Realization and Scalability of Release and Protected Release Consistency Models in NoC based Systems

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

***Abstract*—This paper studies the realization and scalability of release and protected release consistency models in Network-on-Chip (NoC) based Distributed Shared Memory (DSM) multi-core systems. The protected release consistency (PRC) model is proposed as an extension of the release consistency (RC) model and provides further relaxation in the shared memory operations. The realization schemes of RC and PRC models use a transaction counter in each node of the NoC based multi-core (McNoC) systems. Further, we study the scalability of these RC and PRC models and evaluate their performance in the McNoC platform. A configurable NoC based platform with 2D mesh topology and deflection routing algorithm is used in the tests. We experiment both with synthetic and application workloads. The performance of the RC and PRC models are compared using sequential consistency (SC) as the baseline. The experiments show that the average code execution time for the PRC model in 8x8 network (64 cores) is reduced by 30.5% over SC, and by 6.5% over RC model. Average data execution time in the 8x8 network for the PRC model is reduced by almost 37% over SC and by 8.8% over RC. The increase in area for the PRC of RC is about 880 gates in the network interface (1.7%).**

Keywords-Network-on-Chip; Distributed shared memory; Memory consistency; Protected release consistency; Scalability.

I. INTRODUCTION

Network-on-Chip (NoC) is a promising scalable solution [1-5] for future many-core Systems-on-Chip (SoCs). The shared memory can be distributed (DSM) on-chip to utilize high bandwidth of NoC based systems. In DSM systems, hardware optimizations (write buffer, cache, interconnection network) and software optimizations (compiler reordering, register allocation) can reorder the shared memory operations [6] and the system may give unexpected results. For the expected results, memory consistency decides the order in which shared memory operations will appear to execute up to the expectation of programmer. Different memory consistency models correspond to different ordering constraints they impose on the shared memory operations [6, 7, 8]. The sequential consistency (SC) model [9] does not take advantage of the system optimizations due to strict order enforcement on the shared memory operations. Consequently, several relaxed consistency models [6, 7, 8, 10-16] emerged by relaxing the ordering constraints on the shared memory operations and exploit these system optimizations.

Memory consistency models in general multiprocessors DSM systems are typically implemented as an extension of a conventional cache coherence protocol, by using a cache block as the atom of the consistency model. These solutions suffer of false sharing, massive communication overhead and scalability issue. *False sharing* occurs when multiple processors in the system write to the disjoint fields of the same coherence block and unnecessarily invalidate each other. The full block seems to be shared although only a part of it is in reality. Although this approach is elegant because it uses the same mechanism to address both cache coherency and memory consistency, an orthogonal approach is preferable when these two problems have very different requirements (e.g. on the size of the cache block and the consistency object), or when a cache is not used (e.g. for hard real time applications). In those cases a memory consistency implementation, that is independent of the cache coherence protocol, is required.

Moreover, the solutions for general multi-processors may not be applied to application specific NoC based multi-core systems because of different quantitative trade-offs and strict design constraints. Individual nodes in general multiprocessors consume more power and require more memory compared to the processing nodes in NoC based systems, where the on-chip memory bandwidth is very high, and latency and power are tightly constrained. Memory consistency issues have not yet been studied in-depth in the context of NoC based DSM multi-core systems. This paper attempts such a study with the following contributions:

- Proposal of *Protected Release Consistency* (PRC) model as an extension of RC model in NoC based DSM multi-core systems. The PRC model exploits the assumptions behind the well known Release Consistency (RC) model and allows additional relaxation on the order of memory transactions.
- Realization of the RC and PRC models using a *transaction counter* based approach. The transaction counter keeps track of the outstanding shared memory operations issued in each node of the network. This decoupled solution to the memory consistency does not consider cache/cache coherence.
- Scalability study of the RC and PRC models in a McNoC platform.

The configurable McNoC platform is used for the experiments. The platform uses DSM, distributed locks and on-chip 2D mesh Nostrum network [17] with deflection routing algorithm. The experimental results show the performance gain of the PRC model due to additional relaxation in the shared memory operations as compared to the RC and SC models in the McNoC systems.

The paper is organized as follows. The next section gives an overview of the related work. In section III, cache coherence and memory consistency is described. In section IV, RC and PRC models are discussed. In section V, the DSM based

McNoC platform is introduced. Realization of the RC and PRC models are presented in section VI. In section VII, simulation results and performance analysis of these consistency models in the McNoC systems are described and finally, section VIII summarizes our contribution.

II. Related Work

A. Memory Consistency in Multiprocessors DSM systems

A number of relaxed consistency models have been proposed in the literature [6-8, 10-16] for high performance multiprocessor systems. Adve et al. [6] discussed memory consistency models with an emphasis on the system optimizations they allow. They proposed the counter based mechanism to realize the weak consistency model; however, they did not discuss the realization of the RC model. The *weak consistency* (weak ordering) model [10] classifies the shared memory operations as data and synchronization operations. The data operations between the two consecutive synchronization points can be reordered with respect to each other. The *Release Consistency* (RC) model [11] further refines the weak ordering model and allows more reordering of the shared memory operations. RC model further classifies the synchronization operations as acquire and release operations. The DASH multiprocessor [12] implements RC model. There exit different variants of the RC model which claim further reordering and relaxations in the shared memory operations as compared to the RC model. For instance, in the *eager RC* model [13] the processor delays the propagation of all its modifications to the shared data until the release point. The eager RC model hides memory access time by sending updates belonging to the same destination in a single message at release point and reduces the number of messages as compared to the RC model. In the *lazy release consistency* (LRC) model [14], propagations of modifications are further delayed to the acquisition of a lock by another processor. The acquiring processor observes the needed modifications at acquire time. *Entry RC* [15] further classifies the lock acquire into two different modes (exclusive and non-exclusive). Once a lock is acquired in an exclusive mode, the next non-exclusive acquire of that lock, performed by any other processor, is allowed to perform only after the lock owner performs the release. The lock can be acquired by more than one processor in the non-exclusive mode if they only perform read operations in their critical sections. Furthermore, to simplify the implementation of entry consistency, *Scope consistency* [16] defines a scope containing all critical sections guarded by the same lock. Then global ordering is enforced only within a scope. Scope consistency reduces the false sharing as compared to the LRC and uses the page as the coherence granularity, in contrast to entry RC, which uses a variable as coherence atom.

B. Memory Consistency in NoC based Multicores systems

Some work [18, 19] has been done on the memory consistency issue in the context of NoC based multi-core systems. Petrot et al. [18] explored the reordering of the 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 the initiator should wait for the response from 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 interferences are avoided without suffering the program order relaxation. Streaming consistency (StrC) [19] targets systems that run streaming applications and share data through circular buffers in shared memory between multiple producers and consumers. StrC is different from RC where synchronization sections can overlap. However, StrC does not allow implicit synchronization and every write or read to the shared memory must take place inside a synchronization section and not outside of it. None of these solutions [18, 19] in the context of NoC based multi-core systems have used a transaction counter (TC) based approach for the realization of the memory consistency model. In [20-22] SC model and TC based realization of WC and RC models are discussed in the NoC based systems. In [21], the realization of RC model using two TCs is discussed in the McNoC systems. TC1, TC2 keeps track of the outstanding data operations in non-critical section and critical section respectively. However, The TC2 is unnecessarily checked at acquire point to be zero which is already checked at previous release point. Also, this work uses two TCs to realize the RC model instead of a single TC in each node of the network. In this paper, we propose PRC model as an extension of RC model and realize these consistency models using single TC in each node of the network and we argue it is effective and cheap to implement.

III. Cache Coherence and Memory consistency

Multiprocessors systems using caches with write-through or write-back policy have a coherence problem due to different cached copies of the same shared data. The coherence protocol solves this issue. The snooping based cache coherence protocol is impractical in the NoCs because broadcast is expensive in the network. The directory based coherence protocol [23] can be used with distributed directories in nodes to maintain the status information of cache line, which however, generates a significant amount of coherence traffic. For instance, we consider write-back caches with write-allocate and invalidation based MSI coherence protocol with requesting, directory, dirty and sharer nodes in the network. The *requesting node* issues the request, the *directory node* hosts the main memory of the block, and the *dirty node* has the modified cached copy of the block. A read miss in the requesting node generates a read request to the dirty node when the main memory has a dirty copy in the directory node. The dirty node flushes the data, updates the cache of the requesting node (cache-to-cache transfer), and updates the main memory in the directory node. Similarly, on a write miss the additional coherence traffic is triggered. Increasing the coherence granularity (cache line size) even worsens the situation and introduces the *false sharing* problem. The false sharing (in addition to true sharing) further degrades the system performance due to unnecessary invalidations by updating disjoint fields by different processors in the same large block. Significant unnecessary communication caused by false sharing limits the scalability of

DSM systems. Consequently, we argue that decoupling of cache coherence and memory consistency schemes allow for optimizing them separately. Moreover, in systems without a cache (hard real-time systems often avoid caches altogether) a memory consistency support is required that is independent of the cache management. Hence, we propose a *Transaction Counter* (TC) based approach that is independent of a coherence protocol.

IV. Release Consistency and Protected Release Consistency Models

A. *Release Consistency Model*

The RC model proposed by Gharachorloo et al. [11] is a refinement of the weak ordering [10, 22]. The RC model further distinguishes synchronization operations as *acquire* and *release* operations. The purpose of the acquire operation is to delay the future data operations until the lock is obtained. The acquire operation has not to wait for the completion of previous data operations. The purpose of the release operation is to pass information about the completion of previously issued data operations. The release operation does not delay future data operations.

The diagram in Figure 1 (adopted from [8, 19]) compares the ordering constraints under three memory consistency models. The SC model enforces strict order on the shared memory operations and they are completed in the order specified by the program (program order). The sequential order is maintained by interleaving operations on lock S among processors in the system. The processor consistency (PC) model eliminates the ordering constraints between the write on C followed by a read to D. We only focus on the ordering constraints under RC (Figure 1(c)). For convenience we categorize the data (read, write) operations to the shared memory locations as *unprotected* and *protected* data operations. The data operations on variables (A, B, E, F) are unprotected data operations. They are not protected under acquire-release operations on the lock S. The data operations on the variables (C, D) are protected data operations under acquire-release operations on lock S.

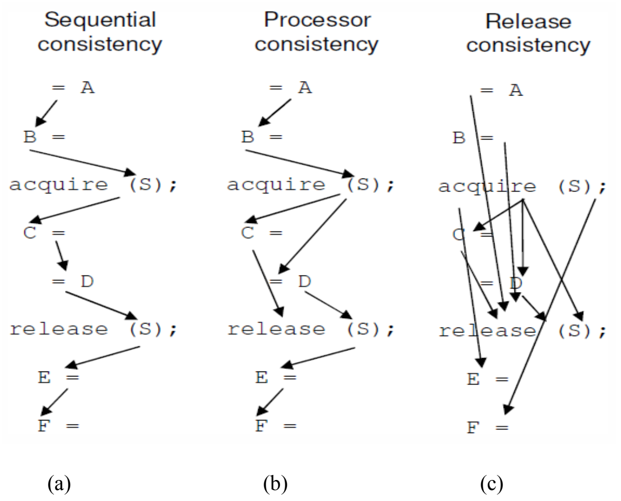

Figure 1. Comparision of three different consistency models

We can make the following observations:

- The independent unprotected data operations (A, B) can be overlapped with each other as they are to different locations in the shared memory.
- The unprotected data operations (A, B) and the independent acquire operations on lock S can be overlapped and reordered.
- After the acquisition of lock S the unprotected data operations (A, B) and protected data operations (C, D) can be reordered up to the point of release operation on lock S.
- The acquire operation on lock S must be performed before the issuance of the following *protected* data operations (C, D). The lock *must* be gained before entering to the protected section. The reordering between acquire operation on lock S to the following protected data operations (C, D) is illegal and may change the system behavior.
- The independent protected data operations (C, D) can be overlapped with each other as they are to different locations in the shared memory.
- Before the issuance of release operation on lock S, the unprotected data operations (A, B) and protected data operations (C, D) must be completed. The reordering between the preceding protected data operations (C, D) and the release operation on lock S is also illegal and may change the system behavior. The lock S *must* not be released before the protected section execution is completed.
- The acquire operation on S must be completed before the release operation on the same lock S.
- The release operation on the same lock S must be completed before the next acquire operation on S.

To sum up, according to the RC model, an acquire operation must be performed before the issuance of the following (protected data, release) operation. Before the issuance of a release operation all independent data operations (unprotected, protected) can be reordered and must be completed at release point. The RC model enforces the global orders on the shared memory operations as given in Figure 2.

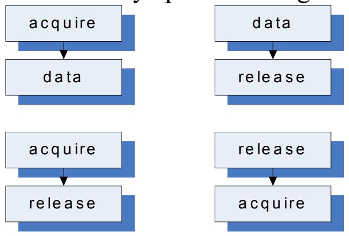

Figure 2. Global orders to enforce under RC model

B. *Protected Release Consistency Model*

Referring again to Figure 1(c), the RC model does not allow the following reordering which would not affect the semantics of a concurrent program:

- Unprotected data operations (A, B) → release operation on S.

- Unprotected data operations (A, B) → *next* unprotected data operations (E, F).

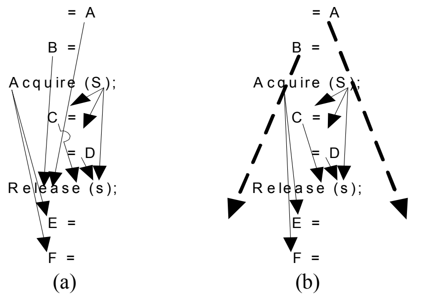

Figure 3. a) RC model b) PRC model

Figure 3(a) reconsiders the RC model given in Figure 1(c). As an extension of the RC model, the PRC model removes the unnecessary ordering constraints (Bold dashed-line arrows) between the unprotected data operation (A, B) and release operation on lock S in Figure 3(b). The PRC model allows further reordering and relaxations in the shared memory operations as compared to the RC model. We give the following reasoning and analysis:

- The RC model in Figure 3(a) allows reordering/overlapping/relaxation between the unprotected data operations (A, B) and independent acquire operations on lock S. As acquire and release operations are to the same lock S, so the unprotected data operations (A, B) and release operation on lock S are also independent and can be overlapped as well. From a multiprocessor point of view, the data dependencies among processors are *only* respected via protected data operations (C, D) and synchronization acquire-release operations on lock S. The purpose of the release operation on S is to notify the completion of the prior protected data operations (C, D). It has nothing in common with prior unprotected data operations (A, B) and there is no reason to delay the issuance of the release operation on S to notify the completion of the preceded unprotected data operations (A, B).
- The unprotected data operations (A, B) can be overlapped with the next independent unprotected operations (E, F) as they are to different locations.

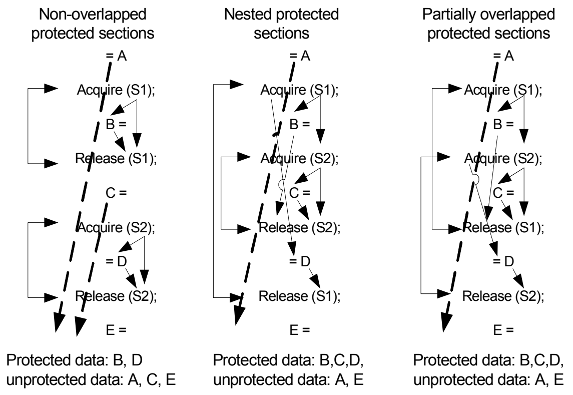

Figure 4. Further cases under PRC model

In Figure 4 we consider further cases and code samples with non-overlapped, nested and partially overlapped protected sections to show the ordering requirements under PRC model. The PRC model enforces the global order on shared memory operations as given in Figure 5 with the following rules:

a) The acquire operation on a lock must be performed before the issuance of the following (*protected* data, release) operations.
b) Before issuance of a release operation to a lock the independent unprotected and protected data operations can be reordered with each other and **only** the prior outstanding protected data operations must be completed.
c) Acquire and release operations on a lock must be sequentially consistent with respect to each other.

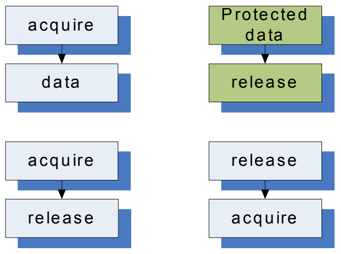

Figure 5. Global orders to enforce under PRC model

V. DSM Based McNoC Platform

A homogenous NoC based multi-core system is given in Figure 6(a) with 16 nodes. All the nodes are interconnected via a packet-switched network. Each node represents a typical processor-memory (PM) node in the platform. Each PM node consists of a processor, transaction controller (TCTRL), synchronization handler, network interface (NI), and the local memory as shown in Figure 6(b). TCTRL processes the transactions issued by the processor and acts as interface between processor and the rest of the system. It uses a transaction counter to realize RC and PRC models in the McNoC platform. The SC model is realized by stalling the processor on issuance of each memory operation till the completion of the previous operation [22]. The platform uses distributed locks maintained in the synchronization handlers. The NI performs packetization, de-packetization, queuing, and message passing and connects a PM node to the NoC. The platform uses Nostrum [17] as on-chip packet switched network with 2D mesh topology and deflection routing algorithm. The local memory is connected to the local processor and NI, respectively. All shared parts in the local memories form the virtual memory, which is organized as a DSM and uses a single global memory address space. Two addressing schemes are adopted for the shared memory access and a virtual-to-physical (VTP) address translation is required. Local shared memory operations (read, write) and lock (acquire, release) operations are accomplished within the node. For the remote shared memory and lock accesses, message passing is carried out to the remote node via the network.

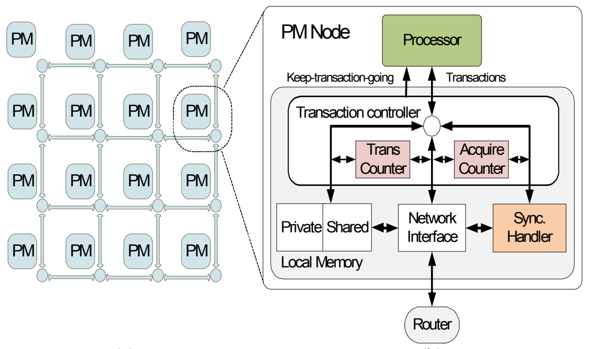

Figure 6. a) Homogeneous McNoC b) PM node

The McNoC platform provides the memory synchronization support both at the software and hardware level. The platform supports busy-wait (block lock) synchronization primitive. The synchronization handler (SH) provides underlying hardware support for memory synchronization. The SH in each node controls synchronization variables (N locks) maintained in the global address space. Every lock is accessed in a sequential order by multiple processors in the system. The lock status transition diagram is shown in Figure 7. A lock can be either in a *locked* or *unlocked* status. The synchronization (acquire, release) requests to the SH come either from the local processor or from a remote processor via the network. If the requested lock's status is *unlocked*, then the lock acquire request changes the lock status to *locked* and successful lock acquire acknowledgement is sent back to the acquiring node. If the requested lock status is *locked*, a failed lock acquire acknowledgement is sent back to the originating node. The source node sends again the same request until the lock is gained. A release request changes the lock's status to *unlocked*.

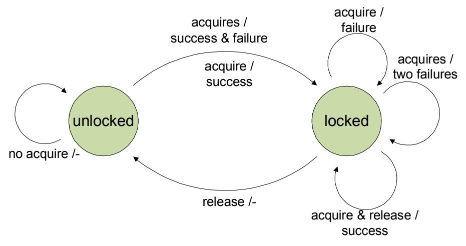

Figure 7. Lock status transition diagram

VI. Realization of Release Consistency and Protected Release Consistency Models

A. Release Consistency Model

The RC model can be realized by enforcing the required global orders on the shared memory operations given in Figure 2.

- **Release → acquire:** This global order is enforced by sequential order on a lock among multiprocessors in the system (refer to discussion on lock status in previous section). The release operation on a lock is completed before the next acquire operation on it.
- **Acquire → data/release:** These global orders can be enforced by stalling the processor upon the issuance of a lock acquire until the successful acquisition of lock.
- **Data → release:** To enforce this global order, a transaction counter (TC_D) is used in each node of the platform to keep track of the outstanding data operations issued before the release operation. The TC_D is only affected by data operations. The TC_D is checked at the release point and issuance of a release operation is delayed (by stalling processor) until TC_D is zero, i.e., completion of previously issued outstanding data operations.

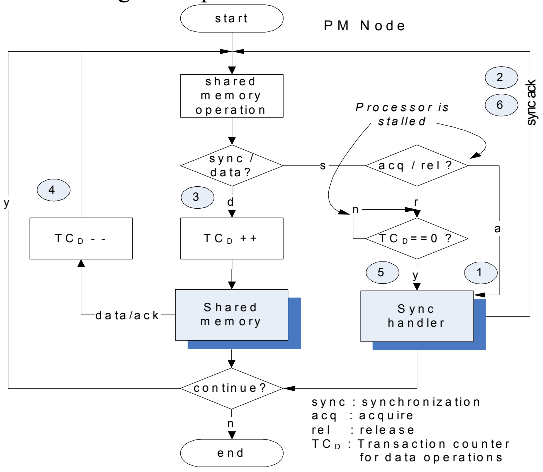

Figure 8. Realization scheme of RC model

The realization scheme of RC model is given in Figure 8. The acquire operations (1) are issued to the locks in the synchronization handler (SH) and are completed by the acquire acknowledgment (2). After issuance of an acquire operation, the processor is stalled and the following operations are delayed until the successful acquisition of the lock. The data operations (unprotected, protected) (3) are issued to the shared memory locations and completed by either data return or write acknowledgment (4). The issuance and completion of the data operations increment and decrement the TC_D. Acquire and release operations (1, 2, 5, 6) neither increment nor decrement the TC_D. The issuance of release operations (5) to the SH are delayed until the TC_D becomes zero in the local node, i.e., completion of previously issued outstanding data operations (unprotected, protected). The release operations are completed by the release acknowledgment (6). The data operations to the remote node via the network also affect the TC_D in the local node. The transaction controller (TCTRL) for the RC model in Figure 6, asserts the keep-transaction-going (KTG) output signal low upon reception of a lock acquire operation to stall the processor. Also, the processor is stalled (low KTG) at the release point until "$TC_D = 0$".

B. Protected Release Consistency Model

The PRC model is realized by enforcing the required global orders on the shared memory operations as given in Figure 5. The ordering requirements of both PRC and RC models are similar except the release operation in the PRC model notifies only the preceding protected data operations:

- **Protected data → Release:** To enforce this global order, a transaction counter (TC_{PD}) is used in each node to keep track of the outstanding protected data operations only. The TC_{PD} is only affected by protected data operations and *not by unprotected data operations*. The TC_{PD} is checked at the release point and the issuance of release operation is delayed (by stalling processor) until TC_{PD} to be zero.

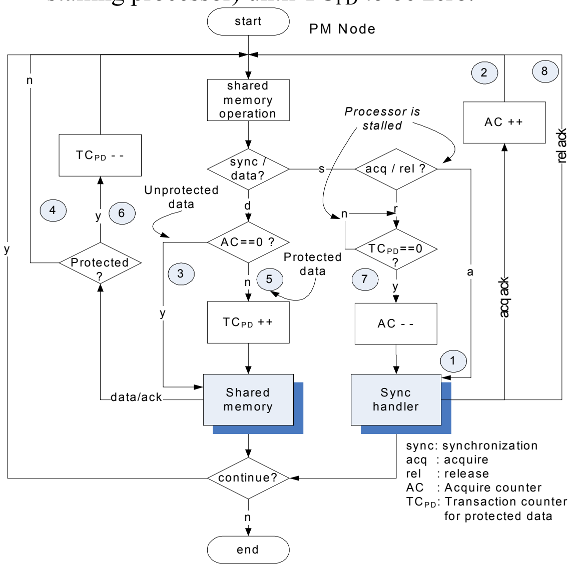

Figure 9. Realization scheme of PRC model

The data operations are classified as unprotected and protected data operations for the realization of PRC model using an acquire-counter (AC) in each node as illustrated in Figure 6. Initially the AC is zero. The AC is incremented by acquisition of a lock and decremented by releasing it. When a data operation is issued, the AC is checked. If it is zero, the data operation is unprotected; otherwise it is protected. The Figure 9 illustrates the realization scheme of PRC model. After the issuance of an acquire operation (1) processor is stalled and the following operations (data, release) are delayed till the successful acquisition of lock (2). The AC is incremented by a lock acquisition (2) and decremented by the issuance of a release operation (7). The issuance of a data operation checks AC, if "AC = 0", then it is unprotected data operations (3) otherwise it is protected data operations (5). The TC_{PD} in each node is incremented with the issuance of the protected data operations (5). It is decremented by the completion of previously issued outstanding protected data operations (6). *The outstanding unprotected data operations (3, 4) are not tracked, and do not affect TC_{PD}*. Also, the TC_{PD} is not affected by acquire, release operations. The issuance of release operations (7) check the TC_{PD} and are delayed until TC_{PD} becomes zero, i.e., completion of previously issued outstanding protected data operations. The release operations are completed by the release acknowledgment (8). The protected data operations to the remote node via network also affect the TC_{PD} in the local node. The transaction controller （TCTRL) for the PRC model in Figure 6, asserts the KTG output low to the processor on reception of a lock acquire operation to stall the processor until the lock acquisition. Also, the processor is stalled (low KTG) at the release point till the completion of previously issued outstanding protected data operations (TC_{PD} = 0).

VII. Experiments and Results

A. *Area/Hardware cost*

Designs are synthesized using Synopsis Design Compiler with SMIC 90nm technology. The synthesis results in term of nand-gate equivalent and maximum frequency are given in Table I. The area/hardware cost of the design with PRC model is very small and negligible as compared to the RC and SC models. The area cost of the PRC model is increased by 1.7% and 7.45% over RC and SC models respectively.

TABLE I. Synthesis Results

	SC Model		**RC Model**		**PRC Model**	
	Area	**Freq**	**Area**	**Freq**	**Area**	**Freq**
Switch	13.24	0.5	13.24	0.5	12.96	0.5
SH	4.14	1.0	3.76	1.25	3.76	1.25
TCTRL	0.59	1.6	1.17	1.6	1.75	1.6
NI	46.20	1.25	49.99	1.25	50.87	1.25
Total Area	**64.17**		**68.16**		**69.34**	

Area (Kilo Nand Gates), Freq: Frequency (GHz)

B. *Experimental Setup*

We constructed a multi-core NoC based simulation platform in VHDL for the experiments. The size of the platform is configurable. The platform uses the LEON3 processor [24] in each PM node as given in the Figure 6. The LEON3 processor core is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The platform uses distributed shared memories and the size of the shared memory in local memory of each node is 16 MB. The platform also uses distributed locks and the synchronization handler in each node maintains 256 locks in the shared address space. The TC_D /TC_{PD} and AC each of 32 bits are used in the TCTRL to realize the memory consistency models. The buffering capacity at the network interface (NI) is 64 packets. The packet formation at NI complies with the network protocol (97 bits, 7 fields). The Nostrum on-chip packet-switched network use 2D regular mesh topology and a deflection routing policy.

C. *Experiment with Synthetic Workload*

We evaluate the behavior of the RC, PRC and SC models with a set of synthetic workloads manually mapped on LEON3 processors in the network. The effects of network size on the code, synchronization, lock acquire, release and data execution times are investigated. The same sequence of transactions is generated by the LEON3 processor executing C-code in each node, which means that the actual amount of processing work done is scaled with the network size. Thus, if the programs in the individual nodes would be independent and only operating on local and private data, the execution time would be identical for all network sizes. Thus, in the experiments we measure the effects of synchronization and inter-node dependencies.

The code sequences have unprotected and protected data transactions and synchronization operations. For sufficient test coverage, we tested with various numbers of locks and data operations in the program. We also tested with different numbers and locations of locks in the network. Different traffic patterns are generated for the unprotected data transactions. For lock transactions and protected data operations hotspot traffic pattern is generated.

D. Results and Discussion

The code execution time (CET) is the average execution time of concurrent code mapped on the processors in the McNoC system. The average and maximum CETs are compared for the RC, PRC and SC models as given in Figure 10. The increase in CET is only due to synchronization and data dependency effects. The SC model constitutes the worst case against which the other models are compared.

We observe that the PRC achieves fairly good CET reduction. When the system size increases from 1 to 64 cores, the CET quickly increases for all three consistency models due to mutual dependencies and growing communication time. But the difference between the observed CET becomes obvious as the network size scales up. For instance, the average CET for the PRC model in the 8x8 network is reduced by 30.5% over SC, and 6.5% over RC. The performance gap is wider in the larger networks due to further overlapping and program order relaxation in the PRC model.

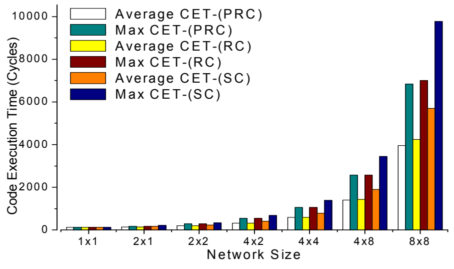

Figure 10. Code Execution Time

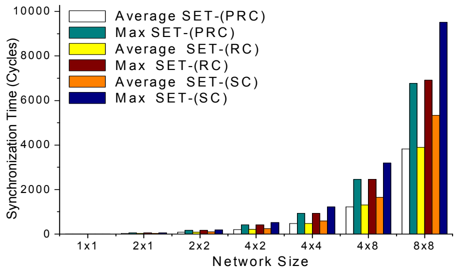

Figure 11. Synchronization Execution Time

The synchronization execution time (SET) is the average execution time of the lock acquire and release primitives. The average and maximum SETs are given in Figure 11. As the system size grows, the synchronization among the cores contributes more to the CET. This is due to the growing lock acquire time and communication latency with increasing system size. The SET further consists of lock acquire and release execution times. The lock acquire execution time (LAET) is the average execution time of the successful lock acquire operations to gain the lock in the system. The lock release execution time (LRET) is the average execution time of the lock release operations to release the lock in the system. The average and maximum LAETs is shown in Figures 12. As observed in the results, the LAET is dominant and it increases exponentially as the network scales up. It is due to the increasing network traffic, delay and waiting time for acquiring the lock. The average LAET for the PRC model in the 8x8 network is reduced by 28.5% compared to SC and by 2% compared to RC. The LAET of RC is 27% lower than that of SC. On the other hand, the LRET is modest as compared to the LAET as it does not involve the waiting time while releasing a lock. The average LRET is nearly the same for all the three consistency models.

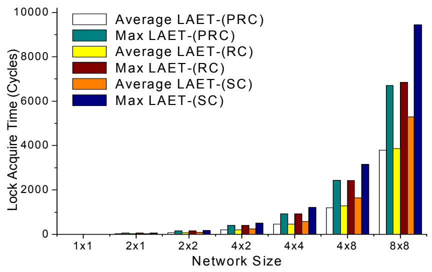

Figure 12. Lock Acquire Execution Time

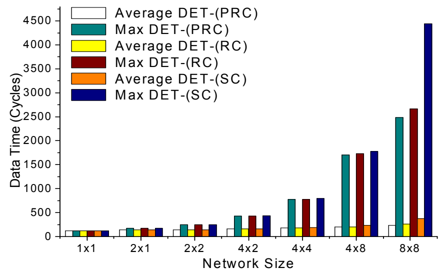

Figure 13. Data Execution Time

The data execution time (DET) is the average execution time of the load and store operations in the concurrent codes in the system. The PRC model decreases the average and maximum DETs by allowing further outstanding data transactions as compared to the RC and SC models. Average and maximum DETs increase for all the three consistency models with increasing network size as shown in Figure 13. Average DET in the 8x8 network for PRC model is reduced by almost 37% over SC and by 8.8% over RC. The DET of RC is 31% lower than that of SC. DET decreases in the PRC model as a result of additional reordering and overlapping in the data operations. Average execution times relative to SC model in percentage are given in Table II.

TABLE II. AVERAGE EXECUTION TIMES RELATIVE TO SC IN %

	SC Model	RC Model	PRC Model
4x4 CET	100	76	76
8x8 CET	100	74.3	69.5
4x4 SET	100	79.2	79.2
8x8 SET	100	73	71.5
4x4 LAET	100	79.2	79.2
8x8 LAET	100	73	71.5
4x4 LRET	100	100	100

8x8 LRET	100	100	100
4x4 DET	100	94.2	94.2
8x8 DET	100	69	63

E. Experiment with Application Workload

As an application workload a matrix multiplication is mapped manually on the LEON3 processors. The matrix multiplication calculates the product of two matrices, A[64x1] and B[1x64], resulting in a C[64x64]. Three matrices are distributed and stored in the shared memory. All three matrices are decomposed into sub-matrices which are manually mapped on the different cores in the system. The number of sub-matrices is equal to the number of nodes in the multi-core system. The matrix multiplication is data intensive application and does not use the synchronization operations. Therefore, the execution time for both RC and PRC models are the same, but, less than the strict SC due to overlapping in the data operations. The multi-core NoC achieves fairly good speedup, as the system size scales up. The speedup is the ratio of single core execution time and the execution time of multi-cores. As shown in Figure 14, the speedup for the matrix multiplication on the 8x8 system for PRC and RC models is 53.92 (very close to the ideal speedup of 64), while for SC model is 42.91.

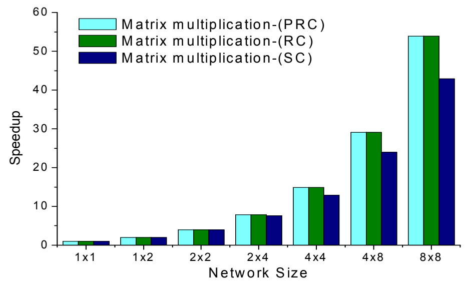

Figure 14. Matrix Multiplication

VIII. Conclusion

We studied realization of the RC and PRC models using a transaction counter based approach in NoC based DSM multi-core systems. The PRC model is proposed as an extension of RC that provides further relaxation in the shared memory operations. The additional relaxation in PRC model does not violate the semantics of properly synchronized concurrent programs and ensures program correctness. We also studied the scalability of RC, PRC and SC models and compared their performance using SC as the baseline. The configurable McNoC platform with 2D mesh topology and deflection routing was used in the tests. We experimented both with synthetic and application workloads. The block lock synchronization primitives are used in tests. The experimental results show that, the PRC model performs consistently better on average than both RC and SC as the system scales up from single core to 64 cores. The average code execution time and data execution time for the PRC model in the 8x8 network is about reduced by 30.5% and 37% over SC, 6.5% and 8.8% over RC respectively.

References

[1] 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.

[2] A. Jantsch, H. Tenhunen, "Networks on Chip," Kluwer Academic Publishers, 2003.

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

[4] T. Bjerregaard, S. Mahadevan, "A survey of research and practices of network-on-chip," ACM Computing Surveys, 38(1): 1–51, March 2006.

[5] J. D. Owens, W. J. Dally, "Research challenges for on-chip interconnection networks," IEEE MICRO, 27(5): 96–108, Oct. 2007.

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

[7] David E. Culler et al. "Parallel Computer Architecture: A Hardware/Software Approach", Morgan Kaufmann Publishers,1999.

[8] J. Hennesy et al. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, San Francisco, 2nd edition, 2003.

[9] 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.

[10] 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.

[11] K. Gharachorloo et al. "Memory consistency and event ordering in scalable shared-memory multiprocessors," Computer Architecture News, 18(2): 15-26, June 1990.

[12] D. Lenoski et al. "The Stanford Dash Multiprocessor", Computer, 87(3), March 1992 pages: 418- 429.

[13] J. B. Carter, J. K. Bennett , and W. Zwaenepoel. " Implementation and performance of Munin" in: Proc. of 13th ACM symposium on Operating systems principles (SOSP '91), pages 152–164, October 1991.

[14] P. Keleher, A.L. Cox, and W. Zwaenepoel. "Lazy Consistency for Software Distributed Shared Memory," In Proc of the 19th Annual Symposium on Computeter Architecture, pages 13–21, May 1992.

[15] B.N. Bershad et al. "The Midway Distributed Shared Memory System" In: Proc. of IEEE COMPCON '93 Conference, pp. 528–537, Feb 1993.

[16] L. Iftode, J. P. Singh, and K. Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. In Proc. of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, June 1996.

[17] A. Jantsch "The Nostrum network-on-chip", in http://www.ict.kth.se/nostrum.

[18] 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.

[19] J.W. van den Brand and M. Bekooij. "Streaming consistency: a model for efficient MPSoC design," In Proceedings of 10th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools (DSD2007), Germany, August 2007.

[20] A. Naeem, X. Chen, Z. Lu, and A. Jantsch, "Scalability of Weak Consistency in NoC based Multicore Architectures," in Proc. of the ISCAS 2010, pp. 3497-3500, Paris, France, June 2010.

[21] A. Naeem et al. "Scalability of Relaxed Consistency Models in NoC based Multicore Architectures," ACM SIGARCH Computer Architecture News, 37(5): 8-15, December 2009.

[22] A. Naeem, X. Chen, Z. Lu, and A. Jantsch. "Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems," In Proceedings of the 16th ASP-DAC, pp. 154-159, Yokohama, Japan, January 2011.

[23] L. M. Censier et al. A new solution to coherence problems in multicache systems. IEEE Trans. on Computer, c- 27(12):1112–1118, Nov. 1978.

[24] "Leon3 processor," in http://www.gaisler.com