Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems

Abdul Naeem, Xiaowen Chen, Zhonghai Lu and Axel Jantsch
Department of Electronic Systems, Royal Institute of Technology, Sweden
E-mail: {abduln, xiaowenc, zhonghai, axel}@kth.se

Abstract-This paper studies realization and performance comparison of the sequential and weak consistency models in the network-on-chip (NoC) based distributed shared memory (DSM) multi-core systems. Memory consistency constrains the order of shared memory operations for the expected behavior of the multi-core systems. Both the consistency models are realized in the NoC based multi-core systems. The performance of the two consistency models are compared for various sizes of networks using regular mesh topologies and deflection routing algorithm. The results show that the weak consistency improves the performance by 46.17% and 33.76% on average in the code and consistency latencies over the sequential consistency model, due to relaxation in the program order, as the system grows from single core to 64 cores.

I. Introduction

There is a trend in the processor development from single core to multi-core architectures [1, 2, 3]. The Network-on-chip (NoC) can be used as a reliable and scalable communication medium among these cores in the system [4, 5, 6, 7]. As the number of processors grows on the chip their memory requirements also grow. The distributed shared memory (DSM) architecture is preferred, because a single centralized shared memory has the performance bottleneck in the multi-core systems. The processor development still confronts many challenges like memory consistency, coherence and parallel programming issues. The memory consistency decides the order of execution of shared memory operations up to the expectation of programmer in the multi-core systems. Various memory consistency models already proposed [8, 9, 10] are based on the ordering constraints on the shared memory operations in the multi-core systems. The strict memory consistency models disable the performance optimizations due to more restrictions on the ordering of shared memory operations. For instance, the sequential consistency model [11] allows minimum performance optimizations due to the strictness in the program order. The strict ordering on the shared memory operations are prohibitively expensive in the DSM based multi-core systems. Consequently, several relaxed ordering (like weak ordering, release consistency) [12, 13] emerged to allow these performance optimizations. Relaxed consistency models enhance the system performance significantly at the reasonable cost.

We investigate the performance of the sequential and weak consistency models that are realized in the NoC based multi-core (McNoC) systems. The strict sequential consistency model (also called as strong ordering) is the extension of uni-processor memory model applied to the multi-processor systems. It does not allow reordering in the shared memory operations. The weak consistency model (often called as weak ordering) is a relaxed consistency model which permits overlapping in the shared memory operations in the McNoC systems. The sequential consistency model is implemented by stalling the processor till the completion of previously issued operation. The weak consistency model is implemented by using a transaction counter in the platform hardware to avoid the interference between the data and synchronization operations. In the results section, the average, maximum code and consistency latencies are explored and compared for the various sizes of the 2D mesh networks. The average code latency is the average of execution times of codes running on the concurrent nodes in the McNoC system. The consistency latency is the code latency without the network latency and synchronization wait time. The differences in the code and consistency latencies for both the consistency models are significant in the large networks. These latencies are reduced by the weak consistency model in comparison to the sequential consistency model due to the allowed reordering in the memory operations.

The rest of the paper is organized as follows. In the next section, we review the related work. In section III, memory consistency background is discussed. In section IV, DSM based McNoC platform is discussed. In section V, realization of the sequential and weak consistency models are focused. In section VI, simulation results and performance analysis of the two consistency models in the McNoC system are described. Section VII summarizes our contribution.

II. Related Work

Very little work has been done on the memory consistency issue in the McNoC systems. Sarita V et al. [8] discussed memory consistency issues with an emphasis on the system optimizations they allow. They proposed the counter based technique to realize the weak memory consistency. The proposed counter keeps track of the outstanding data operations between the two consecutive synchronization operations. The data operations may still be reordered and overlapped with respect to each other. We realized the weak consistency model using the same concept in the McNoC systems. Petrot et al. [14] explored the reordering of synchronization and data operations due to the routing scheme, diverse paths, and physical location of the target in the NoC based shared memory multi-processor SoC architectures. They proposed that the initiator should wait for the response of the first target before sending the request to the second target to avoid the interference between the synchronization and data operations. But in our work such interference is avoided without suffering the relaxation in the program order. In [15], the scalability of weak consistency model in the NoC based multi-core architectures

is discussed. However, the performance of the weak consistency model is not compared with any other consistency model in McNoC system. In this paper the performance of the weak consistency is compared with the sequential consistency in the more flexible McNoC platform. Monchiero et al. [16] proposed a synchronization buffer, a hardware unit to support the memory controller and the lock is locally polled in the shared memory NoC based MPSoC systems. However, since all the synchronization requests issued by all nodes flow through the network into the stand alone synchronization module, the network congestion due to the heavy traffic may affect the system performance. Also, the target system architecture assumes the weak consistency model but does not mention how to constrain the shared memory operations for the weak consistency model.

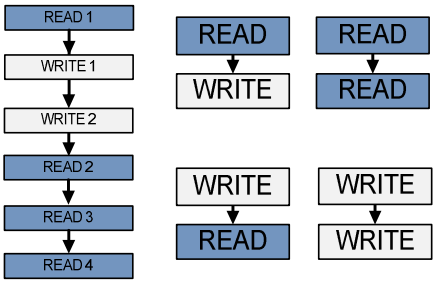
III. Memory Consistency Background

The memory consistency model is a contract between the programmer and parallel system. The programmer follows the rules that are guaranteed by the parallel system to get the predictable results of the shared memory operations. Shared memory access latency can be reduced by the hardware and software optimizations in the system architecture. These performance optimizations can reorder the shared memory operations and the system may give unexpected results. For the expected results, these reordering should be controlled carefully. Different memory consistency models enforce different ordering constraints on the shared memory operations [8]. The sequential consistency enforces strict ordering constraints on the shared memory operations. The IBM 370, total store ordering (TSO) and processor consistency (PC) models eliminate the ordering constraints between write followed by a read to a different location. The partial store ordering (PSO) model also in addition removes the ordering constraints among writes to the different locations. The weak consistency, release consistency, Alpha, relaxed memory order (RMO), and PowerPC models relax the program order requirement among all the shared memory operations to the different locations. In this paper, we focus on the two memory consistency models (sequential and weak consistency) that are realized in the McNoC systems.

A. *Sequential Consistency Model*

The sequential consistency defined by Lamport [11] has to maintain the program order among operations of each individual processor and sequential order among multiple processors in the system. In fact, according to the definition, reorder can be allowed since the result is the same as the strict program order expects. However, it will be difficult to implement due to that the consequence of each re-order has to be globally analyzed to ensure correctness. In the following, we discuss the sequential consistency model that literally follows the program order.

The sequential consistency (strong ordering) is a strict model expected by the programmer. A memory operation (read, write) cannot be reordered or overlapped with the following memory operation to the different locations in the shared memory. The shared memory operations are completed according to the program order as given in Fig. 1(a). The sequential consistency enforces the global orders on shared memory operations as given in Fig. 1(b).



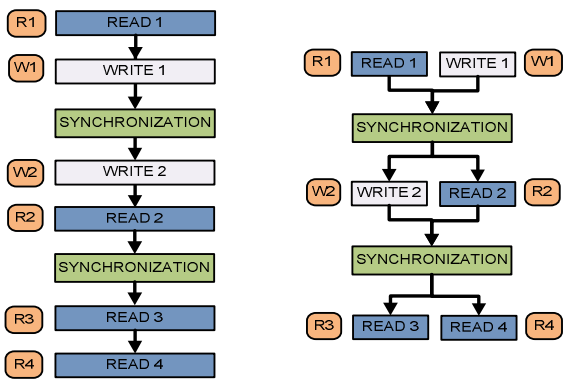
(a) (b)

Fig. 1. a) Strong Ordering b) Global orders to enforce

The sequential memory consistency model does not allow the performance optimizations [8] in the hardware (write buffer, cache, interconnection network) and in the software (compiler reordering, register allocation) due to the strict order enforcement on the shared memory operations. Relaxed memory consistency models permit such optimizations. These models relax the program order requirement to allow the possible reordering in the shared memory operations by the system optimizations. The shared memory operations may not complete according to the program order. The overall program correctness is ensured to enforce ordering constraints on a subset of shared memory operations. We consider the weak consistency model which relaxes the strict program order requirement and allows reordering in the shared memory operations. The shared memory operations (read, write) can be reordered with the following shared memory operations in the specific program segments.

B. *Weak Consistency Model*

The weak consistency model proposed by Dubois et al. [12] classifies shared memory operations as *synchronization* and *data* operations. Synchronization operations are related to the special synchronization variables (locks, semaphores) in the shared address space. The lock must be gained exclusively in the shared memory multi-processor systems. Data operations are the load-store operations related to the ordinary shared variables. To make the strong ordering comparable with the weak ordering Fig. 2(a) also considers the synchronization operation.

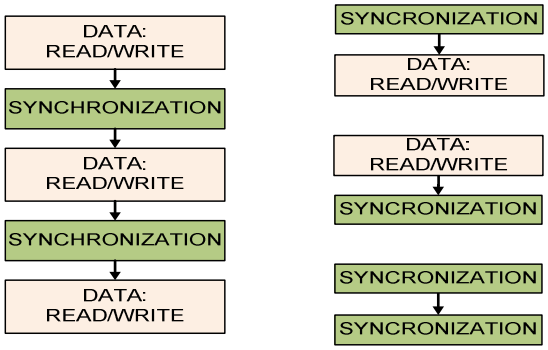


(a) (b)

Fig. 2. a) Strong Ordering b) Weak Ordering

According to the weak memory consistency data operations (R1, W1) can be reordered with respect to one another as shown in Fig. 2(b). Similarly, the data operations (W2, R2) and (R3, R4) in their respective sections can also be reordered with respect to each. The data operations (R1, W1) are not allowed to be reordered and overlapped with the data operations (W2, R2) and (R3, R4) in another section. Also, the data operations (R1, W1, R2, W2, R3, R4) are not allowed to be reordered with respect to the synchronization operations.

The weak consistency model enforces some global orders on the shared memory operations as shown in Fig. 3(b). The enforcement of these global orders on the shared memory operations ensures the program correctness in the weak consistency model with the permitted relaxation in data operations. It also ensures to get the final consistent result of the program execution in the multi-processor systems. All previously issued outstanding data operations must be completed before the issuance of synchronization operation. Similarly, previously issued outstanding synchronization operation must also be completed before the issuance of any data operation. The transaction counter based technique for the enforcement of the global orders, required for the weak consistency model is illustrated in the later section.

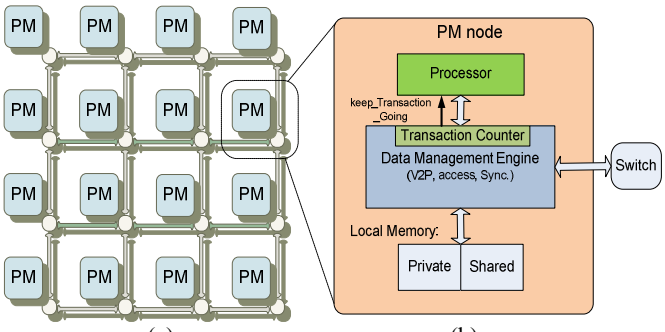


(a) (b)

Fig. 3. a) Weak Ordering b) Global orders to enforce

IV. DSM based McNoC Platform

Fig. 4(a) shows a homogenous NoC based multi-core system having one type of nodes. The system is comprised of 16 nodes interconnected via a packet-switching network. All the nodes are connected in a 2D mesh topology. Each node represents a typical processor-memory (PM) node in the platform. The structure of a PM node is given in Fig. 4(b). Each PM node consists of a processor, data management engine (DME) and a local memory. The DME in the PM node is connected to the NoC, processor and local memory. Routers in the NoC use deflection routing algorithm to route the packets to the proper destinations. The platform uses distributed shared memories (DSM) which are integrated with processors in the nodes. All shared parts in the local memories form virtual memory, which is organized as a DSM and use a single global memory address space. Two addressing schemes are adapted and for the shared memory access, a virtual-to-physical (V2P) address translation is required.



(a) (b)

Fig. 4. a) Homogeneous McNoC b) PM node

The DME is in each node of the McNoC platform. Its architecture is given in Fig. 5. The DME is connected to the CPU core, the local memory, and the network. The DME contains core interface control unit (CICU), network interface control unit (NICU), control store, mini-processor-A, mini-processor-B, synchronization supporter and a transaction counter. The CICU and NICU provide the hardware interface to the local core and network respectively. The two mini-processors are the central processing engine. Micro-program is initially stored in the local memory, and is dynamically uploaded into the control store during the program execution. The synchronization supporter coordinates the two mini-processors to avoid simultaneous accesses to the same synchronization variable (lock) and guarantees atomic read-modify-write operations over the lock. Both the local memory and the control store is dual ported, port A and B, which are connected to the mini-processor A and B, respectively. The transaction counter (TC) is used in the DME hardware to realize the weak consistency model in the McNoC platform. The sequential consistency model platform does not use the TC. The shared memory operations (read, write) and synchronization operations are implemented in the DME micro-code. The DME offers flexibility to implement a verity of read-write commands and different synchronization primitives.

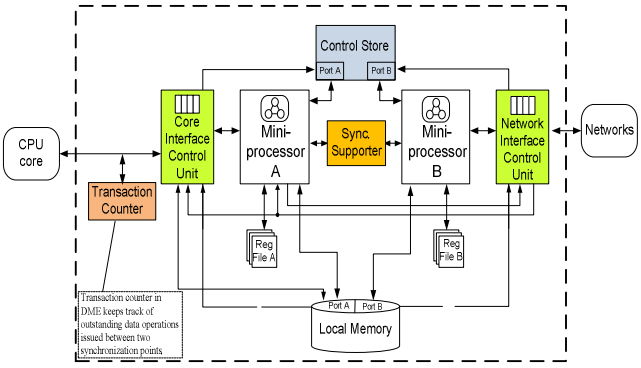


Fig. 5. Structure of DME

Local shared memory operations are accomplished by mini-processor-A within the node. Similarly, for the remote shared memory and lock access, messages are sent to the remote node by mini-processor-A via network. Remote shared memory operations are completed either by the remote data return or write acknowledgment by mini-processor-B in the remote node. The DME provides both the hardware and software support for the memory

synchronization. The synchronization supporter (SS) provides the underlying hardware support for the memory synchronization. The SS can simultaneously receive and respond to two synchronization requests from the local core via mini-processor-A and the remote cores via mini-processor-B. Two special micro-operations (*ll* and *sc*) together with the SS ensure atomic read-modify-write operations over locks. Various synchronization primitives (spin lock, queue locks) like test-and-set() are implemented in the DME micro-code using these special *ll* and *sl* micro-operations. The *ll* micro-operation checks the lock address in the SS, if it is not there, then the lock address is recorded in the SS over the entire period of lock acquire or release operations. The lock is gained exclusively in the shared memory. If the address is present i.e., other node is gaining access to the same lock, then the relevant mini-processor is stalled until the completion of preceding acquire or release operation (removal of lock address from SS) by the other node. The *sl* micro-operation not only acquires or releases a lock but also remove the lock address from the SS. For more detail we refer you to [17].

V. Realization of Memory Consistency Models

A. *Sequential Consistency Model*

The sequential memory consistency model in the McNoC platform is realized by stalling the processor on the issuance of shared memory operation till the completion of preceding operation. On the completion of previous operation, processor issues the next operation. The completion of preceding operation is indicated by the return data or acknowledgment. The program order is maintained due to the strict order between the shared memory operations and sequential order is maintained by the read-modify-write memory operation in the system. Fig. 6 demonstrates the realization scheme of the sequential consistency model in the McNoC system.

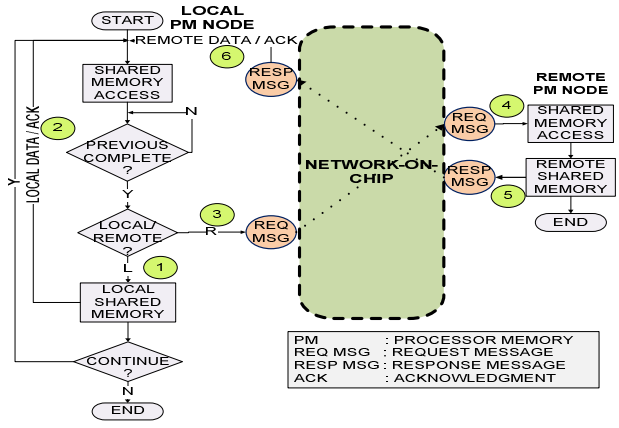


Fig. 6. Sequential consistency implementation scheme

The local shared memory operations are issued (1) and completed (2) in the local node. For the remote shared memory accesses, message passing (3, 4) is carried out to the remote node via network. Remote operations complete via response messages (5, 6). Overall, the memory operations are issued and completed in the order specified in the program.

The transaction processing FSM for the sequential consistency model (FSM-SEQ) in the DME is given in Fig. 7. The FSM-SEQ initializes and configures the DME in the working state. The FSM-SEQ asserts the Keep-Transaction-Going (KTG) output of the DME low (refer to Fig. 4) as it receives a memory operation from the processor and issue it to the memory system. The active low KTG stall the processor and the FSM-SEQ switches to the wait state. The FSM-SEQ returns to the working state on the completion of the previously issued operation, and asserts the KTG output high. Active high KTG signals the processor to issue the next memory operation in the program.

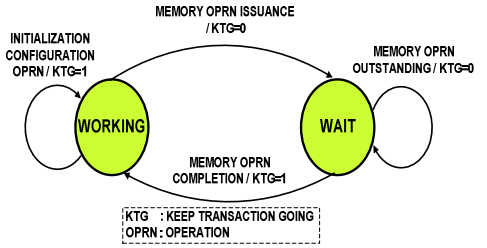


Fig. 7. Transaction processing FSM for sequential consistency

B. *Weak Consistency Model*

The transaction counter (TC) based approach [8] is adapted for the realization of the weak memory consistency model in the McNoC systems. The counter is implemented in the DME hardware of each node to keep track of the outstanding data operations issued between the two synchronization points. It is incremented and decremented by the issuance and completion of data operations correspondingly. It is not affected by the synchronization operations. Fig. 8 illustrates the realization scheme of the weak consistency model in the McNoC platform.

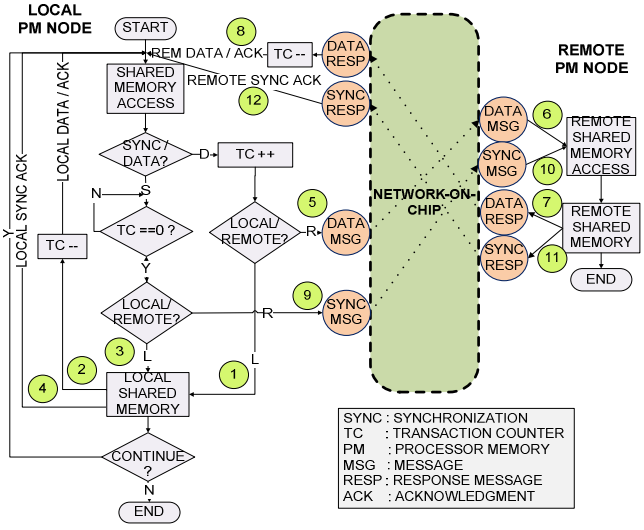


Fig. 8. Weak consistency implementation scheme

The issuance (1) and completion (2) of the data operations to the local shared memory increment and decrement the TC. Local or remote synchronization operations are not issued (3, 9) until the TC becomes zero, i.e., the completion of all the previously issued outstanding data operations. The TC is neither incremented nor decremented by the synchronization

operation. The atomicity over the synchronization operation is achieved by using two special purpose micro-operations *sl* and *ll* and the synchronization supporter. The subsequent data operations wait and cannot be issued until the completion of the previously issued synchronization operation. A local synchronization operation (3) is completed by synchronization acknowledgment (4). For the remote data operations message passing (5, 6) is carried out to the remote node via network. Remote data operations are completed by response messages (7, 8) from the remote node. The completions of remote data operations also decrement the same TC in the local node. The message passing (9, 10, 11, 12) is involved to the remote node for the remote synchronization operations. In summary, the TC in each node is incremented with the issuance of the local and remote data operations (1, 5). It is decremented by the completion of previously issued local and remote data operations (2, 8). It is not affected by the local (3, 4) and remote (9, 10, 11, 12) synchronization operations.

The transaction processing FSM for the weak memory consistency model (FSM-WK) in the DME is depicted in Fig. 9. The FSM-WK is in the working state initially. The FSM-WK issues outstanding data operations in the working state as it asserts the KTG output of the DME high to the processor on the issuance of each data operation. When the FSM-WK receives the synchronization operation, its issuance to the memory system is postponed until the TC becomes zero (completion of previously issued outstanding data operations). The FSM-WK asserts the KTG output low on the issuance of the synchronization operation to the memory system to stop the subsequent data operations from the processor. The FSM-WK switches to the synchronization wait state and waits for the completion of the previously issued synchronization operation. The FSM-WK returns to the working state on the reception of the synchronization acknowledgment and asserts the KTG output high. The processor can send now the outstanding data operations again until the next synchronization point.

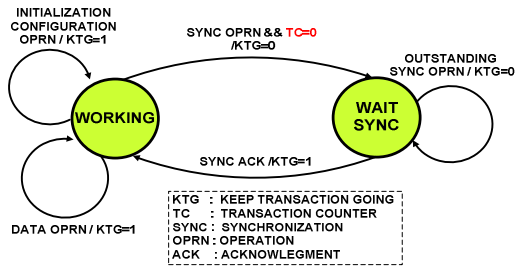


Fig. 9. Transaction processing FSM for weak consistency

VI. Experiments and Results

A. *Purpose and Setup*

We analyze the performance of the sequential and weak consistency models that are realized in the McNoC systems. The affects of network size on the code and consistency latencies are investigated. Average and maximum latencies of the sequential and weak consistency models are compared with the increasing size of the NoC in the system. The experimental setup uses the DME based flexible and configurable McNoC platform. Tests are performed for various sizes of the networks. Regular 2D mesh topology network using deflection routing algorithm is considered in the tests. Two hotspot nodes are used one for the critical section (CS-node) and the other for the lock (SYNC-node). As shown in Fig. 10, the critical section in the CS-node is protected by the lock maintained in the SYNC-node.

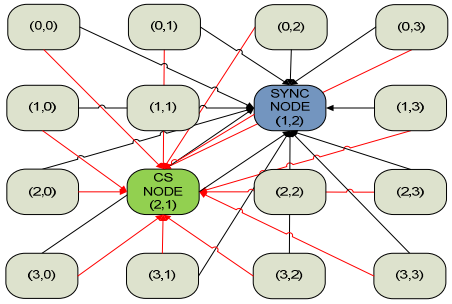


Fig. 10. Synchronization and data requests

Each node sends the synchronization request to the SYNC-node. The shared memory locations in the CS-node are accessed exclusively after acquiring the lock successfully in the SYNC-node. After the critical section execution, the lock is released for the other waiting acquires requests. The hotspot traffic is generated by the code running on each node in the platform. The pseudo-test-code is given in Fig. 11.

```
//NON-CRITICAL SECTION
<MEMORY_WRITE>, <ADDRESS>, <DATA>;     // REMOTE SHARED WRITE
<MEMORY_READ>, <ADDRESS>;              // REMOTE SHARED READ

//LOCK ACQUIRE
<LOCK_ACQUIRE>, <ADDRESS>;             // REMOTE LOCK ACQUIRE

//CRITICAL SECTION
<MEMORY_WRITE>, <ADDRESS>, <DATA>;     // REMOTE SHARED WRITE
<MEMORY_READ>, <ADDRESS>;              // REMOTE SHARED READ

//LOCK RELEASE
<LOCK_RELEASE>, <ADDRESS>;             // REMOTE LOCK RELEASE

//NON-CRITICAL SECTION
<MEMORY_WRITE>, <ADDRESS>, <DATA>;     // REMOTE SHARED WRITE
<MEMORY_READ>, <ADDRESS>;              // REMOTE SHARED READ
```

Fig. 11. Pseudo-test-code

B. *Code Latency*

The average code latency is the average of codes execution time running on the concurrent nodes in the McNoC system. The average and maximum code latencies for the different sizes of networks are shown in Fig. 12. The average code latency increases exponentially for both the sequential and weak consistency models as the network grows from single core to 64 cores. It is due to the increasing synchronization wait time and network congestion in the larger networks. The network congestion and synchronization wait time may suffer the system performance in the larger networks. Average code latency for the weak consistency model in the 8x8 network is about 241.96 times of the single core, whereas for the sequential consistency model it is 353.67 times. The weak consistency model reduces these latencies due to the relaxation in the program order as compared to the sequential consistency model. The performance gain of the weak consistency over the sequential consistency model is 46.17% in the average code latency, as the system grows from single core to 64 cores.

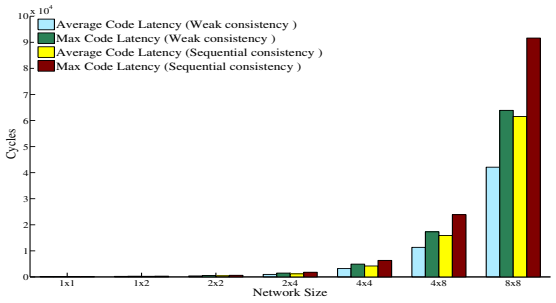


Fig. 12. Code latency

C. *Consistency Latency*

The consistency latency is the code latency without the network latency and synchronization wait time. The average and maximum consistency latencies observed in the experiments for various sizes of network are shown in Fig. 13. The increasing trend in the consistency latency is linear for both the consistency models as the network size scales. The average consistency latency for the weak consistency model in the 8x8 network is about 1.54 times of the single core, while for the sequential consistency model it is 1.96 times. The performance gain of the weak consistency over the sequential consistency model in the average consistency latency is 33.76%.

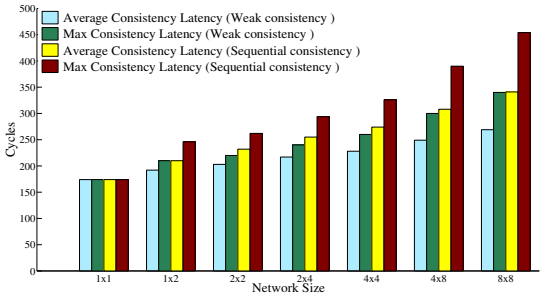


Fig. 13. Consistency latency

VII. Conclusion

We explore the memory consistency models in the DSM based McNoC systems. The sequential consistency model is realized by stalling the processor to serialize the issuance and completion of the shared memory operations. The hardware transaction counter can enforces the required global orders needed for the weak consistency model. The transaction counter based realization of the weak consistency model avoids the possible interference problem between the data and synchronization operations. We analyze the performance of both the consistency models in the McNoC systems. The mesh topology networks are considered in the McNoC platform for both the consistency models. All the nodes synchronized over the same block (queue) lock in a particular node. The average and maximum code and consistency latencies increase significantly for both the consistency models as the network scales. The experimental results show that the weak consistency model performs 46.17% and 33.76% better on average in the code and consistency latencies in comparison to the sequential consistency model due to the relaxation in the program order.

Acknowledgements

This work has been supported partially by the FP7 EU project Mosart under contract number IST-215244, and the HEC/SI joint scholarship program of Pakistan and Sweden.

References

[1] M. Horowitz, W. Dally, “How scaling will change processor architecture,” in: Proc. of International Solid-State Circuits Conference. (ISSCC’04), Digest of Technical Papers, pp. 132–133, Feb. 2004.

[2] S. Borkar, “Thousand core chips: A technology perspective,” in: Proc. of the 44th Design Automation Conference. (DAC’07), pp. 746–749, June 2007.

[3] S. Vangal, J. Howard, G. Ruhl, “An 80-tile 1.28tflops network-on-chip in 65nm cmos,” in: International Solid-State Circuits Conference. (ISSCC’07), Digest of Technical Papers, pp. 98–100, Feb. 2007.

[4] A. Jantsch, H. Tenhunen, “Networks on Chip,” Kluwer Academic Publishers, 2003.

[5] L. Benini, G. D. Micheli, “Networks on Chip: A new SoC paradigm,” IEEE Computer, 35(1):70–78, January 2002.

[6] T. Bjerregaard, S. Mahadevan, “A survey of research and practices of network-on-chip,” ACM Computing Surveys, vol. 38, no. 1, pp. 1–51, March 2006.

[7] J. D. Owens, W. J. Dally, “Research challenges for on-chip interconnection networks,” IEEE MICRO, vol. 27, no. 5, pp. 96–108, Oct. 2007.

[8] S. V. Adve, Kourosh Gharachorloo, “Shared Memory Consistency Models: A Tutorial,” Digital Western Research Laboratory, report no. 95/7, Palo Alto, California 94301 USA, September 1995.

[9] D. Mosberger, “Memory Consistency Models, ACM SIGOPS Operating Systems Review,” Vol. 27, No. 1, USA, January 1993.

[10] R. C. Steinke, G. J. Nutt, “A unified theory of shared memory consistency,” Journal of the ACM, vol. 51, no. 5, pp. 800-849, 2004.

[11] L. Lamport, “How to Make a Multiprocessors Computer That Correctly Executes Multiprocessor Programs,” IEEE Transaction on Computers, Vol.C-28. No. 9, pp. 690-691, September 1979.

[12] M. Dubois, Christoph Scheurich, Fayb Briggs, “Memory access buffering in multiprocessors,” in: Proc. of the 13th Annual International Symposium on Computer Architecture, pp. 434-442, June 1986.

[13] K. Gharachorloo, D. Lenoski, J. Laudon, Phillip Gibbons, Anoop Gupta, John Hennessy, “Memory consistency and event ordering in scalable shared-memory multiprocessors,” Computer Architecture News, 18(2): 15-26, June 1990.

[14] F. Petrot, A. Greiner, P. Gomez, “On cache coherency and memory consistency issues in NoC based shared memory multiprocessor SoC architectures,” in: Proc. of 9th EUROMICRO Conf. on Digital System Design: Architectures, Methods and Tools, pp. 53-60, Croatia 2006.

[15] A. Naeem, X. Chen, Z. Lu, and A. Jantsch, “Scalability of Weak Consistency in NoC based Multicore Architectures,” in Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS), pp. 3497-3500, Paris, France, June 2010.

[16] M. Monchiero, G. Palermo, C. Silvano, and O. Villa, “Power/performance hardware optimization for synchronization intensive applications in MPSoCs,” in Proc. of the Conf. on Design, automation and test in Europe (DATE’06), pp. 606–611, 2006.

[17] X. Chen, Z. Lu, A. Jantsch and S. Chen, “Supporting Distributed Shared Memory on Multi-core Network-on-Chips Using a Dual Microcoded Controller,” In Proc. of the Conf. for (DATE’10), Dresden, Germany, March 2010.