Architecture Support and Comparison of Three Memory Consistency Models in NoC based Systems

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

***Abstract*—We propose a novel hardware support for three relaxed memory models, Release Consistency (RC), Partial Store Ordering (PSO) and Total Store Ordering (TSO) in Network-on-Chip (NoC) based distributed shared memory multicore systems. The RC model is realized by using a *Transaction Counter* and an *Address Stack* based approach to enforce the required global orders on the shared memory operations. The PSO and TSO models are realized by using a *Write Transaction Counter* and a *Write Address Stack* based approach to enforce the required global orders on the shared memory operations. In the experiments, we use a configurable platform based on a 2D mesh NoC using deflection routing policy. The results show that under synthetic workloads, the average execution time for the RC, PSO and TSO models in 8x8 network (64 cores) is reduced by 35.8%, 22.7% and 16.5% over the sequential consistency (SC) model, respectively. The average speedup for the RC, PSO and TSO models in 8x8 network under different application workloads is increased by 34.3%, 10.6% and 8.9% over the SC model, respectively. The area cost for the TSO, PSO and RC models is increased by less than 2% over the SC model at the interface to the processor.**

Keywords- Memory consistency; Release consistency; Scalability; Distributed shared memory; Network-on-Chip

I. Introduction

The parallelization of computation, communication and memory architecture has to be matched [1]. The full potential can be harvested with Distributed Shared Memory (DSM) on-chip by exploiting the distributed nature of Network-on-Chip (NoC) based systems. Since shared memory operations can be reordered in the network, the DSM systems may show unexpected behavior. A memory consistency model defines the execution order of the shared memory operations for the expected behavior of the DSM systems [2]. The strict *Sequential Consistency* (SC) model [3] does not take advantage of potential performance benefits in the DSM systems. As a result, several *relaxed* consistency models [2][4][9][11] emerged to exploit the system optimizations by relaxing the ordering constraints on the shared memory operations.

Memory consistency and cache coherence are two distinct problems. Both aim to achieve consistent view of the memory system but at different levels. The cache coherence problem arises due to different cached copies of the same shared data. Memory consistency in contrast is related to the ordering constraints on the shared memory operations for the correct behavior of the DSM systems. In some situations, where 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) an independent implementation of the memory consistency and cache coherence is preferred [1][23-25].

Furthermore, heterogeneous and customized systems have different design constraints and requirements than general multiprocessors systems. The former have tighter power constraints, require less but heterogeneous memory, make less or no use of caches, and have often soft or hard real-time constraints. In the context of customized NoC based multicore (McNoC) systems, this paper studies the memory consistency issue with the following contributions:

- A novel realization scheme of the Release Consistency (RC) model which is independent of the cache coherence protocol is proposed. In contrast to [25], reordering among the outstanding shared memory operations issued by a processor to the *same* location in the memory is avoided, which is mandatory to ensure the parallel program correctness, i.e., the read operation must always fetch the most recent value from the memory location in the adaptive NoC based DSM systems. To that end, an additional hardware structure called address stack *(A-Stack*) is used in each node of the network (Section V).
- The Total and Partial Store Ordering (TSO, PSO) models are realized using Write Transaction Counter (*WTC)* and Write Address Stack (*WA-Stack*) in each node of the network. The *WTC* is used to keep track of the outstanding shared memory write operations. The *WA-Stack* is used to constrain the outstanding write operations issued to the *same* location in the shared memory.
- Performance of the RC, PSO, TSO and SC models is evaluated in the NoC based systems with 1 to 64 cores.

For the experiments, a configurable McNoC platform is used with DSM, distributed locks and on-chip 2D mesh *Nostrum* network [5] using deflection routing policy. We compare the performance of these memory models in the McNoC systems. The experimental results show the performance gain of the RC, PSO and TSO models due to the reordering and relaxation in the shared memory operations compared to the SC model.

The paper organization is as follows. The next section gives an overview of the related work. In section III, the TSO, PSO and RC models are discussed. In section IV, the DSM based McNoC platform is introduced. The realization schemes of the three consistency models are presented in section V. In section VI, simulation results and performance analysis of these consistency models are described in the McNoC systems and finally, section VII summarizes our contribution.

II. Related Work

A. Memory Consistency in Multiprocessors DSM systems

Several memory consistency models are discussed in the literature [2][3][4][9]. The SC model [3] enforces total order on the memory operations. The TSO model [6][7] relaxes the ordering constraint in the case of a *write followed by a read*

978-0-7695-4798-5/12 $26.00 © 2012 IEEE
DOI 10.1109/DSD.2012.27

operation. The PSO model [6] further provides relaxation among the *write* operations. The ordering constraints on the memory operations under both TSO and PSO models are enforced using different kind of *fences* (non-memory references). For instance, the SPARC architectures [6] use (MEMBARs, SBAR), and the x86 architectures [7] use (MFENCE, SFENCE, LFENCE) fences.

Adve et al. [2] proposed a counter based mechanism to realize the weak consistency model; however, they did not discuss the realization of the RC model. The weak consistency model [8] classifies the shared memory operations as *data* and *synchronization* operations. The data (read, write) operations issued by a processor between two consecutive synchronization operations can be reordered with respect to each other. The RC model [9] further classifies the synchronization operations as *acquire* and *release* operations. The DASH project [10] implements the RC model using tracking mechanism via several counters and is dependent on the cache coherence protocol. The directory based coherence protocol is used to maintain the status information of cache blocks.

Recent work [12][13] on the directory based coherence protocols aims to reduce the directory overheads in terms of area, energy and power consumption. However, the directory based coherence protocols have some limitations in the larger networks due to extra coherence traffic, directory overhead, additional latencies and complexities. Token Coherence [14] decouples the performance and correctness of the coherence protocols. Tokens are associated with each memory block which traverses in the system and are used to track the correct transfer and accesses to that block. The performance protocol (*TokenB*) is based on broadcasting transient requests which is not scalable in the larger networks.

Transactional memories aim to improve programmer productivity by moving the synchronization burden to the platform. The hardware approach [15] relies on additional transactional caches and coherence protocols. The transaction size is bounded by the set size of the transactional caches. The software approach has no such restriction and relies on the runtime data structures but is less efficient. A hybrid approach [16] combines the benefits of both. To ensure the consistent behavior of the memory system aborted transactions due to the conflicts are re-executed. Memory models are also explored at high-level programming languages. For example, the Java memory model [17] specifies the legal transformations and optimizations for the compiler and virtual machine/hardware.

Of late, address translation aware memory consistency models at physical and virtual address levels (PAMC, VAMC) have been proposed [18]. To enforce total order on all operations, the address translation and translation coherence is proposed. They focus on the detection of the design and runtime faults due to the address translation. In [19], a memory model is defined in terms of instruction reordering and store atomicity. The main emphasis of the work is on the serializability and store atomicity issues.

B. Memory Consistency in NoC based Multicores systems

In NoC based systems, the proposed mechanism in [20] allows one outstanding transaction of an initiator at a time in the network. But, this strict ordering could not utilize the parallel nature of the network. A protocol stack for on-chip interconnects is proposed in [21]. The work specifies a stack at different levels of the SoC design. They briefly outline the mechanisms to implement the RC model at the memory-mapped stack. However, they do not discuss the implementation detail of it. Streaming consistency [22] is based on the software cache coherence protocol. In contrast to the RC model the synchronization sections can overtake each other. However, polling and updating the circular buffer administration at each request level may not be feasible in the larger systems.

The *Transaction Counter (TC)* based hardware approach is adopted in [1][23-25] to realize the memory models independent of the coherence protocols in the McNoC systems. The SC model is realized in the McNoC systems by stalling the processor on issuance of each memory operation till its completion [23]. In [1][24], the RC model is realized by using two *TCs* in each node of the network. *TC1* and *TC2* are used to keep track of the outstanding shared data operations issued in the non-critical and critical sections, respectively. However, *TC2* is unnecessarily checked at the acquire points to be zero in [24]; it is already checked at previous release points. In [25], a single *TC* based approach is adopted to realize the RC model in the McNoC systems. But, these works have not used a mechanism to avoid reordering among the outstanding shared memory operations issued by a processor to the *same* location in the memory. In this paper, an additional hardware structure *A-Stack* is used to avoid reordering of transactions to the same address to ensure parallel program correctness, and the TSO and PSO models are realized by using the *WTC* and *WA-Stack* hardware structures. Further, we compare the performance of these memory models with the SC model.

The OCP protocol [26] allows out-of-order completion of the *tagged* transactions of the same master (thread ID). But, *non-tagged* transaction requests and responses are strictly ordered, since connection IDs are assigned to all the masters in the system to restrict accesses to shared resources. The AXI protocol [27] allows transactions from the same master with *different ID values* to be reordered with respect to each other. But, transactions with the *same ID value* are not allowed to be reordered. Consequently, the *A-Stack* and *WA-Stack* structures in the RC and PSO/TSO models do not allow for reordering among the shared memory operations issued by the *same* processor with the *same address (non-tagged case [26], same ID case [27])*. But, the operations issued with *different addresses* are allowed to be reordered *(tagged case [26], different IDs case [27])*.

To sum up, in order to focus on the memory consistency issues, we have realized the memory models in the customized McNoC systems, which have no dependence on the cache coherence protocols. Specifically, the *TC* based realization of RC model [25] is an improvement with respect to correctness and performance.

III. TSO, PSO AND RC MODELS

The ordering constraints under the SC, TSO and PSO models are compared in Figure 1. The variables (A, B, C, D, E, F) are ordinary shared memory variables and the variable S is a special synchronization (lock) variable. The variables to the left side of the assignment operators are written and those to the right side are read. An *arrow* between the two variables indicates an ordering constraint between the operations on those variables. For instance, A→B indicates that an operation on A is followed by an operation on B in the program, and an operation on A is completed before the issuance of an operation on B. The two operations are not allowed to be reordered with each other.

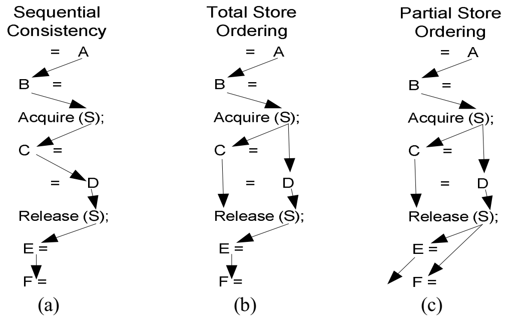


Figure 1. Comparison of SC, TSO and PSO models

As illustrated in Figure 1(a), according to the SC model, the shared memory operations 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.

A. TSO Model

The TSO model (Figure 1(b)) allows the write operation on C *to be reordered and overlapped* with respect to the following read operation on D. The TSO model compared to the SC model allows reordering and relaxation in the case of a *write followed by read* operation. It enforces the ordering constraints in the cases of *a read followed by a write* (A→B), *a write followed by a write* (E→F), or *a read followed by a read* operation. In addition, ordering constraints with respect to the synchronization (Sync) operations must also be enforced. The global orders to be enforced on the shared memory operations under TSO model are given in Figure 2(a). We refer to these global orders in section V.

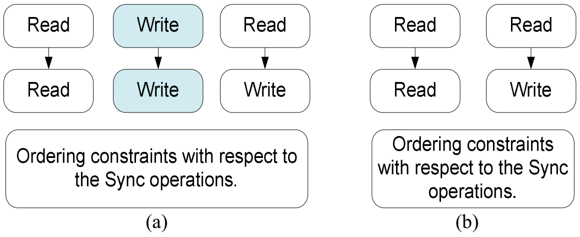

Read
Read
Write
Write
Read
Write
Ordering constraints with respect to the Sync operations.
(a)
Read
Read
Read
Write
Ordering constraints with respect to the Sync operations.
(b)

Figure 2. Global orders under: a) TSO model b) PSO model

B. PSO Model

The PSO model [6] is a refinement of the TSO model. As demonstrated in Figure 1(c), the PSO model further eliminates the ordering constraint in the case of a *write operation on E followed by a write operation on F*. It allows additional reordering among the *write* operations compared to the TSO model. The PSO model enforces the global orders on the shared memory operations as shown in Figure 2(b).

C. RC Model

The RC model [9][25] is a refinement of the weak consistency model [8][23]. It classifies the *synchronization* operations as *acquire* and *release* operations. Acquire and release operations are related to the special synchronization variables (locks, semaphores) maintained in the shared address space. The *data* operations are the read and write operations related to the ordinary shared variables. As illustrated in Figure 3(a), according to the RC model, the independent data operations on (A, B) are allowed to be reordered with each other, with the acquire operation on lock S and with the data operations on (C, D) in the critical section. They are not permitted to be reordered with respect to the release operation on lock S. The data operations (C, D) can be reordered and overlapped with respect to each other, but, they are not allowed to be reordered with the acquire and release operations on lock S. The data operations on (E, F) are allowed to be reordered with each other, with the prior outstanding release operation on lock S and with the prior outstanding data operations on (C, D). However, they are not permitted to be reordered with respect to the prior acquire operation on lock S. The data operations on (A, B, E, F) outside the acquire-release operations can be moved inside the critical section. The data operations on (C, D) cannot be moved outside the critical section. The global orders to be enforced on the shared memory operations under the RC model are given in Figure 3(b). These global orders are discussed in Section V with more detail.

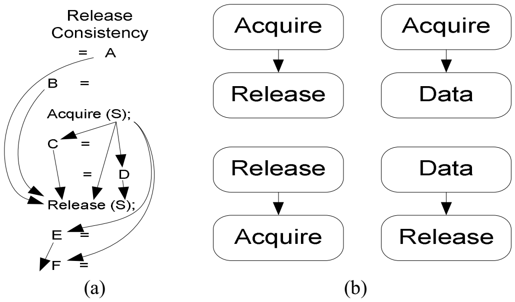


Figure 3. a) RC model b) Global orders under RC model

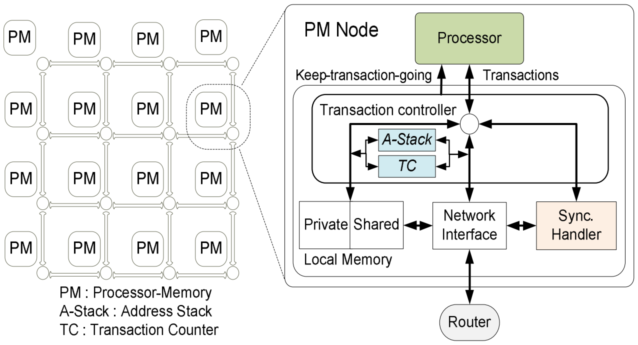
IV. DSM Based McNoC Platform

A homogenous McNoC system is shown in Figure 4(a). All the nodes are interconnected via a packet-switched network. As demonstrated in Figure 4(b), each processor-memory (PM) node consists of a processor, transaction controller (TCTRL), synchronization handler, network interface, and the local memory.

The platform uses 2D mesh packet switched *Nostrum* NoC [5] with an adaptive routing algorithm. It is a buffer-less network and only buffers in the Network Interfaces (NIs) are used to store the packets before injection into and after ejection from the network. The NI connects a PM node to the network. It deals with the transactions from the processor via TCTRL and performs packetization, queuing, arbitration and message passing over the network. It also receives the packet from the network, de-packetizes it and hands it over to the processor or memory system. The adaptive, nondeterministic nature of the routing policy means that two consecutive packets (transactions) from the same source to the same destination can be reordered on the way.

The shared memory is distributed across the network. The local memory is connected to the local processor and network interface, respectively. All shared parts in the local memories form the DSM in a single global address space.

The platform also uses distributed locks in the synchronization handler (SH). The SH controls *N* locks maintained in the global address space. Every lock is accessed in a sequential order by multiple processors in the system. A lock can either be in *locked* or *unlocked* status. The synchronization (acquire, release) requests to the SH either come from the local processor or from a remote processor via the network. If the requested lock's status is *unlocked*, then the acquire request changes its status to *locked* and an acknowledgement is sent back to the acquiring node. If the requested lock status is *locked*, a negative 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*.



(a) (b)

Figure 4. a) Homogeneous McNoC b) PM node

The transaction controller (TCTRL) is a customized interface to integrate the processor with the rest of the system. It also implements the key functions which may be required under any standard interface such as OCP [26] and AXI [27]. The TCTRL deals with the transactions from the processor and classifies them on the basis of address translation and memory mapping. It communicates with the processor to control the flow of transactions, and transmits the transactions between the processor and memory system. It also implements the *memory consistency protocols* using the hardware structures like (*TC, Address-Stack*).

The platform uses a LEON3 processor [28] in each node of the network. The data cache system is disabled from the base processor for the independant implementation of the memory consistency protocols. The cache coherence scheme can be implemented on top of this, but, the issuance and completion of the data transactions should be redefined to be tracked by the *TC* and *A-Stack* in the TCTRL. This discussion is out of the scope of the paper. The TCTRL is developed specifically for the LEON3 IP core and the main goal is to highlight the hierarchy and level where independent memory consistency protocols can be implemented in the McNoC systems. The TCTRL receives different types of transactions (read, write, memory barriers) with word granularity from the processor.

V. Realization of the TSO, PSO and RC Models

A. *TSO Model*

The flow of the operations under the realization scheme of the TSO model is illustrated in Figure 5. The write (1) and read (3) operations to the shared memory are completed by the write acknowledgment (2) and return data (4), respectively. The synchronization operation (5) to the synchronization handler is completed by the synchronization acknowledgment (6).

The TSO model is realized in the McNoC platform by enforcing the required global orders as given in Figure 2(a).
Write → Write: To enforce this global order, a Write Transaction Counter (*WTC)* is used in each node of the network to keep track of the outstanding write operations issued by a processor. The *WTC* is initialized to zero. The *WTC* is incremented by the issuance of a write operation (1). It is decremented by the completion of a write operation (2). The *WTC* is checked at the issuance of each write operation and the issuance of a write operation is delayed by stalling the processor till the completion of previously issued outstanding write operation (*WTC=0*). The processor is stalled by issuing an active low *keep-transaction-going* signal by the TCTRL in Figure 4.
Read → Read/Write: These global orders are enforced by *stalling* the processor on the issuance of a shared memory read operation (3) till its completion by returned data (4). The issuances of the subsequent *read* and *write* operations are delayed till the completion of previously issued read operation.

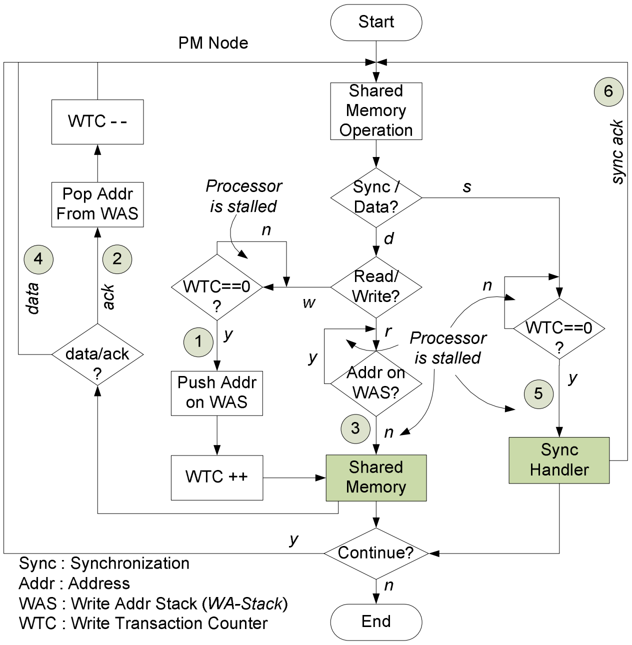


Figure 5. Realization scheme of TSO model

In order to ensure the parallel program correctness, outstanding memory operations to the same location in the memory under the TSO model are constrained to accomplish as per *program order*. A hardware structure *WA-Stack* (Write Address Stack*)* is used in each node of the network for this purpose. The *WA-Stack* keeps track of the *addresses* to be accessed by the previously issued outstanding *write* operations. At the issuance of a write operation (1) the address to be accessed by the write operation is pushed on the *WA-Stack*. On the completion of the write operation (2) the address is popped from the *WA-Stack*. The issuance of a *read* operation (3) checks the *WA-Stack*. If the address is on the *WA-Stack*, then the issuance of the read operation is delayed until the same address is popped from the *WA-Stack*. The address is popped from the *WA-Stack* on the completion of previously issued write operation to the same location in the shared memory.

The ordering constraints with respect to the Sync operations are also enforced. The shared memory operations are completed before the issuance of a Sync operation and vice versa. At the issuance of a Sync operation (5) there is no outstanding read operation, because, the processor is stalled on the issuance of a read operation till its completion. However, to ensure the completion of the outstanding write operation *WTC* is checked at the issuance of each Sync operation. The issuance of a Sync operation is delayed till the completion of previously issued outstanding write operations (*WTC=0*). After the issuance of a Sync operation the following memory operations are delayed by stalling the processor till the successful completion of the Sync operation (6).

B. *PSO Model*

The PSO model is realized by enforcing the required global orders as shown in Figure 2(b). The ordering requirements under the PSO and TSO models are mostly similar except the PSO model further relaxes the ordering constraint on the shared memory operations in the case of a *write followed by a write* operation.

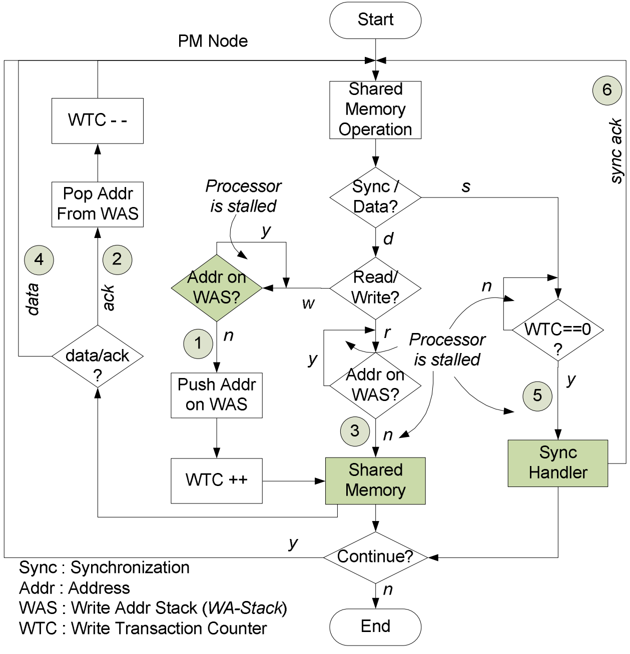


Figure 6. Realization scheme of PSO model

As illustrated in Figure 6, the issuance of a shared memory write operation (1) is not delayed till the completion of previously issued outstanding write operation (*WTC=0*). It allows multiple outstanding shared memory write operations on the network, which is not allowed under the TSO model (Figure 5). The independent shared memory *write* operations can be reordered with respect to each other. The PSO model also uses *WA-Stack* to prohibit the reordering among the shared memory *write* operations to the *same* location in the shared memory. There is an additional check on the *WA-Stack* under the PSO model. The *WA-Stack* is checked on the issuance of each shared memory write operation (1). If the address to be accessed by the write operation is on the *WA-Stack,* then there is an outstanding write operation to the same location. The issuance of the write operation is then delayed until the same address is popped from the *WA-Stack* on the completion of a previously issued write operation to the same location. The rest of the realization scheme is similar to that described under the TSO model.

C. *RC Model*

The realization scheme of the RC model is demonstrated in Figure 7. A data (read, write) operation (1) to the shared memory is either completed by the return data or write acknowledgment (2). The *acquire* operation (3) to the synchronization handler is completed by the acquire acknowledgment (4). The *release* operation (5) is completed by the release acknowledgment (6).

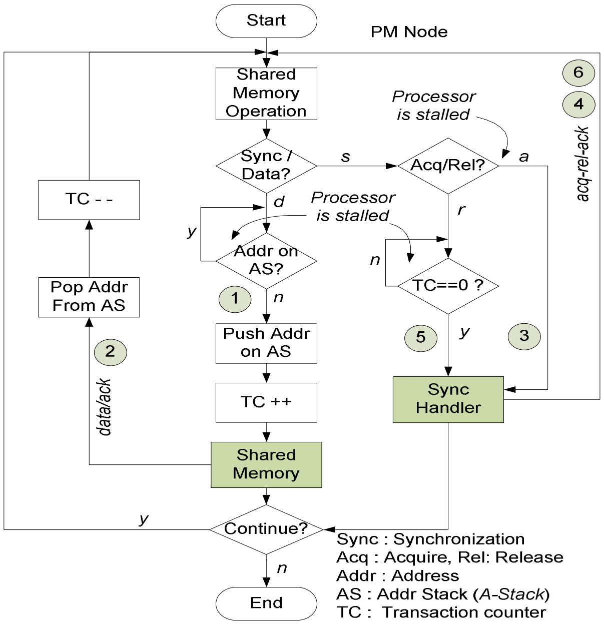


Figure 7. Realization scheme of RC model

The RC model is realized in the McNoC platform by enforcing the required global orders as given in Figure 3(b).

Data → Release: To enforce this global order, a Transaction Counter (*TC)* is used in each node of the network to keep track of the outstanding *data* operations issued before the release operation. Initially, the *TC* is zero. The *TC* is incremented by the issuance (1) of a data operation. It is decremented by the completion (2) of a data operation. The *TC* is checked at the issuance of a release operation (5) and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations (*TC=0*).

Acquire → Data/Release: These global orders are enforced by *stalling* the processor upon the issuance of an acquire operation (3) till the successful acquisition of a lock (4). The following shared memory operations in the critical section and the release operation are delayed for the lock acquisition. The lock must be gained by a processor before entering to the critical section and also before trying to release it.

Release → Acquire: This global order is enforced by sequential order on a lock in the multiprocessors system as discussed in section IV. A lock must be released by a processor before the next acquire on it.

The *A-Stack* is used in each node of the network to constrain the *data* operations issued by a processor to the *same* location in the shared memory. The *A-Stack* keeps track of the *addresses* to be accessed in the shared memory by the previously issued outstanding shared memory *data* operations. At the issuance of a data operation (1), the address to be accessed in the shared memory by the data operation is pushed on the *A-Stack*. On completion (2) of a data operation the address is popped from the *A-Stack*. On issuance of each data operation the *A-Stack* is checked. If the address is already on the *A-Stack,* there is an outstanding data operation issued to the same location in the shared memory. The issuance of the data operation is then delayed until that address is popped from the *A-Stack* by the completion of the previous data operation on the same address.

VI. EXPERIMENTS AND RESULTS

A. *Hardware implementation cost*

The designs are synthesized by using Synopsis Design Compiler with SMIC 90nm technology and optimized for area. The synthesis results in term of nand-gate equivalent and maximum frequency are given in Table I. The difference in the area costs of the designs is mainly in the transaction controller (TCTRL), which uses *transaction counter* and *address stack* to implement the memory models. The TCTRL consumes 26.85%, 27.3%, 26.97% and 27.04% of the total area under the SC, TSO, PSO and RC models, respectively. The area cost for the TSO, PSO and RC models are increased by 1.8% (463 gates), 0.57% (153 gates) and 0.93% (243 gates) over the SC model in the TCTRL. The switch, synchronization handler (SH) and network interface (NI) together consume 73.14%, 72.78%, 73.03% and 72.95% of the total area under the SC, TSO, PSO and RC models, respectively. In all cases, the maximum clock frequency is 500 MHz or above.

TABLE I. SYNTHESIS RESULTS WITH 90 NM SMIC TECHNOLOGY

	SC Model		TSO Model		PSO Model		RC Model	
	A	F	A	F	A	F	A	F
Switch	13.24	0.5	13.24	0.5	13.24	0.5	13.24	0.5
SH	3.76	1.25	3.76	1.25	3.76	1.25	3.76	1.25
NI	49.99	1.25	49.99	1.25	49.99	1.25	49.99	1.25
TCTRL	24.6	0.5	25.05	0.5	24.74	0.5	24.83	0.5
Total	**91.59**		**92.04**		**91.73**		**91.83**	

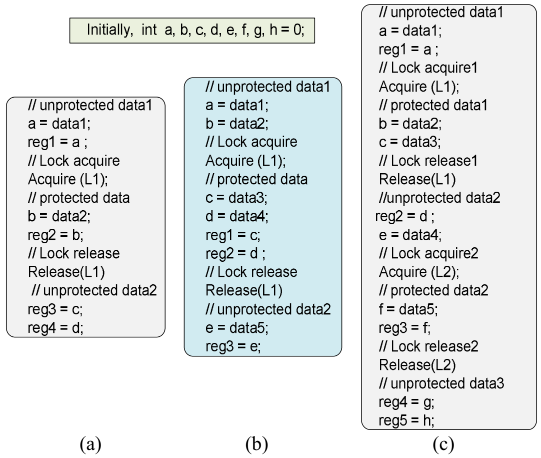
A: AREA (KILO NAND GATES), F: FREQUENCY (GHZ)

B. *Experimental Setup*

We experimented on a configurable multicore NoC based cycle true simulation platform constructed in VHDL (Figure 4). The LEON3 processor core is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. The platform uses a DSM system and the size of the shared, local memory of each node is 16 MB. The SH in each node maintains 256 locks in the shared address space. The *TC* and *WTC* each are 32 bits. The *A-Stack* and *WA-Stack* can stack up to 64 addresses each with 24 bits. The sizes of the stacks are kept small and they are utilized efficiently. The addresses are popped from the *stack* continuously by the completion of operations in a pipelined style. The packet formation in the NI uses 7 fields (96 bits). The buffering capacity at the NI is 64 packets. The *Nostrum* NoC [5] uses 2D regular mesh topology and deflection routing policy. The caches are disabled from the LEON3 processors in the experiments, since; they are neutral for our evaluation of the memory models. In the experiments, the effects on execution time of network size and different traffic patterns are investigated and speedup, overhead and efficiency under memory models are reported.

C. *Experiments with Synthetic Workloads*

The performance of the RC, PSO, TSO and SC models are evaluated with three different synthetic workloads (WL1-WL3) as shown in Figure 8. These workloads are manually mapped on the LEON3 processors in the network. The same sequence of transactions is generated by the processor in each node. WL1 contains data and synchronization operations and has both *write followed by read* and *read followed by read* cases. WL2 contains *write followed by write, write followed by read* and *read followed by read* sequences. WL3 has, in addition, *read followed by write* operations and uses two non-overlapped critical sections. For the lock and protected (critical section) data operations hotspot traffic pattern is generated.



```
Initially,  int  a, b, c, d, e, f, g, h = 0;
```

(a)

```
// unprotected data1
a = data1;
reg1 = a ;
// Lock acquire
Acquire (L1);
// protected data
b = data2;
reg2 = b;
// Lock release
Release(L1)
 // unprotected data2
reg3 = c;
reg4 = d;
```

(b)

```
// unprotected data1
a = data1;
b = data2;
// Lock acquire
Acquire (L1);
// protected data
c = data3;
d = data4;
reg1 = c;
reg2 = d ;
// Lock release
Release(L1)
// unprotected data2
e = data5;
reg3 = e;
```

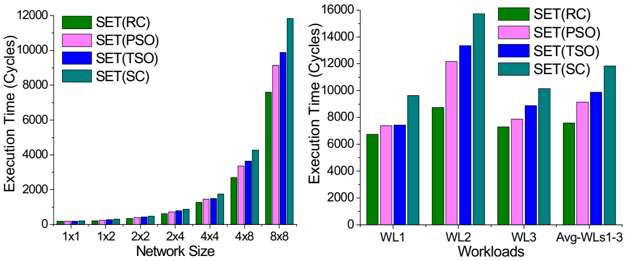
(c)

```
 // unprotected data1
 a = data1;
 reg1 = a ;
 // Lock acquire1
 Acquire (L1);
 // protected data1
 b = data2;
 c = data3;
 // Lock release1
 Release(L1)
 //unprotected data2
reg2 = d ;
 e = data4;
 // Lock acquire2
 Acquire (L2);
 // protected data2
 f = data5;
 reg3 = f;
 // Lock release2
 Release(L2)
 // unprotected data3
 reg4 = g;
 reg5 = h;
```

Figure 8. Sequences of transactions generated a) WL1 b) WL2 c) WL3

D. *Results and Discussion*

The Synthetic workload Execution Times (SETs) are compared for the RC, PSO, TSO and SC models (Figure 9(a)). The SC model is used as the baseline model. As the system scales up, the SETs are quickly increased under all the consistency models. It is due to the increasing network traffic and increasing waiting time for acquiring the locks. The SET is decreased under the RC model by allowing further reordering and relaxation compared to the other memory models. The average SETs of the RC, PSO and TSO models in the 8x8 network are reduced by 35.8%, 22.7% and 16.5% over the SC model, respectively. For the three synthetic workloads the SETs under all memory models in the 8x8 network are shown in Figure 9(b). The SET reduction is more under the WL2 compared to the WL1/WL3, due to the issuance of more data operations before the release operation.



(a) (b)

Figure 9. SETs for WL1, WL2, and WL3

E. *Application Workloads*

1) *Matrix Multiplication*

The matrix multiplication application workload calculates the product of two integer matrices X[64x1] and Y[1x64], resulting in a Z[64x64]. Three matrices are decomposed into sub-matrices and stored in the distributed shared memory.

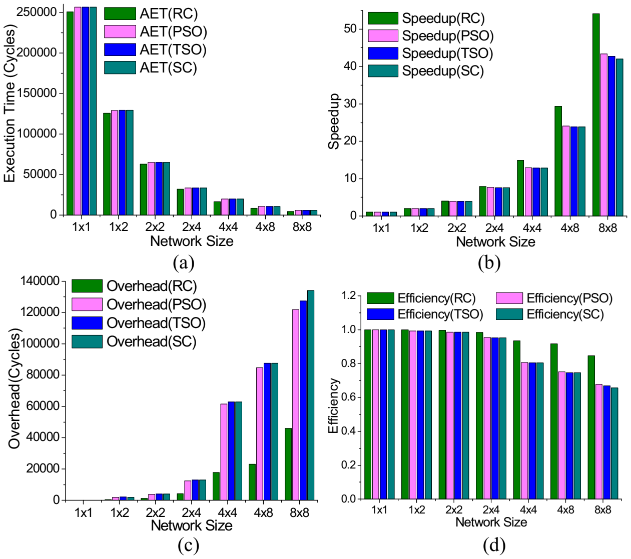


Figure 10. Matrix Multipl: a) AETs b) Speedup c) Overhead d) Efficiency

The Application workload Execution Times (AETs) under all memory models are decreased due to the division of the computation cost in the larger networks (Figure 10(a)). The AETs under the RC, PSO and TSO models in the 8x8 network are reduced by 24.1%, 3.2% and 1.8% over the SC model, respectively. The *speedup* is defined as the ratio of the single core execution time (*Ts*) and the execution time of the multicore (*Tm*). The speedup (Figure 10(b)) grows faster under the RC model compared to the other memory models as the system size scales up. The AET reduction and speedup under the RC model is higher compared to the other memory models due to additional reordering and overlapping in the shared memory operations. We define the *overhead* of the multicore system as: *Nc*Tm-Ts*, where, *Nc* is the number of cores in the system. The overhead increases as the system size grows due to the increasing communication cost (Figure 10(c)). The *efficiency* of the multicore system is defined as the ratio *speedup*/*Nc*. The RC model maintains high efficiency throughout compared to the stricter memory models (Figure 10(d)) when the system size is increased.

2) *Pattern Search*

The application searches P[64] data patterns against the M[64] data elements, which are initialized in the distributed shared memory across the network. Each node operates on the data patterns in its local shared memory and the data elements from remote nodes, and writes the output results into its local shared memory. The output values N[64] are the number of times the data patterns that appear in the data elements.

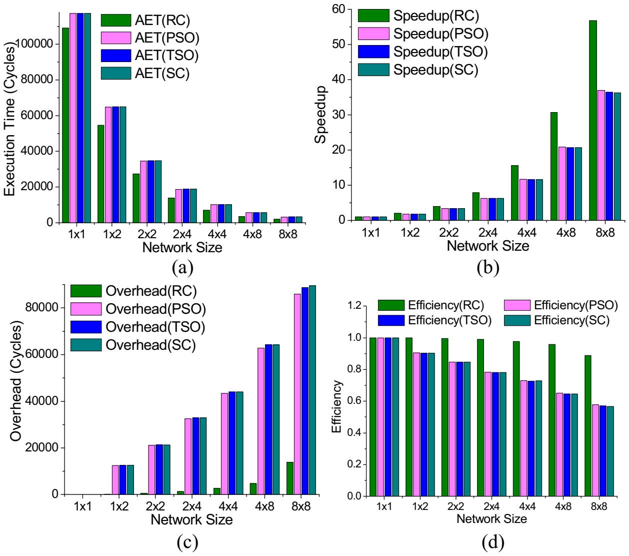


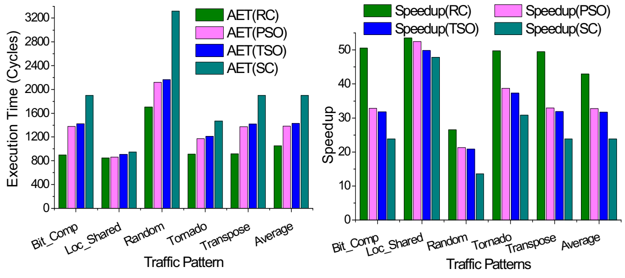
Figure 11. Pattern Search: a) AETs b) Speedup c) Overhead d) Efficiency

The AET under the RC, PSO and TSO models in the 8x8 network is reduced by 40.6%, 1.8% and 0.4% over the SC model, respectively (Figure 11(a)). The speedup for the RC model is 56.8 (almost ideal) while for the PSO, TSO and SC models are 36.9, 36.4 and 36.2 (Figure 11(b)). The AET reduction is more and speedup is high due to low computation to communication ratio compared to the matrix multiplication application. As a result, the RC model allows more outstanding data operations in the network, which can overlap and pipeline with each other. Similarly, the overhead and efficiency are also affected. The overhead in the 8x8 network is reduced by 84.6%, 4.1% and 0.9% under the RC, PSO and TSO models over the SC model, respectively (Figure 11(c)). The efficiency

(Figure 11(d)) on the 8x8 system for the RC model is high 0.88 (close to the ideal case 1), while for the PSO, TSO and SC models are 0.58, 0.57 and 0.56, respectively.

3) Bit Count/Data Analysis

The application analyzes a data vector M[1024] and calculates the number of set bits (1) in each integer data item. The M data elements are initialized in the distributed shared memory across the network. These data items are read, analyzed and the output values (number of 1s) are stored in the distributed shared memory. Five different traffic patterns are used. Each node operates on the data items in its (*Bit complement, local shared, random, tornado of degree one and transpose)* node and also writes the output results in to the same node.



(a) (b)

Figure 12. Bit Count: a) AETs b) Speedup

As illustrated in Figure 12(a), the AETs reduction is more for the RC, PSO and TSO models over the SC model under the *bit complement* traffic pattern, while least is observed under the *local shared* traffic pattern. The AETs are dependent on the *physical distance* between the initiator and target nodes in the network. The AETs are high under *random* traffic pattern, because, irregular traffic pattern increases the network congestion and delay. The AETs under the RC, PSO and TSO models on the average are reduced by 44.5%, 27.2% and 24.9% over the SC model in the 8x8 network. As expected, the AET reduction is more under the RC model. Likewise, the speedup is high under the local shared and least under random traffic pattern (Figure 12(b)).

VII. Conclusion

The hardware support (transaction counter, address stack) for the three memory models and their performance comparison are shown in the McNoC systems. The results show that under synthetic workloads, the average execution time for the RC, PSO and TSO models in the 8x8 network is reduced by 35.8%, 22.7% and 16.5% over the SC model, respectively. For the application workloads, the execution time in the 8x8 network under the RC, PSO and TSO models is decreased on average by 36.4%, 10.7% and 9% compared to the SC model. The area cost for the relaxed models is increased by less than 2% over the SC model at the processor interface.

References

[1] Axel Jantsch et al., "Memory architecture and management in an NoC platform," in: Axel Jantsch and D. Soudris, editors, Scalable Multicore Architectures: Design Methodologies and Tools. Springer, 2011.

[2] S. V. Adve et al., "Shared Memory Consistency Models: A Tutorial," Digital Western Research Laboratory, report no. 95/7, USA, 1995.

[3] 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.

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

[5] A. Jantsch "The Nostrum NoC," in: http://www.ict.kth.se/nostrum.

[6] David L. Weaver and Tom Germond, editors. "The SPARC ArchitectureManual," Prentice Hall, 1994, SPARC International, Ver. 9.

[7] P. Sewell et al., "x86-TSO: A Rigorous and Usable Programmer's Model for x86 Multiprocessors," Communications of the ACM, 2010.

[8] M. Dubois et al., "Memory access buffering in multiprocessors," in: Proc. of 13th Ann. Inter. Symp. on Comp. Arch. (ISCA'86), 1986.

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

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

[11] Daniel J. Sorin et al., "A Primer on Memory Consistency and Cache Coherence," Morgan & Claypool Publishers, 2011.

[12] J. Zebchuk, V. Srinivasan, M.K. Qureshi, and A. Moshovos, "A Tagless Coherence Directory," MICRO '09: 2009 42st International Symposium on Microarchitecture, New York, NY, 2009.

[13] Ferdman, M.; Lotfi-Kamran, P.; Balet, K.; Falsafi, B.; "Cuckoo directory: A scalable directory for many-core systems," in: Proc. of 17th International Symposium on High Performance Computer Architecture (HPCA), pp.169-180, Feb. 2011.

[14] M.M.K. Martin, M.D. Hill, and D.A. Wood, ''Token Coherence: Decoupling Performance and Correctness,'' Proc. 30th Ann. Int'l Symp. Computer Architecture (ISCA '03), ACM Press, 2003, pp. 182-193.

[15] M. Herlihy and J. E. B. Moss. "Transactional memory: Architectural support for lock-free data structures," In: Proc. of the 20th Ann. Int'l Symp. on Computer architecture (ISCA'93), pages 289–300, 1993.

[16] P. Damron et al, "Hybrid Transactional Memory," in Proc. of the 12th international conference on (ASPLOS'06), California, Oct. 2006.

[17] J. Manson, W. Pugh, and S. Adve. "The Java memory model," In: Proc. of the ACM Symposium on Principles of Programming Languages (POPL'05), pages 378–391, Long Beach, CA, Jan. 2005.

[18] Romanescu, B.; Lebeck, A.; Sorin, D.J.; , "Address Translation Aware Memory Consistency," IEEE Micro, , vol.31, no.1, pp.109-118, 2011.

[19] Arvind; Maessen, J.-W.; , "Memory Model = Instruction Reordering + Store Atomicity," in: proc. of 33rd International Symposium on Computer Architecture (ISCA '06), pp.29-40, 2006.

[20] F. Petrot, A. Greiner, P. Gomez, "On cache coherence 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.

[21] Andreas Hansson, and Kees Goossens. "An On-Chip Interconnect and Protocol Stack for Multiple Communication Paradigms and Programming Models," In: Proc. of CODES+ISSS'09, France, 2009.

[22] 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), August 2007, pp. 27-34.

[23] A. Naeem et al., "Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems," In Proc. of the ASP-DAC, Japan, 2011.

[24] A. Naeem, X. Chen, Z. Lu, and A. Jantsch, "Scalability of Relaxed Consistency Models in NoC based Multicore Architectures," ACM SIGARCH Computer Architecture News, April 2010, 37(5): 8-15.

[25] A. Naeem et al., "Realization and scalability of release and protected release considtency models in NoC based systems," in: Proc. of Euromicro Conf. on Digital Systems Design, DSD, Oulu, Finland, 2011.

[26] OCP International Partnership. OCP Specification 2.2, 2007.

[27] "AMBA AXI Protocol Specification," in: http://infocenter.arm.com/

[28] http://jorisvr.nl/leon3_insntiming.html