Scalability of Relaxed Consistency Models in NoC based Multicore Architectures

Abdul Naeem[1], Xiaowen Chen[1, 2], Zhonghai Lu[1] and Axel Jantsch[1]
[1]Department Electronic Systems, School of ICT, Royal Institute of Technology (KTH), Stockholm, Sweden
[2]Institute of Microelectronics and Microprocessor, School of CS, NUDT, Changsha, China
{abduln, xiaowenc, zhonghai, axel}@kth.se

ABSTRACT
This paper studies realization of relaxed memory consistency models in the network-on-chip based distributed shared memory (DSM) multi-core systems. Within DSM systems, memory consistency is a critical issue since it affects not only the performance but also the correctness of programs. We investigate the scalability of the relaxed consistency models (weak, release consistency) implemented by using transaction counters. Our experimental results compare the average and maximum code, synchronization and data latencies of the two consistency models for various network sizes with regular mesh topologies. The observed latencies rise for both the consistency models as the network size grows. However, the scaling behaviors are different. With the release consistency model these latencies grow significantly slower than with the weak consistency due to better optimization potential by means of overlapping, reordering and program order relaxations. The release consistency improves the performance by 15.6% and 26.5% on average in the code and consistency latencies over the weak consistency model for the specific application, as the system grows from single core to 64 cores. The latency of data transactions grows 2.2 times faster on the average with a weak consistency model than with a release consistency model when the system scales from single core to 64 cores.

Categories and Subject Descriptors
C.1.4 [Parallel Architectures]: Distributed architectures;

B.3 [Memory Structures]: Shared Memory;

General Terms
Performance, Design

Keywords
Synchronization, Scalability, Memory consistency, Distributed shared memory.

1. INTRODUCTION
As a general trend, processor development has been shifted from single sequential processor to parallel multi-core systems. Most computer companies for example, AMD, Intel, Sun, ARM and IBM have shifted their next generation designs to be based on multi-core systems [1, 2, 3, 4]. NoC based multi-core (McNoC) systems are promising solutions to the modern and future processor design challenges [5, 6]. In order to reuse the huge amount of legacy code and facilitate programming, distributed memories are preferable to be shared. Multi-threaded applications running on multi-core, shared memory platforms suffer from memory consistency problem. Various memory consistency models have been proposed as alternative solutions [7, 8, 9, 10]. The sequential consistency model does not allow performance optimizations due to the strictness in the program order. The strict ordering on memory operations is prohibitively expensive in distributed shared memory (DSM) systems. Consequently, a large number of relaxed ordering (weak ordering, release consistency) [11, 12, 13, 14] have been developed to allow for compiler and hardware optimizations. The relaxed models enhance the system performance significantly at reasonable cost. The weak consistency model distinguishes shared memory operations as *synchronization* and *data* operations. Atomic synchronization operations on reserved shared variables (locks) protect shared data operations (critical section). The data operations can be reordered and overlapped in the weak consistency model. The release consistency model further differentiates synchronization operations as *acquire* and *release* operations. It permits additional reordering in the data operations of critical and non-critical sections as compared to the weak consistency model.

We investigated scalability of the transaction counter based relaxed consistency models in the NoC based system. Both the consistency models used locks in the shared address space for the synchronization. We explored and compared code, synchronization and data latencies for the mesh networks with various sizes in the tests. The code latency comprises of synchronization and data latencies. The differences in the code and data latencies for both the consistency models are significant in large networks. It is due to the allowed additional reordering in the release consistency model. The synchronization latency increases exponentially with the network growth. It is due to waiting for acquiring dependent locks. The system performance is greatly suffered from synchronization latency in the large networks. Average and maximum code, synchronization and data latencies increase for both the consistency models in the larger networks. These latencies are reduced by the release consistency in comparison to the weak consistency model as a result of further relaxation in the program order.

The rest of the paper is organized as follows. In the next section, we describe related work to the scalability analysis of synchronization in multi-core systems. In section 3, DSM based McNoC platform is discussed. In section 4, transaction counter based realization of weak consistency model while in section 5, transaction counter based realization of release consistency model has been focused. In section 6, simulation results and scalability analysis of both the consistency models in the NoC based system is described. Section 7 summarizes our contribution and future plan.

2. RELATED WORK
The NoC work has so far mainly focused on the architectural and modeling aspects [15, 16]. Very few researchers have worked on the scalability analysis of synchronization among cores in the NoC based multi-processor systems. Oreste Villa et al. [17] performed

quantitative analysis to understand how different topologies behave when dealing with different SW/HW barriers implementations in the NoC systems. Four different barriers were implemented and evaluated for a number of cores (from 4 to 128) and five different network topologies. Simple network topologies proved to be more efficient than the complex topologies. However, their work focused on off-chip main memory rather than on-chip DSM based systems. Petrini et al. [18] analyzed the scalability of HW and SW based barriers designed and implemented in a programmable network interface card for the quadrics interconnection networks. This work evaluated that the HW approach is more efficient than the SW approach in the presence of network contention. Without the network contention both algorithms can be used interchangeably on a flat-tree topology for systems of 64 to128 nodes. Although this work is not related to the on-chip scalability analysis. Sarita V et al. [11] discussed memory consistency issues with an emphasis on the system optimizations they allow. They proposed to use a counter to realize weak memory consistency. The proposed counter keeps track of the outstanding data operations between two synchronization operations. The data operations may still be reordered and overlapped with respect to one another. Petro et al. [19] explored the reordering of synchronization and data operations due to diverse paths, routing scheme and physical location of the target in the network.

3. DISTRIBUTED SHARED MEMORY BASED MULTI-CORE NOC PLATFORM

3.1 McNoC platform

Figure 1a shows a homogenous NoC based multi-core system having one type of nodes. The system is composed of 16 nodes interconnected via a packet-switched network.

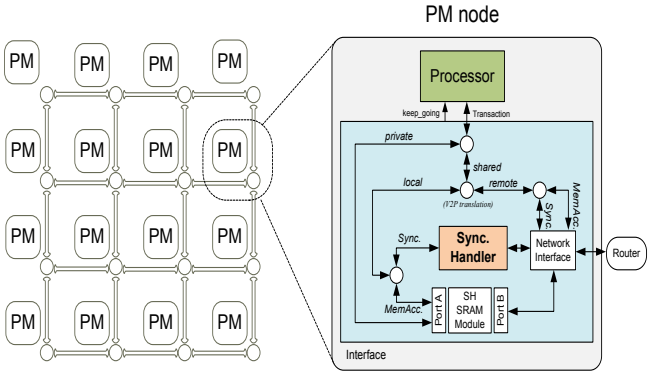

Figure 1. a) Homogeneous McNoC b) PM node

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 Figure 1b. Each PM node consists of a processor, synchronization handler, network interface and local memory. The network interface performs packetization, de-packetization, arbitration, queuing and connects a PM node to the NoC. Routers use X-Y deterministic routing algorithm to rout the packets to proper destinations.

3.2 Distributed shared memory (DSM)

The platform uses distributed shared memories which are integrated with processors in the nodes. The local memory is partitioned into private and shared parts. The private memory can only be accessed by the local processor and it is physical. All of shared memories are visible to every node in the network and known as virtual memory. The virtual memory is organized as a distributed shared memory. All shared parts of local memories can logically form a single global memory address space. Two addressing schemes physical and logical (virtual) are adopted. Software developers can access DSM in a single virtual address space.

The traditional centralized memory system has the performance, power and area bottleneck in on-chip systems [20]. The centralized UMA architecture not only limits the scalability but also become the hot spot on the chip. The DSM architecture is much preferred to single centralized memory. As the number of processor grows on the chip their memory requirements also grow. There is already a lot of memory on logic chip and there will be even more. Recent Tilera's 100 cores processor TILE-Gx100 has 32MB of on-chip distributed caches. The distributed shared memory can be used for several purposes. One way is to use it for caches another way is to use most of it as a shared memory for sharing data among different cores. That may allow for a more efficient memory handling and management in the applications where memory needs are well known (e.g. multimedia). The DSM architecture will also assist the 3D integration process in the chip design where logic die with one or more DRAM memory dies stacked on top of it. The expected gains in the 3D integration process can be in terms of memory bandwidth, latency and power [21]. The 3D integration can reduce individual die size, improve chip yield, ease packaging and reduce power for main memory [22]. Such 3D memory stacks are critically dependent on the through-silicon-via (TSV) diameters and overhead.

3.3 Synchronization handler

Each PM node in the platform has a synchronization handler which maintains a set of synchronization variables and queues. It provides efficient synchronization support and mutual exclusion for synchronization operations. The synchronization handler is connected to the processor and network interface. It serves the synchronization requests from the local processor within the node and remote processors via the network.

3.3.1 Structure

The synchronization handler is mainly composed of a synchronization variable pool, scheduling logic, two physical channels and a crossroad as given in Figure 2. The synchronization variable pool has N locks one bit each. These locks have special reserved addresses in the shared address space known to the software developers. Programmers can use it through relevant programming language constructs. There are two access ports for these locks, one from the local processor and other from the network. Two physical channels can receive and respond simultaneously to two synchronization requests from the local processor and the network. Each physical channel owns a buffer queue of depth Q. The crossroad dispatches the synchronization requests to the proper physical channels. The scheduling logic controls the crossroad to determine directions of synchronization requests. It organizes the two physical channels in to N virtual channels logically. It also monitors lock‘s status, maintains the virtual channels and performs correct actions on the coming synchronization requests. It handles the lock requests in an efficient way to reduce the contention, overhead and improve

the response time. To access a shared lock in a sequential order, the synchronization handler architecture uses logical virtual channels per lock to maintain synchronization requests over the same lock.

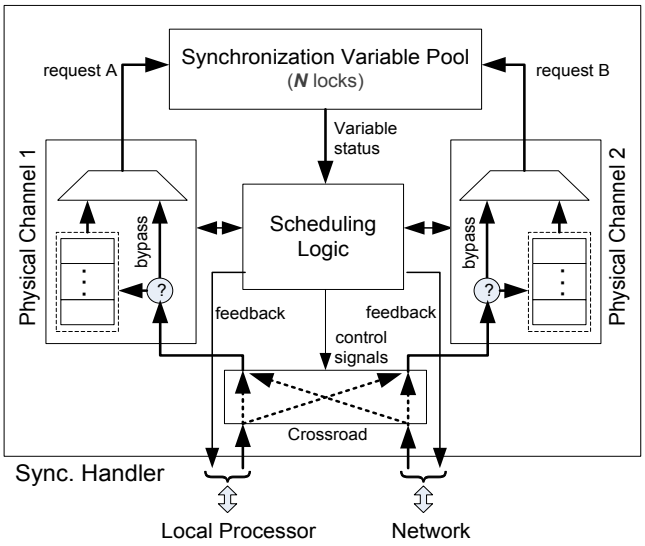

Figure 2. Structure of Synchronization handler

3.3.2 Operation mechanism

The synchronization requests come from the local processor and the network, go through the virtual channels and finally enter the synchronization variable pool. The synchronization request may be either lock acquire or release operation. The lock release request always goes along the bypass path, while the lock acquire request may go along the bypass path only when the related virtual channel is empty. The lock acquires request changes lock status from unlocked to locked and the lock release request changes the status from locked to unlocked. The scheduling logic monitors the status of all the locks and performs actions accordingly. The synchronization message size is 80 bits and has six fields, i.e., message type, source node number, destination node number, transfer type (reserved bit), address and data. The synchronization request may be dependent (to the same lock) or independent (to different locks). We consider two situations accordingly.

3.3.2.1 Two simultaneous independent requests

If the requested lock's status is unlocked then the lock acquire request goes along the bypass path to access the synchronization variable pool directly. The status of the lock is changed to locked. The lock acquire acknowledgement with the success status is sent back to the source node. If the requested lock is in the locked status, and the buffer queues in the physical channels are full, then the lock acquire acknowledgement with fail status is sent back to the source node. If the buffer queues are not full, the request is buffered in the related virtual channel until the lock is released. When a lock release request comes and there are one or more lock acquire requests buffered in the virtual channel which is related to the lock to-be-released. The lock release request does not need to access the synchronization variable pool but the first lock acquire request in the virtual channel is forwarded into the synchronization variable pool. The related lock acquire acknowledgement with the success status is sent back. The lock's status is still locked. If the related virtual channel is empty the lock release request goes along the bypass path to access the synchronization variable pool and changes the lock's status to be unlocked.

3.3.2.2 Two simultaneous dependent requests

If the requested lock status is unlocked and one of the two dependent lock acquire requests is selected to access the synchronization variable pool through the bypass path directly. The lock acquire acknowledgement with the success status is sent back to the relevant node. If the buffer queues are full, the lock acquire acknowledgement with the fail status is sent back to which node the other request is from. If not, the other is buffered in the related virtual channel. Meanwhile, the lock's status is changed into locked. If the requested lock is on its locked status, and the buffer queues are full, two lock acquire acknowledgement with fail status are sent back to the requesting nodes. If there is only one empty item in the buffer queues, one request is buffered in it and a lock acquire acknowledgement with fail status is sent back to the other node. If there are at least two empty items in the buffer queues, the two requests are buffered in the related virtual channel. When the dependent lock acquire request and lock release request come simultaneously, and there is one or more lock acquire requests buffered in the related virtual channel, then the lock release request does not access the synchronization variable pool. The first lock acquire request in the virtual channel is forwarded into the synchronization variable pool and the related lock acquire acknowledgement with the success status is sent back. While the coming lock acquire request goes into the virtual channel. Therefore, the lock's status is still locked. If the related virtual channel is empty and two requests comes, the lock release request does not need to access the synchronization variable pool while the lock acquire request goes along the bypass path to access the synchronization variable pool. The lock's status is still locked.

4. TRANSACTION COUNTER BASED WEAK CONSISTENCY

4.1 McNoC platform for weak consistency

Figure 3 shows our McNoC platform for the weak consistency model. It is identical to the platform discussed earlier except for the transaction counter which is used to realize the weak consistency model.

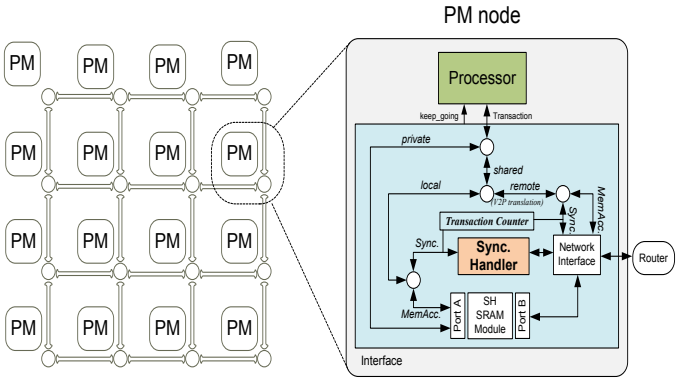

Figure 3. NoC platform for weak memory consistency

4.2 Weak memory consistency

The sequential consistency requires strict program order for individual processor and sequential order among multi-processor in the parallel system. It enforces the global order on each individual shared memory operation pair according to the program order. It does not allow performance optimizations in the hardware (cache, interconnection network) and software (compiler reordering, register allocation) in multi-processor systems [7]. Relaxed memory consistency models (weak consistency, release consistency) permit such optimizations. They relax the program order and enhance the system performance as compared to the sequential consistency model. The weak consistency model classifies shared memory operations as *synchronization* and *data* operations. It enforces the following global orders on shared memory operations:

- Synchronization to data
- Data to synchronization
- Synchronization to synchronization

The data operations (1, 3, 2) before, after and between the synchronization operations can be reordered as shown in Figure 4a. All previous data operations must be completed before the issuance of synchronization operation and vice versa. The global orders need to be enforced for the weak consistency model is given in Figure 4b. We illustrate the transaction counter based technique for the enforcement of the global orders in the upcoming section.

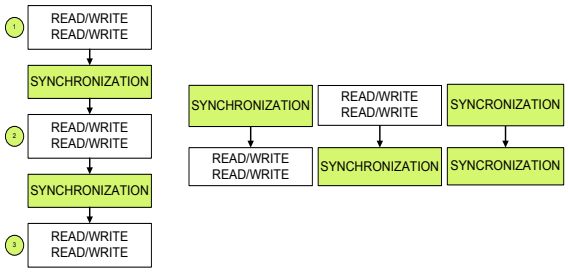

Figure 4. a) Weak Ordering b) Global orders to enforce

4.3 Realization of weak memory consistency

We adopted the transaction counter based approach [7] to realize the weak memory consistency model in the McNoC system. Transaction counter in each node keeps track of outstanding data operations. The counter is incremented and decremented by the issuance and completion of data operations correspondingly. It is not affected by the synchronization operations. The counter zero value indicates completion of all previously issued data operations. Synchronization operations are not issued until the transaction counter becomes zero. Figure 5 illustrates shared memory operations initiated by the processor in each node. After virtual to physical address translation shared memory operation is checked whether it is in the local or remote node. Local shared memory operations are accomplished within the node. Messages are sent to the remote node for the remote shared memory accesses. Shared memory operations are classified into synchronization and data operations. Local data operations in the critical section of code are issued to the shared locations (1) in the same node. A data operation may be memory read (load) or write (store) operation and is completed by either local data return or write acknowledgment respectively (5-1). Local data operations issued to the shared locations (2) in the non-critical section of code are also completed locally within the node either by local data returns or write acknowledgments (5-2). Local synchronization operations are issued (3) to the local memory mapped synchronization handler and are completed by synchronization acknowledgments (5-3). For remote memory operations (data, synchronization) message passing (4) is carried out to the remote node in the network. Remote shared data operations in the critical section are issued to the remote shared locations (6). Remote data operations are completed either by the remote data returns or write acknowledgments (9-6). Similarly, (7) and (9-7) are for the remote data operations in the non-critical section of code. Remote synchronization operations (8, 9-8) are issued to the remote memory mapped synchronization handler.

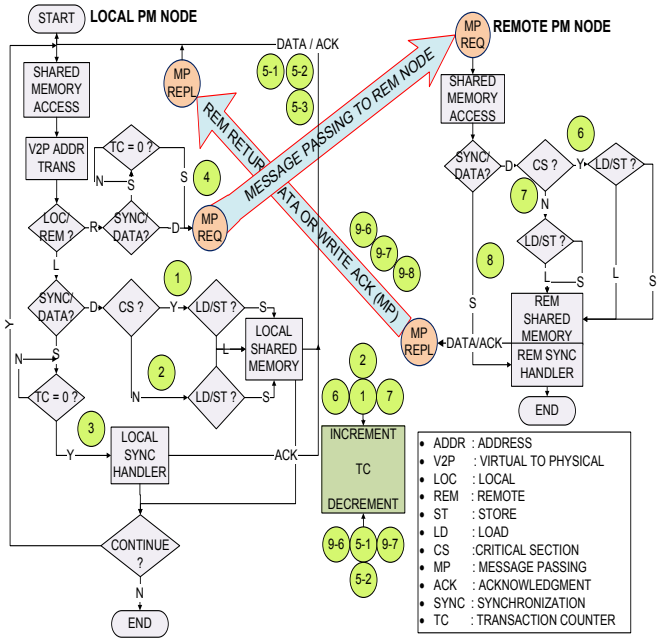

Figure 5. Transaction counter based weak consistency

Overall, transaction counter in each node is incremented with the issuance of local data operations (1, 2) and remote data operations (6, 7). It is decremented by the completion of previously issued local data operations (5-1, 5-2) and remote data operations (9-6, 9-7). It is not affected by the local synchronization operations (3, 5-3) and remote synchronization operations (8, 9-8).

5. TRANSACTION COUNTER BASED RELEASE CONSISTENCY

5.1 McNoC platform for release consistency

The NoC based multi-core platform for the release consistency model is similar to the weak consistency platform except for the number of transaction counters, requesting messages, response messages and the number of commands. Two transaction counters are used to realize the release consistency model in the McNoC platform.

5.2 Release memory consistency

The release consistency is a refinement of the weak ordering. It allows further overlapping, reordering and relaxation in the

program order. It further distinguishes synchronization operations as *acquire* and *release* operations. An acquire operation must be performed before the issuance of any data operation in the critical section (CS) and in the non-critical section (NCS) after it. All the data operations in the critical section and non-critical section prior to the release operation must be completed before the issuance of release operation. The release consistency model enforces the following global orders on shared memory operations as shown in Figure 6:

- Acquire to data
- Data to release
- Acquire to release
- Release to acquire

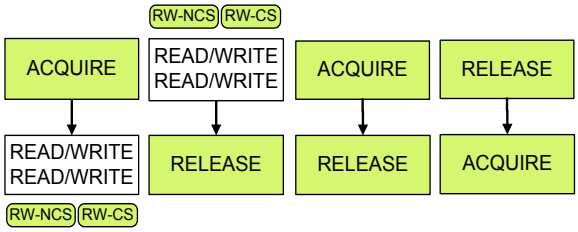

Figure 6. Global orders enforcement for release consistency

Figure 7 demonstrates that (RW2-CS, RW4-CS) and (RW1-NCS, RW3-NCS, RW5-NCS) are data operations in the critical and non-critical section of code. Acquire operation (ACQUIRE-L1) must be performed before any data operation in the critical section (RW2-CS) and also before the non-critical section (RW3-NCS) after it. The serial order violation between acquire operation to the following data operations may change the system behavior.

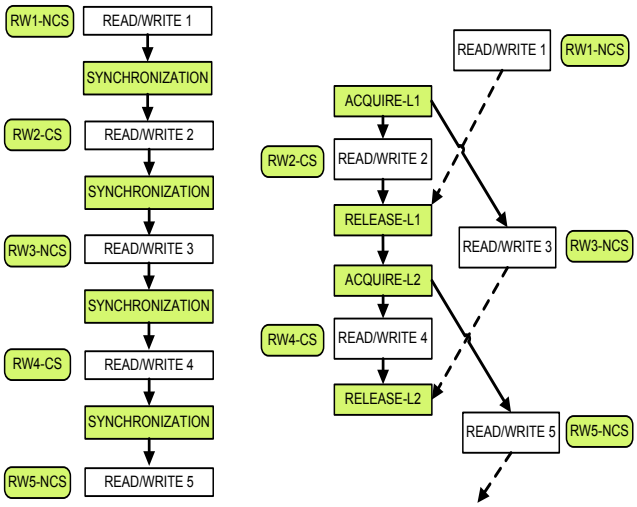

Figure 7. a) Weak Ordering b) Release Consistency

All the data operations in the critical section (RW2-CS) and non-critical section (RW1-NCS) before the previous acquire operation must be completed before the issuance of release operation (RELEASE-L1). The serial order violation between the preceding data operations and release operation may also change the system behavior. The reordering between the release operation and previous critical section data operations is forbidden, i.e., exit from critical section in the middle. The reordering between release operation and non-critical section data operations violate the program correctness. The program order relaxation in the release consistency model must ensure the overall program correctness according to the global orders enforcement. After the completion of acquire operation (ACQUIRE-L1) all the data operations in critical section (RW2-CS) and non-critical section after release operations (RW3-NCS) can be reordered. The data operations (RW3-NCS) in the non-critical section can be overlapped with the data operations (RW2-CS) in the critical section. Also, data operations in the critical section (RW4-CS) before the release operation (RELEASE-L2) and data operations in the non-critical section (RW3-NCS) before the previous acquire operation can be reordered. Ideally, non-critical section data operations (RW3-NCS) should overlap with the data operations in just preceding and following critical sections (RW2-CS, RW4-CS). The program order can be relaxed ideally for the code segment from acquire operation (ACQUIRE-L1) of one lock to the release operation of just next lock (RELEASE-L2). But practically, enforcement of the global order on shared memory operations for the release consistency model squeeze the relaxation window to the two adjacent release operations. The preceding non-critical section data operations must be completed before the issuance of the next release operation to ensure the program correctness. The data operations (RW3-NCS) in the non-critical section can practically be reordered only with the data operations in the following critical section (RW4-CS) and not with the preceding critical section data operations (RW2-CS). The data operations (RW2-CS) in the critical section can be reordered with the preceding non-critical section data operations (RW1-NCS). Besides ordering between data and synchronization operations (acquire-data, data-release) ordering among synchronization operations (acquire-release, release-acquire) must also be ensured. An acquire operation must be completed before a release operation on the same lock. Otherwise, a critical section will be entered without a lock and release of a lock is tried that is not yet locked. The release operation on the previous lock must be completed before the issuance of acquire operation of the next lock. Otherwise, two different critical sections will be entered simultaneously under the same lock which is illegal and may leads to the system failure. To enforce all the requisite serial orders, transaction counters based realization of the release consistency model is illustrated. The counters avoid all the possible interference problems between the synchronization and data operations.

5.3 Realization of release memory consistency

To realize the release memory consistency model in the multi-core system, transaction counters are used. We implemented two transaction counters in each node for two different types of data operations. Transaction counter1 (TC1) keeps track of outstanding data operations issued in the non-critical section of code. Transaction counter2 (TC2) keeps track of outstanding data operations issued within the critical sections of code. Each counter is incremented and decremented by the issuance and completion of relevant data operations correspondingly. Both the counters are not affected by acquire and release synchronization operations. "TC1=0" indicates the completion of all the previously issued data operations in the non-critical section of code. "TC2=0" indicates the completion of all the previously issued data operations in the critical section of code. Acquire and release synchronization operations are not issued until the relevant transaction counters become zero. The local data operations in the critical and non-critical sections are issued to the shared locations (1, 2) within the node as given in Figure 8. These data operations are completed locally in the same node. A data operation may be completed by

either local data return or write acknowledgment, respectively (6-1, 6-2). Acquire operations are not issued to the local or remote synchronization handlers until TC2 in the local node becomes zero. Release operations are not issued to the local or remote synchronization handlers until both the counters become zero in the local node. Local acquire synchronization operation is issued (3) to the local synchronization handler and is completed by synchronization acknowledgment (6-3). Local release synchronization operation is also issued (4) to the local synchronization handler.

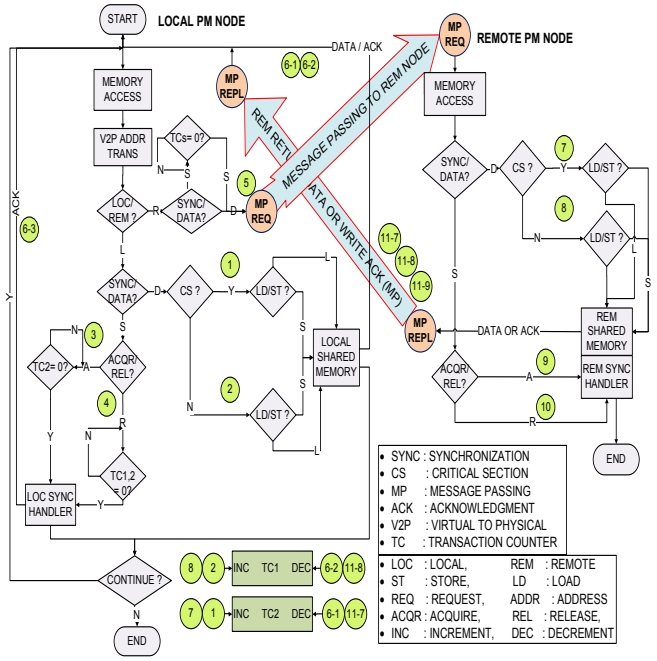

Figure 8. Transaction counter based release consistency

For remote memory accesses message passing (5) is carried out to the remote node in the network. Remote shared data operations in the critical section are issued (7) to the remote shared locations. Remote shared data operations in the non-critical section are issued (8) to the remote shared locations and are completed either by the remote data returns or write acknowledgments (11-7, 11-8). The issuance and completion of remote data operations affect the transaction counters in the local node. Remote synchronization operations (acquire, release) are issued to the remote synchronization handler (9, 10). Overall, TC1 in each node is incremented with the issuance of local and remote data operations in the non-critical section (2, 8). It is decremented by the completion of previously issued local and remote data operations in the non-critical section (6-2, 11-8). TC2 in each node is incremented with the issuance of local and remote data operations in the critical section (1, 7). It is decremented by the completion of previously issued local and remote data operations in the critical section (6-1, 11-7). Both the transaction counters are not affected by the local synchronization operations (3, 4, 6-3) and remote synchronization operations (9, 10, 11-10).

6. Experiments and results

We analyzed scalability of the transaction counter based relaxed consistency models in the McNoC system. Tests were performed for various network sizes. We investigated the effect of network size on the code, synchronization and data latencies. Average and maximum latencies were compared for the weak and release consistency models with increasing size of the McNoC system. In the experimental platforms for both the consistency models processor in each node was replaced by the stimulus to initiate the data and synchronization operations. The synchronization handler in both platforms has 256 locks in the shared address space. Transaction counters enforced the required global orders for the relaxed consistency models. The NoC supports both 2D mesh and torus topologies but we considered regular mesh topologies networks in the tests. Priority based round-robin arbitration and X-Y deterministic routing were used. Experiments were performed with the simple and short code running on each node in the platform. The pseudo-code is given in Figure 9. The code has data and synchronization operations.

```
••••••
// NON-CRITICAL SECTION
STIMULUS (MEMORY_WRITE, ADDRESS, DATA);      // REMOTE SHARED WRITE
STIMULUS (MEMORY_READ, ADDRESS, DATA=0);     // REMOTE SHARED READ

// LOCK ACQUIRE
STIMULUS (LOCK_ACQUIRE, ADDRESS, DATA=0);    // REMOTE LOCK ACQUIRE

// CRITICAL SECTION
STIMULUS (MEMORY_WRITE, ADDRESS, DATA);      // REMOTE SHARED WRITE
STIMULUS (MEMORY_READ, ADDRESS, DATA=0);     // REMOTE SHARED READ

// LOCK RELEASE
STIMULUS (LOCK_RELEASE, ADDRESS, DATA=0);    // REMOTE LOCK RELEASE

// NON-CRITICAL SECTION
STIMULUS (MEMORY_WRITE, ADDRESS, DATA);      // REMOTE SHARED WRITE
STIMULUS (MEMORY_READ, ADDRESS, DATA=0);     // REMOTE SHARED READ
••••••
```

Figure 9. Pseudo-code running on each node

The lock protects the shared memory access in the critical section as shown in Figure 10. Both the lock and critical section can be in any node of the network. For example the critical section in the CS node is protected by the lock maintained in the SYNC node. Every node sends synchronization (acquire, release) request to the SYNC node. On successful lock acquire in the SYNC node, it accesses the shared memory location in the CS node exclusively. After the critical section execution, the lock is released for other waiting acquire synchronization requests.

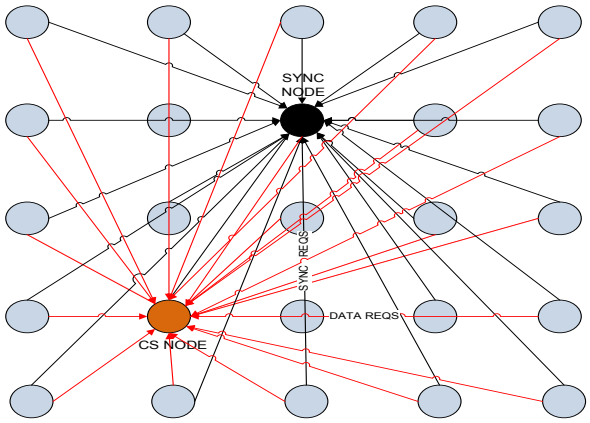

Figure 10. Synchronization and data requests

6.1 Code latency

The average and maximum code latencies for different size networks are shown in Figure 11. The code latency increases for both the consistency models as the network grows from single core to 64 cores. Average code latency for the release consistency

model in the 8x8 network is approximately 101.1 times of the single core, whereas for the weak consistency model it is 113.8 times respectively. The difference between the observed code latencies become obvious as the network size grows. It is due to the further overlapping and program order relaxation in the release consistency as compared to the weak consistency model.

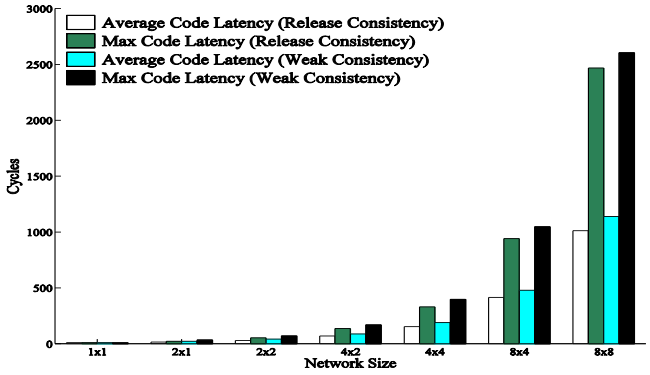

Figure 11. Code latency

6.2 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 size networks are shown in Figure 12. It increases as the network size increases for both the consistency models. Average consistency latency for the weak consistency model in the 8x8 network is approximately 8.2 times of the single core, whereas for the release consistency model it is 7 times. The difference between the observed consistency latencies becomes obvious in the large networks. It is due to the increasing overlapping, reordering and program order relaxation in the release consistency as compared to the weak consistency model.

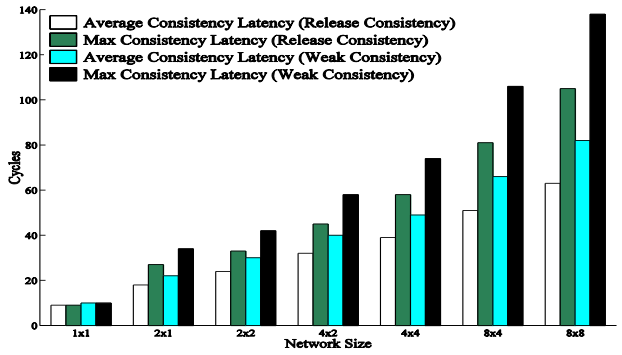

Figure 12. Consistency latency

6.3 Synchronization latency

The code latency comprises of synchronization and data latencies. Average and maximum synchronization latencies were compared for both the consistency models with increasing network size as given in Figure 13. Overall, there is no big difference in the synchronization latencies for both the consistency models as they use the same synchronization handler and network. The difference in the synchronization latencies is mainly due to the simultaneous issuance of synchronization requests in the release consistency model as initially TC2 is zero. While these requests are issued at different time in the weak consistency model as the completion time of the previous data operations in the non-critical section for all the nodes are different. Average synchronization latency for the weak consistency model in the 8x8 network is approximately 964 times of the single core, while for the release consistency model it is 1077 times. Average and maximum synchronization latencies increases exponentially for both the consistency models as the network size grows. The synchronization latency limits the system performance in large networks.

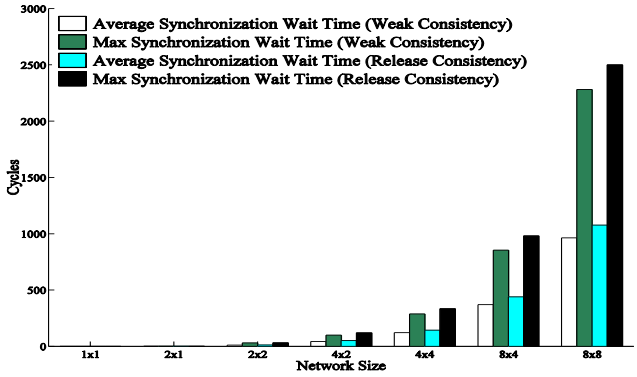

Figure 13. Synchronization latency

6.4 Data latency

The main performance gain of the release consistency model over the weak consistency model is the decrease in data latency. Average and maximum data latencies increases for both the consistency models with increasing network size as shown in Figure 14. For larger network the difference between the average and maximum data latencies become evident. Data latencies increase exponentially for both the consistency models with the increase in network size. Average data latency in the 8x8 network for the weak consistency model is approximately 17.11 times of the single core and for release consistency model it is 5.5 times. The difference in the average data latencies for both the consistency models in the 8x8 network is the highest (110 cycles). A large difference in the data latencies is observed in very large networks. The data latency decreases in the release consistency model as a result of reordering between non-critical section and next critical section of code.

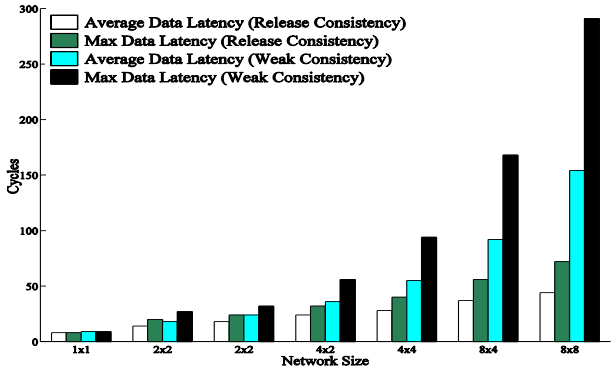

Figure 14. Data latency

7. CONCLUSION AND FUTURE WORK

We analyze the scalability of transaction counter based relaxed consistency models in the NoC based MPSoC platforms. We

observe that a single transaction counter can enforce the required global orders needed for the weak consistency model. Also, two transaction counters ensure the serial order enforcement needed for the release consistency model. Transaction counter based realization of the relaxed consistency models avoids the possible interference problem between the data and synchronization operations. In the experimental platforms, we consider a mesh network for the weak and release consistency models. All the nodes synchronized over the same lock in a particular node. Average and maximum code, synchronization and data latencies increase significantly for both weak and release consistency models as the network size scales. The experimental results show that the release consistency model scales nicely in comparison to the weak consistency model. The synchronization latency affects the efficiency of memory consistency in very large networks. In the future, we will study the implementation overhead and power analysis of various memory consistency models in the NoC based DSM systems.

8. ACKNOWLEDGMENTS

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

REFERENCES

[1] D. C. Pham, T. Aipperspach and D. Boerstler, “Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor”, IEEE J. Solid-State Circuits, 2006, 41, (1), pp. 179–196.

[2] S. Bell, B. Edwards and J. Amann, “TILE64TM processor: A 64-core SoC with Mesh Interconnect”. Digest of Technical Papers, IEEE Int. Solid-State Circuits Conf., February 2008, vol. 51, pp. 588–598.

[3] B. Stackhouse, B. Cherkauer and M. Gowan, “A 65-nm 2-billion-transistor quad-core Itanium processor”. Digest of Technical Papers, IEEE Int. Solid-State Circuits Conf, February 2008, vol.51, pp.592–598.

[4] L. Seiler, D. Carmean and E. Sprangle, et al: ‘Larrabee: a many core x86 architecture for visual computing’, ACM Trans. Graph., 2008, 27, (3), Article 18.

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

[6] W. J. Dally and B. Towles. Route packets, net wires: on-chip inteconnectoin networks. In DAC’01: Proceedings of the 38th Conference on Design Automation, pages 684–689, New York, NY, USA, 2001.

[7] S. V. Adve and K. Gharachorloo, “Shared Memory Consistency Models: A Tutorial”, IEEE Computer, Vol. 29 No. 12, pp. 66–76, Dec. 1996.

[8] J. Protic, I.Tartalja, “Memory consistency models for shared memory multiprocessors and DSM systems”, Melecon 96, 8th Mediterranean IEEE Electrotechnical Conference, vol.2, Page(s):1112-1115, May 1996.

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

[10] Robert C. Steinke and Garry J. Nutt, "A unified theory of shared memory consistency", Journal of the ACM, vol. 51, no. 5, pp. 800-849, 2004.

[11] Sarita V. Adve and Kourosh Hgarachorloo, Shared Memory Consistency Models: A Tutorial, Digital Western Research Laboratory, report no. 95/7, Palo Alto, California 94301 USA, September 1995.

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

[13] S.V.Adve, V.S.Pai and P.Ranganathan “Recent advances in memory consistency models for hardware shared memory systems”, Proceedings of the IEEE, Vol. 87, No.3, March 1999 Page(s):445–455.

[14] K. Gharachorloo. “Memory Consistency Models for Shared-Memory Multiprocessors”, PhD thesis, Stanford University, Dec. 1995.

[15] Axel Jantsch and Hannu Tenhunen, “Networks on Chip”, Kluwer Academic Publishers, 2003.

[16] Fayez Gebali, Haytham Elmiligi, Mohamed Watheq El-Kharashi, “Networks on Chip: Theory and Practice”. Taylor & Francis Group LLC-CRC Press, 2009.

[17] O. Villa, G. Palermo, C. Silvano, "Efficiency and Scalability of Barrier Synchronization on NoC Based Many-core Architectures". In Proceedings of CASES 2008- International Conference on Compilers, Architectures and Synthesis for Embedded Systems. Atlanta, Georgia, USA, October 2008, pp. 81-90.

[18] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie. Hardware- and software-based collective communication on the quadrics network. In Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA’01), page 24-35, Washington, DC, USA, 2001.

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

[20] E.J. Marinissen, B. Prince, D. Keltel-Schulz and Y. Zorian, “Challenges in embedded memory design and test”, Proceedings of Design, Automation and Test in Europe Conference (DATE’05), vol. 2, pp. 722-727, Mar. 2005.

[21] Yuan Xie, “Processor Architecture Design Using 3D Integration Technology”, In Proceedings of 23rd International Conference on VLSI Design (VLSID '10), Page(s):446–451, India, January 2010.

[22] S. S. Iyer, “Three Dimensional integration-memory applications”, In Proceedings of IEEE International SOI Conference, Page(s):1-5, USA, Oct. 2009.