Section 4 discusses our simulation methodology. Section 5 shows the simulation results of RING-DATA ORDER compared to RING-ORDER mechanism. Section 6 concludes the paper.

2. RELATED WORKS

Ring-based point to point links provide distributed and concurrent execution of coherent operations, with less interconnection wire load and enhanced operating frequency. However ring based interconnect does not provides the total ordering for coherent operations because each core has its own point of view to the same coherence request. But in case of bus interconnect, all processing cores monitor the bus transactions and recognize the coherent operation equally. Several coherence ordering mechanisms for ring based point to point interconnect have been proposed.

The simplest approach for coherence ordering mechanism is known as ORDERING-POINT [1]-[4] mechanism. ORDERING POINT inserts an ordering point into the ring as the starting point of coherent operations. The first coherent operation arrived at the ordering point has the highest priority and it must be handled first. Each processor core requests a coherent operation to the ring interconnect and the request traverses to ordering point. Coherence ordering is maintained by the order of arrival of request to ordering point. When a request arrives at the ordering point, the ordering point activates it and the request traverses the ring interconnect to complete a coherent operation. When the request arrives at the ordering point again, then the ordering point returns the acknowledgement message to the requesting processor core. The performance of ORDERING-POINT mechanism can be described as 2N hops on average where N represents the number of processing cores. Average 2N hops can be calculated as N/2 hops to reach the ordering point, N hops to perform the coherent operation, and N/2 hops to receive an acknowledgement. Average latency is huge because every request traverses the ring interconnect twice. Thus, the bandwidth utilization becomes poor.

GREEDY-ORDER [5]-[7] is a more advanced mechanism to perform coherence ordering. GREEDY-ORDER mechanism does not require special element such as ordering point. In GREEDY-ORDER mechanism, the request which acquires a valid data item first has the highest priority. In some cases, the following request may not acquire valid data and may return to the requesting processor core without valid data. In this case, processor core re-requests that request to the ring interconnect. The performance of GREEDY-ORDER mechanism can be described as average N(1+M) hops where N denotes the number of processing cores and M denotes the number of unbounded retries. Most requests can complete the coherent operation in N hops, but some requests may lay on the unbounded retry situation. Unbounded retries reduce the effective bandwidth utilization per processing core. In GREEDY-ORDER mechanism, the order of coherence operation is determined by the sequence of data acquirement. Who first acquires the data gets the highest priority for coherence operation.

RING-ORDER [8] mechanism is the most recent ring-based coherence scheme which is based on Token Coherence [9]. Like Token Coherence, RING-ORDER mechanism is essentially based on token counting. Token counting and special priority token transfer can be used to determine the order of coherent requests. RING-ORDER mechanism blocks incoming requests from remote processing cores when it contains a priority token. Priority token is used to represent the ownership of a specific memory block. To modify a specific memory block in Token Coherence, the core must gather all tokens for that memory block. Coherent operation can be performed when sufficient tokens are gathered. On average it takes N hops to perform a coherence ordering for each request. Useless token and data transfer can be blocked. This increases the effective bandwidth utilization. But extra token transfer overhead is expected always. When the number of processing cores increases, the total number of token for a memory block also increases. And token preserving is also required. Especially when cache replacement occurs, all tokens for that block should be backed up safely to remote or lower hierarchy of memory system and this can cause extra overhead.

3. RING-DATA ORDER MECHANISM

3.1. Motivation and Basic Idea

RING-DATA ORDER mechanism is basically motivated from GREEDY-ORDER mechanism and RING-ORDER mechanism. GREEDY-ORDER mechanism determines coherence order by acquiring data but RING-ORDER mechanism blocks the data and token transfer which is requested by remote processing core. RING-DATA ORDER mechanism merges the advantages of these two approaches.

Each processing core determines whether the incoming request is including data transfer or not. If data transfer is indicated then current processing core checks the issued request table entry which contains the requests that have not been completed yet. A request which includes data transfer can be blocked to a special buffer so called a pending buffer and continues to traverse the ring interconnect when the issued request returns to current processing core. And when an incoming request arrives at current processing node without data transfer, current processing node also blocks that incoming request if issued request table match and current processing node can be the start point of data transfer such as "MODIFIED", "EXCLUSIVE", "OWNED" state for that incoming request.

RING-DATA ORDER mechanism removes the abundant retries of GREEDY-ORDER mechanism and extra overhead of token management in RING-ORDER mechanism by determining data transfer. Every coherent request is completed when it arrives at the requestor. When there are competitions for a specific memory block, the order of requests is determined by data transfer order which begins from a core that has the ownership for that memory block.

3.2. Correctness of RING-DATA ORDER

In RING-DATA ORDER, the order of all coherent requests for a specific memory block is determined by the order of ring with ownership. Ownership means that the current processing node can be the start point of data transfer when the current processing node is in "MODIFIED" or "EXCLUSIVE" or "OWNED" state for that memory block. At a certain time, the ownership for a specific memory block is maintained by only a single processing node.

The ownership is also traverses the ring interconnects. The order of coherent requests can be determined by the distance from a node which maintains the ownership for that memory block. Because the closest node A from ownership blocks all other coherent request until the request of node A returns to node A. This condition ensures the single writer policy for coherent requests.

By doing this, at most there can be only one writer for a specific memory block. Single writer policy is the necessary and sufficient condition for maintaining cache coherence. RING-DATA ORDER guarantees single writer policy and can be used as a cache coherence protocol. Ownership determination can be calculated by the following equation.

$$O_{NEXT} = \mathrm{MIN}\,(D_1, D_2, \ldots D_N)$$

O_{NEXT} represents the next node who will acquire the ownership and D_N represents the distance from the node which has the ownership to node N.

3.3. RING-DATA ORDER example

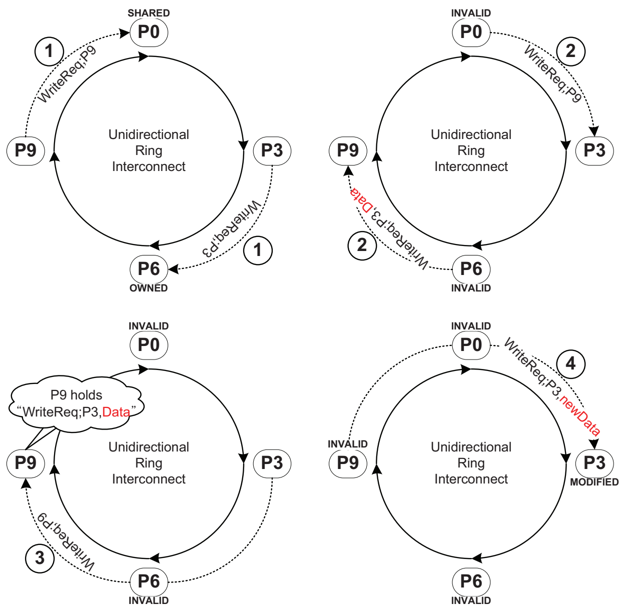

Figure 1. RING-DATA ORDER (write vs. write example)

Figure 1 shows the example of RING-DATA ORDER for WRITE and WRITE request competition. At first the processing core P3 and P9 launches the write request to the ring interconnect at the same time. Processing core P6 is changed to INVALID state and data traverses the ring continuously. At the same time, P9's write request arrives at P3. P9 blocks P3's write request with data when it arrives at processing core P9 and waiting for P9's write request return to P9. P9's write request finally arrives at P9 and then P9 completes the write operation. After the completion of P9's write operation, blocked P3's request continues to traverse the ring interconnect with modified data which is updated by P9. Finally P3's write request with modified data arrives at P3 to complete the write operation. P3 goes to MODIFIED state.

Figure 2 shows another example of RING-DATA ORDER for WRITE and READ request competition. At first the processing core P3 launches the read request and

P9 launches the write request to the ring interconnect at the same time. Processing core P0 is changed to INVALID state by the P9's write request. But when the P3's read request arrives at P9 with data which is acquired from P6, P9 blocks P3's read request and waiting for P9's write request returns to P9. P6 goes to INVALID state when P9's write request arrives at P6 and P9's write request returns to P9 with data. Now P9 completes its write request and goes to OWNED state when P9 launches P3's read request with modified data to the ring interconnect. P3 goes to SHARED state when the P3's read request with data returns to P3.

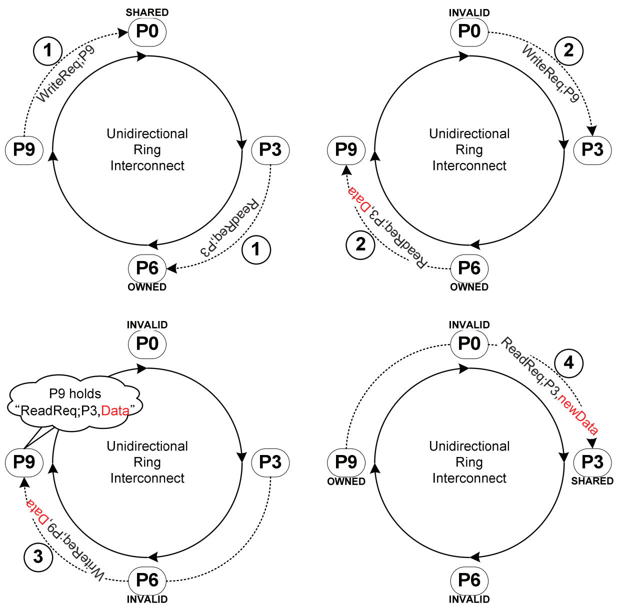

Figure 2. RING-DATA ORDER (write vs. read example)

Table 1. Comparison of Mechanisms

Mechanisms	Complexity	Hops (avg)	Bandwidth Utilization	Comments
ORDERING-POINT	Easy	2N	Low	Simple
GREEDY-ORDER	Normal	N (1+M)	Normal	M : retries
RING-ORDER	Hard	N	High	Token overhead
RING-DATA ORDER	Hard	N	High	

The average number of hops, design complexity and bandwidth utilization of different coherence ordering mechanisms are summarized in Table 1.

4. SIMULATION MODEL

Simulation methodology is composed of 2 parts. First, we use SimpleScalar[10] sim-cache simulator to extract the program behavior of SPEC 2000 integer benchmark suites. After profiling the program behavior, customized multicore simulator based on ring interconnect is used to evaluate the performance of RING-ORDER and RING-DATA ORDER mechanism. We did not consider ORDERING-POINT and GREEDY-ORDER mechanisms in our simulation since RING-ORDER outperforms both as evaluated in related works [8].

4.1. Profiling

By using sim-cache simulator in SimpleScalar 3.0d, all load/store operation to the cache system was extracted. SPEC 2000 integer (CINT2000) benchmark was used as test benchmark suite. CINT2000 64-bit alpha EV6 binaries are used. Coherence operation (read or write), execution gap since previous coherence operation, and 40-bit address are profiled into 7-byte packet. The size of profiled information is the result of 5 Billion instruction execution of CINT2000.

4.2. Simulator

To model and simulate the RING-DATA ORDER and RING-ORDER mechanisms, a special multicore simulator was developed. The development environment is 32-bit/64-bit Linux with gcc-4.1.2. To simulate a multicore processor, one of the profiled CINT2000 benchmark is assigned to a core. Each benchmark in the system is treated as single thread and accesses shared memory. The simulator supports 2 up to 32 cores with unidirectional ring interconnect. The number of point to point interconnection channels between cores is configurable. Initially a single point to point link channel is defined and a single request can traverse it. User can change the number of point to point communication channels to transfer multiple requests at once.

The configuration of each core is summarized in Table 2. Issued request table can have multiple entries if the processing core supports multithreading. Buffers for point to point communication are also defined. Buffer entry 0 is always used for local processing core. MOESI state machine updates L1 tag, IRT, BUFFER and PENDING. The size of L2 cache is defined as 64 times of L1 cache in a processing core.

The configuration of target system is summarized in Table 3.

Table 2. Processor Core Model

Items	Comments
L1 data cache	128 KB direct mapped cache 24-bit tag, 32 bytes per block, 12-bit index 2-bit offset, 64-bit data
Address	40-bit
IRT	1 issued request table entry
BUFFER	Point to point incoming buffer Default 32 entries (configurable) Entry 0 used for local requests Circular queue
PENDING	Pending buffer for blocking coherence requests which contains data transfer Default 8 entries (configurable) Circular queue
Protocol	MOESI protocol used

Table 3. System Configuration

Items	Comments
L2 cache	8 MB direct mapped cache 24-bit tag, 32 bytes per block, 18-bit index Perfect L2 cache model L2 access hit always L2 provides the data immediately
P2P Links	Default 1 P2P link between cores Link size is configurable

5. SIMULATION RESULT

33 CINT2000 integer benchmarks are profiled for multicore simulation. Each profiled data is treated as coherence traffic source and allocated to the specific processing core. The profiled benchmark execution is limited to 5 Billion cycles because simulation can take more than a month for a larger simulation. Multicore configurations from 2 to 32 cores are considered. The reports includes total execution cycles and processing core statistics such as L1 cache hit/miss count for local or incoming requests, L2 access count, program execution cycles to solve coherence operation and system configuration. RING-DATA ORDER and RING-ORDER mechanisms were evaluated while ORDERING-POINT and GREEDY-ORDER mechanisms are ignored.

Figure 3 shows the latency of RING-DATA ORDER mechanism normalized by RING-ORDER mechanism. The latency difference is under 1%. It means that the performance of RING-DATA ORDER mechanism is almost the same as that of RING-ORDER mechanism. It seems that there is no difference between RING-DATA ORDER and RING-ORDER mechanism. It is because, when a core requests a coherent operation, that core does not generate another coherent operation until previous request is completed. This situation makes the point to point link between cores in idle state. If the traffic is increased, the latency gap is expected to grow due to extra token management overhead of RING-ORDER.

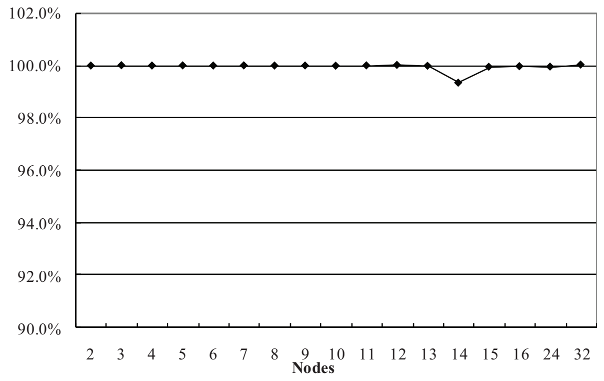

Figure 3. Normalized Latency (100 % = RING-ORDER)

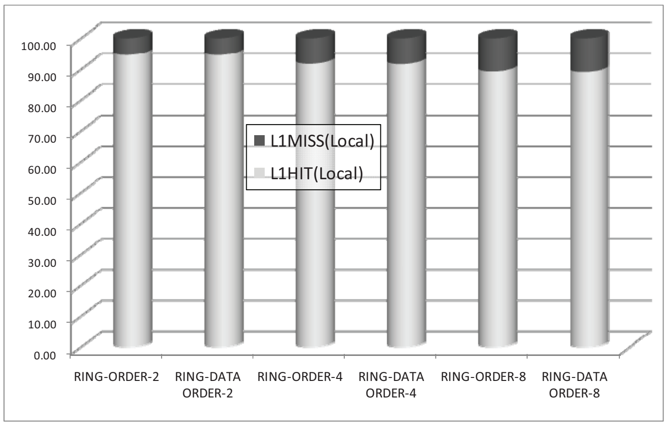

Figure 4. L1 Cache Hit Ratio for Local Requests

Figure 4 shows the L1 cache hit ratio for local requests. As the number of cores is increased, L1 cache hit ratio for local requests for both RING-ORDER and RING-DATA ORDER is decreased. It is due to L1 cache invalidation by incoming remote request.

Figure 5 shows the L1 cache hit ratio for external requests. It shows that around 15% of incoming request is correct cache coherence request for dual core system. For quad core system, the ratio goes down about 10% and for 8 core system, the ratio goes down about 6%. As the number of nodes is increased, the portion of coherence operation goes down.

Figure 6 shows the normalized L1 cache eviction ratio of RING-DATA ORDER. As the number of core is increased, eviction ratio goes down because the portion of independent request is increased. And RING-ORDER mechanism requires token eviction but RING-DATA

ORDER avoids this extra overhead. The communication traffic for RING-DATA ORDER is much optimized because of elimination of token.

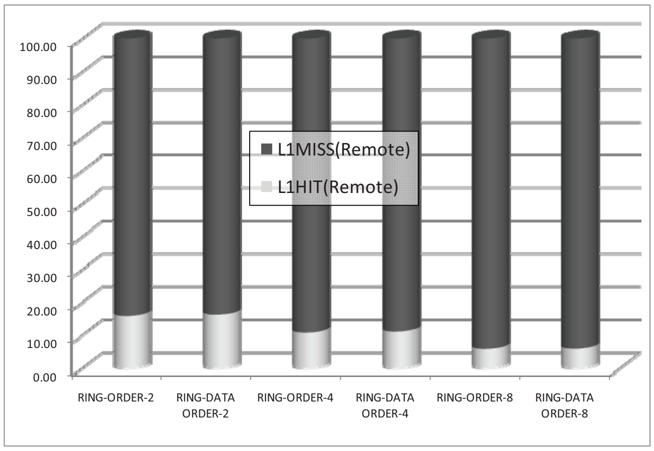

Figure 5. L1 Cache Hit Ratio for Remote Requests

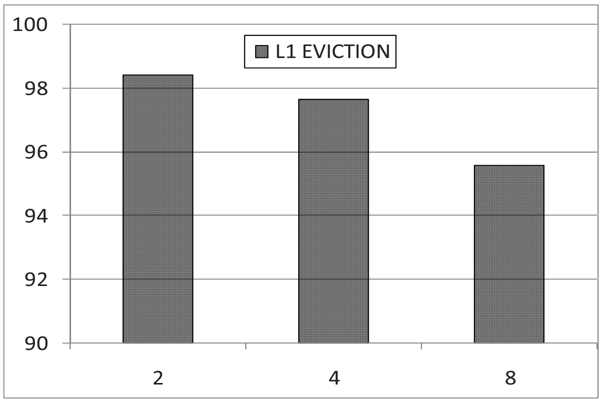

Figure 6. Normalized L1 Cache Eviction Ratio (100 % = RING-ORDER, lower is better)

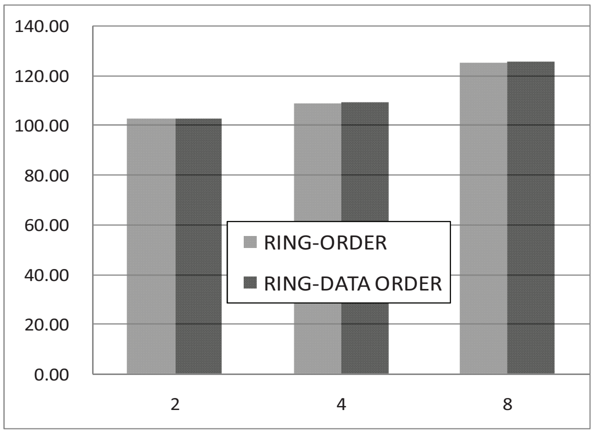

Figure 7. Single Program Execution Cycles for Various Size of Ring

Figure 7 shows average single program execution cycles for various size of ring interconnect. As the number of core is increased, single program execution time is increased because of latency to handle cache coherence. For dual core system, performance degradation for single program execution cycles is less than 3%. For four core system, it is measured less than 10%. In eight core system, single program execution cycles increased by 25%.

6. CONCLUSIONS

In this paper we propose a simple yet efficient coherence ordering mechanism called RING-DATA ORDER for ring-based multicores and multiprocessors. RING-DATA ORDER allows a coherence operation to travel the ring interconnect as in GREEDY-ORDER but it optimizes the request order and improves bandwidth utilization as in RING-ORDER mechanism while removing unbounded retries of GREEDY-ORDER and also the token manipulation overhead of RING-ORDER. Our detailed simulation results collected from profiled SPEC 2000 integer benchmark suites show that RING-DATA ORDER can successfully enforce coherence ordering without relying on token management and it can achieve the same level of performance as that of RING-ORDER. In addition, with higher traffic load RING-DATA ORDER is expected to show better performance than RING-ORDER by removing the tokens.

Although ring interconnects may be a good design choice for low to medium size multicores, the ring structure may have inherently long latency when the number of nodes in the ring increases since a coherent request must visit all nodes in the ring before returning to the originating node. For future work, we are currently working on extending the RING-DATA ORDER mechanism to be used for a more large-scale multicore systems based on a hierarchical ring structure.

REFERENCES

[1] C. N. Keltcher, K. J. McGrath, A. Ahmed, and P. Conway. "*The AMD Opteron Processor for Multiprocessor Servers*" IEEE Micro, 23(2):66–76 (2003)

[2] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing". In Proceedings of the 27th Annual International Symposium on Computer Architecture, p282–293, (2000)

[3] 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, p 13–24, Nov. (2000)

[4] D. Gustavson. "*The Scalable Coherent Interface and related standards projects*". IEEE Micro, 12(1):10–22, Feb. (1992)

[5] L. A. Barroso and M. Dubois. "Cache Coherence on a Slotted Ring". In Proceedings of the International Conference on Parallel Processing, p 230–237, Aug. (1991)

[6] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner. "*Power5 System Microarchitecture*". IBM Journal of Research and Development, 49(4) (2005)

[7] S. W. Chung, S. T. Jhang, and C. S. Jhon. "PANDA: ring-based multiprocessor system using new snooping protocol". In International Conference on Parallel and Distributed Systems, p10–17 (1998)

[8] Michael R. Marty and Mark D. Hill, "Coherence Ordering for Ring-based Chip Multiprocessors", International Symposium on Microarchitecture, (2006)

[9] Milo M.K. Martin, "Token Coherence", PhD Thesis, University of Wisconsin - Madison, (2003)

[10] http://www.simplescalar.com/