Scalability Analysis of Memory Consistency Models in NoC-based Distributed Shared Memory SoCs

Abdul Naeem, Axel Jantsch, *Member, IEEE,* and Zhonghai Lu, *Member, IEEE*

***Abstract*—We analyze the scalability of six memory consistency models in network-on-chip (NoC)-based distributed shared memory multicore systems: 1) protected release consistency (PRC); 2) release consistency (RC); 3) weak consistency (WC); 4) partial store ordering (PSO); 5) total store ordering (TSO); and 6) sequential consistency (SC). Their realizations are based on a transaction counter and an address-stack-based approach. The scalability analysis is based on different workloads mapped on various sizes of networks using different problem sizes. For the experiments, we use Nostrum NoC-based configurable multicore platform with a 2-D mesh topology and a deflection routing algorithm. Under the synthetic workloads, the average execution time for the PRC, RC, WC, PSO, and TSO models in the 8 × 8 network (64-cores) is reduced by 32.3%, 28.3%, 20.1%, 13.8%, and 9.9% over the SC model, respectively. For the application workloads, as the network size grows, the average execution time under these relaxed memory models decreases with respect to the SC model depending on the application and its match to the architecture. The performance improvement of the PRC and RC models over the SC model tends to be higher than 50% as observed in the experiments, when the system is further scaled up. The area cost in the network interface for the relaxed memory models is increased by less than 4% over the SC model.**

***Index Terms*—Distributed shared memory, memory consistency, network-on-chip, performance, scalability.**

I. Introduction

ADVANCED systems-on-chip (SoCs) tend to support parallelization at the computation (multicore), communication (network-on-chip, NoC), and memory architecture levels [1]. The distributed shared memory (DSM) on-chip is preferred to exploit the distributed nature of the NoC-based systems. Since shared memory operations can be reordered due to the system optimizations both in the hardware (e.g., write buffer, cache, interconnection network) and in the software (e.g., compiler optimization, register allocation), which may lead to the unexpected behavior of DSM systems [2]. Memory consistency determines the execution order of memory operations for the correct behavior of DSM systems. Different memory consistency models (often called memory models) enforce different ordering constraints on the memory operations [2]. The sequential consistency (SC) model [3] enforces strict ordering constraints and does not take advantage of the system optimizations. As a result, several relaxed consistency models [2], [4], [9], [10] are proposed by relaxing the ordering constraints on the memory operations to exploit the system optimizations.

The cache coherence and memory consistency are two problems of a similar nature in the DSM systems. The coherence problem is due to the different cached copies of the same shared data in the system (either using a write through or write back policy). The coherence protocols resolve this issue. The snooping-based coherence protocol relies on broadcasting and bus snooping by the cache controllers. But it is unscalable for the network-based multicore systems. Alternatively, the directory-based coherence protocol is used to maintain the state information of cache blocks and sends messages to invalidate or update the cached copies of the requested block. However, it has some confronting issues like extra coherence traffic, directory overhead, additional latencies, and complexities. In contrast, memory consistency is related to the ordering constraints on memory operations for the parallel program correctness, i.e., a read operation must always return the correct value of a memory location in the multiprocessor systems. In situations like hard real time applications and some other applications where caches are not used or when these problems have different requirements on the size of consistency and cache object, independent implementation schemes for these two problems are preferred [1], [28]–[30], [34].

Moreover, heterogeneous and customized DSM systems have different requirements and design constraints compared to the general multiprocessor systems. The former have tighter power constraints, require 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, we study six memory models: protected release consistency (PRC), release consistency (RC), weak consistency (WC), partial store ordering (PSO), total store ordering (TSO), and SC models. These models are realized independent of the coherence protocols [1], [28]–[30], [34] by means of transaction counters and address stacks (Section V).

Our experiments use a configurable McNoC platform. The platform uses DSM, distributed locks and on-chip 2-D mesh *Nostrum* network [5] with a deflection routing policy. We analyze the scalability of six memory models in the McNoC systems. The experimental results show the performance gain

Manuscript received June 6, 2012; revised October 15, 2012; accepted December 7, 2012. Date of current version April 17, 2013. This paper was recommended by Associate Editor Y. Xie.

The authors are with the Department of Electronic Systems, School of Information and Communication Technology, KTH-Royal Institute of Technology, SE-164 40 Kista, Stockholm, Sweden (e-mail: abduln@kth.se; axel@kth.se; zhonghai@kth.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2235914

of the relaxed memory models due to reordering in the memory operations compared to the SC model. Going beyond the previous works, this paper makes the following contributions.

1) A comprehensive analysis of the ordering constraints under six different memory models is presented compared to the previous works [1], [28]–[30], [34].
2) The previous works [1], [28]–[30], [34] focus on the architecture support of memory models in the McNoC systems. In [36], the scalability of two memory models is analyzed. This paper analyzes the scalability of six memory models in the McNoC systems up to 64-cores. Workloads are mapped on the increasing size of the network. On average, the execution time under PRC, RC and WC models relative to the SC model decreases by 2.5 percentage points for each doubling of the core count up to 64-cores, the maximum number of cores considered in our study, while for the TSO and PSO models the performance improvement is approximately 0.5 percentage points.
3) The performance of these memory models is studied under a variety of scenarios with both data and synchronization intensive workloads to highlight some more aspects. For instance, the performance of the PRC and RC models over the stricter models improves with the network size up to the point where the application matches to the available parallelism. Beyond this point, the improvements flatten or drop. The studies show the dependence of performance gain on the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio and problem size.
4) In contrast to [1], [28]–[30], [34], the realization schemes of the WC and PRC models are improved to avoid reordering of accesses to the same memory location.
5) In contrast to [29], the performance gain of 5% to 10% under PRC over RC model is observed under the synchronization intensive wavefront computations.

The rest of the paper is structured as follows. Related work is overviewed in the next section. In Section III, the memory models are described. In Section IV, the DSM-based McNoC platform is introduced. Section V presents the realization schemes of these memory models. In Section VI, the impact of system optimizations on the memory models is discussed. In Section VII, simulation results and scalability analysis of six memory models are described in the McNoC systems and finally Section VIII summarizes our contributions.

II. Related Work

A. Memory Consistency in Multiprocessors DSM Systems

Several memory consistency models are described in the literature [2], [4], [9], [10]. Adve *et al.* [2] discussed the memory consistency models from the system optimizations point of view. The SC model [3] enforces a total order on the memory operations. The TSO model [6], [7] relaxes the ordering constraints in the case of a write followed by a read operation. The PSO model [6] further provides the relaxation among the write operations. The ordering constraints on the memory operations under both the TSO and PSO models are enforced using different kind of *fence* instructions (non-memory references). For instance, (MEMBARs, SBAR) are used in the SPARC architectures [6] and (MFENCE, SFENCE, LFENCE) are used in the x86 architectures [7]. The WC model [8] classifies the shared memory operations as data and synchronization operations. The data operations issued between two consecutive synchronization points can be reordered with each other. The RC model [9] further classifies the synchronization operations as acquire and release operations.

Some memory models provide additional relaxations compared to the RC model. For example, in the *eager RC* model [11] the propagation of all the modifications are delayed at the release points and the number of messages are reduced compared to the RC model. The *lazy RC* [12], further delays the propagations of modifications till the acquisition of a lock by another processor. The *Entry RC* [13] categorizes the lock acquire into exclusive and non-exclusive modes. A lock can be acquired by only one processor in the exclusive mode. However, in the non-exclusive mode, a lock can be acquired by more than one processor, if they perform only read operations in their critical sections. The *scope consistency* [14] enforces the global orders within a scope, which is defined by all the critical sections protected under a lock. It reduces the false sharing compared to the lazy RC model and uses the page as the coherence atom compared to the entry RC model.

The DASH project [15] implements the RC model using several counters-based mechanism which depends on the coherence protocol. The directory-based coherence protocol is used to maintain the state information of the cache blocks. Recent works [16] and [17] on the directory-based protocols focus on the reduction of the directory overheads, energy and power consumption. Token coherence [18] decouples the performance and correctness of the coherence protocols by associating tokens with each memory block to track the correct transfer and accesses to that block. Since the performance protocol (*TokenB*) is based on broadcasting transient requests, it is not a scalable approach. The location consistency [19] relies on the memory coherence for the scalable architectures by eliminating the limitations of coherence protocols in the form of directories and snooping.

Recently, address translation aware memory consistency models at physical and virtual address levels (PAMC, VAMC) have been proposed in [20]. The address translation and translation coherence are proposed to enforce a total order on all the operations. They emphasized on the detection of design and runtime faults due to the address translation. In [21], a memory model is defined in terms of instruction reordering and store atomicity. The main focus of the work is on the store atomicity and serializability issues.

Transactional memories target to scale the programmer productivity by moving the synchronization burden to the hardware or/and software platform support. The hardware approach [22] relies on the 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 [23] combines the

benefits of both. Memory models are also explored at the high-level programming languages. The Java memory model [24] specifies legal transformations and optimizations for the compiler and virtual machine or hardware.

B. Memory Consistency in NoC-Based Multicore Systems

In NoC-based systems, the proposed mechanism in [25] is very restrictive and allows only one outstanding transaction of an initiator at a time in the network. A protocol stack for on-chip interconnects is proposed at different levels of the SoC design [26]. They briefly outlined the mechanism to implement the RC model at the memory-mapped stack. However, they do not discuss its implementation detail. The streaming consistency [27] is based on the software cache coherence protocol. However, polling the circular buffer at each request level may not be feasible in the larger systems.

The transaction counter (TC)-based hardware approaches are adopted in [1], [28]–[30], and [34] to realize the memory models independent of the coherence protocols in the McNoC systems. In [28], the SC model is realized by stalling the processor on the issuance of an operation. The WC model is realized using a TC in each node of the network to keep track of outstanding data operations issued by a processor in the system. In [34], the RC model is realized by using two TCs-based approach. 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, which is already checked at the previous release points. In [29], a single TC-based approach is adopted to realize the RC and PRC models in the McNoC systems. The PRC model is proposed as an extension of the RC model. In [30], the realization of RC model is further enhanced to ensure the parallel program correctness by using a hardware structure address stack (A-Stack) in each node of the network. Also, the TSO and PSO models are realized by using the write transaction counter (WTC) and write address stack (WA-Stack)-based approaches.

The AXI [31] and OCP [32] protocols enforce the ordering models by using transactions IDs and thread IDs, respectively. In [31], transactions of the same master with different IDs can be reordered, but transactions with the same ID are not allowed to be reordered. In [32], tagged transactions of the same master using thread IDs are allowed to be reordered, but non-tagged transactions are strictly ordered. Likewise, the A-Stack and WA-Stack are used in the realization schemes of the memory models to constrain the memory operations issued by a processor with the same address, but memory operations with different addresses are allowed to be reordered.

The coherence protocols deal with the consistency at cache block level, while the memory consistency models address it for the entire shared memory. The memory consistency issue can be observed both in the systems which may or may not use the data caches. As discussed earlier, some applications require independent solutions for these two problems [30].

To summarize, our implementation of memory models in the McNoC systems have lower hardware overhead and uses simple programming model without too many fences. The ordering constraints are mostly enforced on the memory operations by using the hardware structures like transaction counter and address stack in the processor interface. More specifically, this paper compares six memory models and comprehensively analyzes their scalability in the systems up to 64-cores. For a number of applications mapped on the various sized networks, the average execution time under the relaxed memory models relative to the SC model is analyzed and some key performance affecting factors are highlighted.

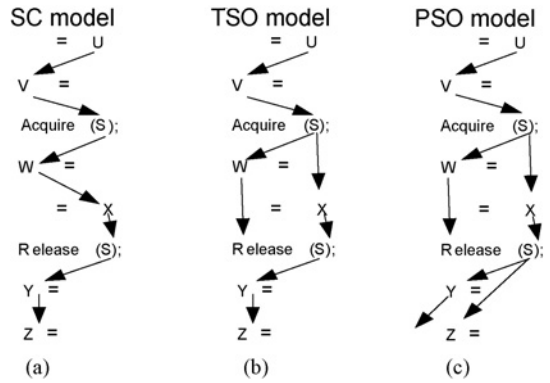

Fig. 1. Comparison of SC, TSO, and PSO models.

III. Memory Consistency Models

Different memory consistency models enforce different ordering constraints on the memory operations. We discuss the ordering constraints under six different memory consistency models as shown in Figs. 1 and 2. The variables (U, V, W, X, Y, Z) are ordinary global (shared) variables and the variable S is a special synchronization (lock) variable. The variables to the left side of the assignment operators are updated (written) and those on the right side are read. An arrow between two variables indicates an ordering constraint between the operations on these variables. For instance, $U \rightarrow V$ indicates that an operation on variable U is followed by an operation on variable V in the program, and an operation on U is completed before the issuance of an operation on V. These operations are not allowed to be reordered with each other.

A. SC Model

The SC model [3] enforces the program order among the operations of an individual processor and the sequential order among the multiple processors on the critical resource/shared memory. As illustrated in Fig. 1(a), according to the SC model, the 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 the processors in the system.

B. TSO Model

The TSO model [Fig. 1(b)] allows the write operation on W to be reordered and overlapped with respect to the following read operation on X. In contrast to the SC model, TSO model allows reordering and relaxation in the case of a write followed by a read operation. The ordering constraints are enforced in

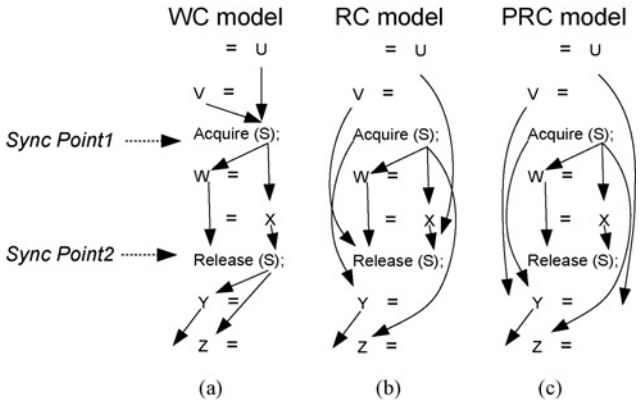

Fig. 2. Comparison of WC, RC, and PRC models.

the cases of a read followed by a write ($U \rightarrow V$), a write followed by a write ($Y \rightarrow Z$), and a read followed by a read operation.

C. PSO Model

The PSO model [6], [7] is a refinement of the TSO model. As given in Fig. 1(c), the PSO model further eliminates the ordering constraint in the case of a write operation on Y followed by a write operation on Z. It allows additional reordering among the write operations compared to the TSO model.

D. WC Model

The WC model [8], [28] classifies the shared memory operations as *synchronization* (Sync) and *data* operations. The Sync operations are related to the special Sync variables (locks, semaphores) in the shared address space. The data operations are the (read, write) operations related to the ordinary shared variables. As illustrated in Fig. 2(a), the independent data operations issued between the two consecutive Sync points can be reordered with respect to each other. The data operations are not allowed to be reordered with respect to the Sync operations and vice versa. For instance, the data operations on (U, V) are allowed to be reordered with each other, but they are not permitted to be reordered with the data operations on (W, X) or (Y, Z), because they are not issued in between the two consecutive synchronization points.

E. RC Model

The RC model [9], [15], and [29] provides additional reordering and relaxation compared to the WC model [8]. It further classifies the synchronization operations as *acquire* and *release* operations. An acquire operation delays the future data operations until the lock is obtained. It does not wait for the completion of previously issued data operations. A release operation notifies the completion of previously issued data operations. It does not delay the future data operations. As demonstrated in Fig. 2(b), according to the RC model, the independent data operations on (U, V) are allowed to be reordered with each other, with the acquire operation on lock S, and with the data operations on (W, X) in the critical section. They are not permitted to be reordered with respect to the release operation on lock S. The data operations (W, X) 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 (Y, Z) are allowed to be reordered with respect to each other, with the prior outstanding release operation on lock S, and with the prior outstanding data operations on (W, X). However, they are not permitted to be reordered with respect to the prior acquire operation on lock S. The data operations on (U, V, Y, Z) outside the acquire-release operations can be reordered with the data operations in the critical section, while this is not permitted under the WC model. The data operations on (W, X) cannot be moved outside the critical section.

F. PRC Model

The PRC model is proposed as an extension of the RC model which provides additional reordering and relaxations in the memory operations [29]. It further categorizes the data operations as *unprotected* and *protected* operations. As given in Fig. 2(c), the data operations on (U, V, Y, Z) are unprotected data operations. They are not protected under acquire-release operations on lock S or any other lock. The data operations on (W, X) are protected data operations under acquire-release operations on lock S. The unprotected data operations on (U, V) can be reordered and overlapped with each other, with the following acquire, protected data, release, and unprotected data operations. The PRC model allows the reordering of the unprotected data operations on (U, V) with the following release operation on lock S and with the unprotected data operations on (Y, Z), because they are independent of each other. However, it is not allowed under the RC model. The issuance of a release operation is only delayed till the completion of previously issued protected data operations and it is not delayed for the completion of previously issued unprotected data operations. While in the case of RC model, the release operation unnecessarily notifies the completion of previously issued unprotected data operations (U, V) which are independent of each other. This additional categorization of the data operations under the PRC model distinguishes the ordering requirements for the different types of data operations compared to the RC model.

G. Global Orders Under Memory Consistency Models

The global orders (ordering constraints) to be enforced under the SC model are shown in Fig. 3(a), program order and sequential order. The global orders under the TSO and PSO models are given in Fig. 3(b) and 3(c). The TSO model enforces the order among the write operations compared to the PSO model. Both the TSO and PSO models also enforce the ordering constraints with respect to the synchronization (Sync) operations for the parallel program correctness.

As shown in Fig. 4(a), the WC model does not allow reordering among data and synchronization operations. The synchronization operations must be completed in the program order. Under the RC model, as given in Fig. 4(b), an acquire operation must be completed before the issuance of a data

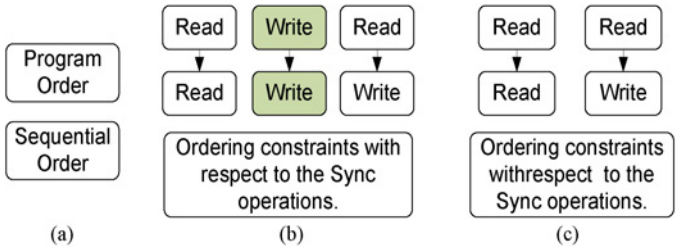

Fig. 3. Global orders under: (a) SC model; (b) TSO model; (c) PSO model.

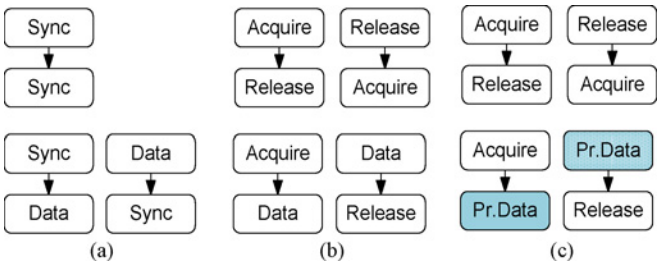

Fig. 4. Global orders: (a) WC; (b) RC; (c) PRC, Pr. data: protected data.

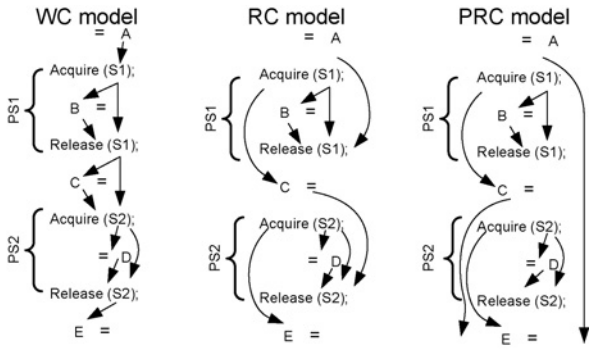

Fig. 5. Non-overlapped protected section, protected data: B, D, and unprotected data: A, C, E.

and release operations. Similarly, before the issuance of a release operation the prior acquire and data operations must be accomplished. Unprotected data operations under the PRC model are not constrained at the acquire and release points and they can cross these barriers [Fig. 4(c)]. These global orders are discussed later in Section V with more details.

H. Further Analysis of the WC, RC, and PRC Models

For comprehensive analysis and deeper understanding of the ordering constraints under the WC, RC, and PRC models, we consider further cases with non-overlapped, nested, and partially overlapped protected sections.

Fig. 5 illustrates the ordering requirements under the WC, RC, and PRC models for the code segments using non-overlapped protected sections. The critical sections under the locks S1 and S2 are the two protected sections (PS1, PS2), which do not overlap with each other. A curly bracket represents a protected section. The WC model does not permit the unprotected data operations on (A, C, E) to be reordered with respect to the protected data operations. The RC model allows the unprotected data operations on (A, E) to be reordered and overlapped with the protected data operations in PS1 and PS2,

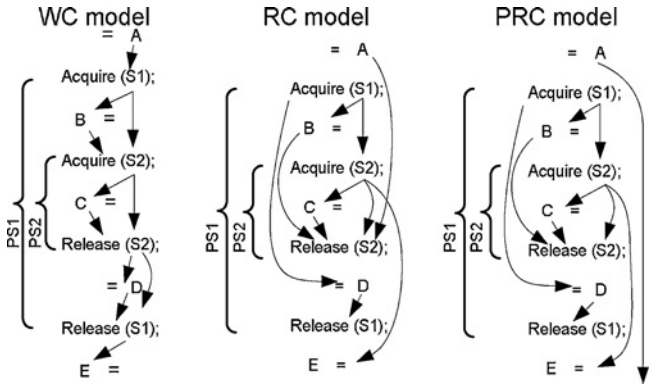

Fig. 6. Nested protected sections, protected data: B, C, D and unprotected data: A, E.

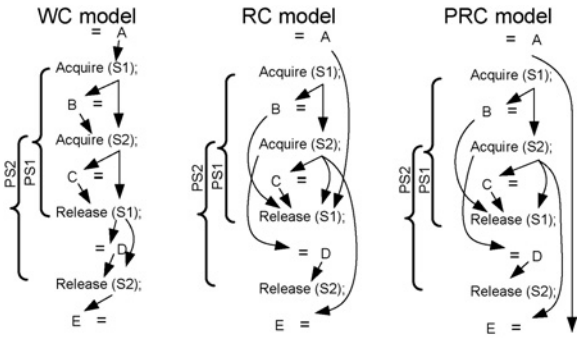

Fig. 7. Partially overlapped protected sections, protected data: B, C, D and unprotected data: A, E.

respectively, while the unprotected data operations on C can be reordered with the data operations of both the protected sections. The RC model does not allow the unprotected data operations on (A, C) to be reordered with the subsequent release and unprotected data operations, which is allowed under the PRC model.

Fig. 6 considers the code segments using nested protected sections. The PS2 is nested inside the PS1. The data operation on C is now a protected data operation compared to the non-overlapped protected case (Fig. 5). The ordering requirements under the WC model remain the same as those under the non-overlapped protected case, because the acquire and release operations are treated as one type of (Sync) operation. According to the RC model, the unprotected data operations on (A, E) now can be reordered with the data operations in both the protected sections under certain conditions. The unprotected data operation on A is not allowed to be reordered with the operations issued in that part of the code which comes after the PS2. Also, the unprotected data operation on E is not permitted to be reordered with the operations issued in that part of the code which comes before the PS2. The PRC model further relaxes the ordering constraints on the unprotected data operations on A to be reordered with the subsequent release and unprotected data operations.

Fig. 7 demonstrates the ordering requirements under the code segments using partially overlapped protected sections. A portion of the PS1 and PS2 overlaps with each other. The ordering constraints under the WC model still remain the same

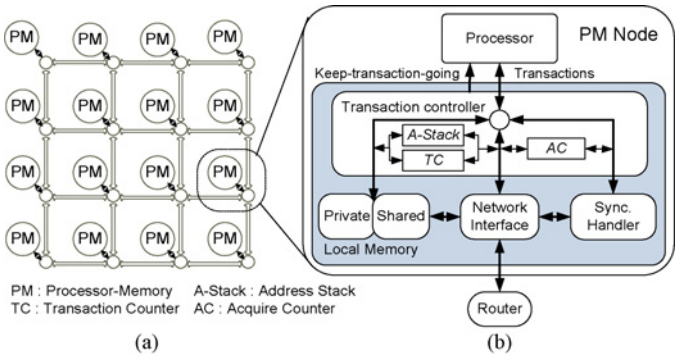

Fig. 8. (a) Homogeneous McNoC. (b) PM node.

as that in Fig. 5. The protected data operation on D is now part of the PS2 compared to the nested protected case (Fig. 6). Therefore, both under the RC and PRC models, the protected data operation on D is now restricted and cannot be reordered with the operations issued before the acquire S2.

I. Operations to the Same Memory Location

The weaker or relaxed consistency models permit outstanding memory operations which are reordered and overlapped with each other and the system performance is enhanced. However, to ensure the parallel program correctness, the operations to the same memory location must not be allowed to be reordered with respect to each other. For the correct behavior of the DSM systems, the operations issued by a processor to the same memory location must be constrained efficiently to accomplish as per program order.

IV. DSM Based McNoC Platform

A homogenous McNoC system is shown in Fig. 8(a). All nodes are interconnected via a packet-switched network. As demonstrated in Fig. 8(b), each processor-memory (PM) node consists of a processor, transaction controller (TCTRL), synchronization handler, network interface, and a local memory. Each local memory is partitioned into private and shared parts. All the physically distributed shared parts in the local memories form the DSM in a single address space. For a shared memory access two addressing schemes are used and a virtual-to-physical (VTP) address translation is required.

The platform uses 2-D 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 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 communication 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 packet format is given in Fig. 9. The packet has a total width of 97 bits including 37 bits header and 60 bits payload. It has total seven fields (source relative address, destination relative address, valid packet, hopcount, packet type, address, and data). The first two fields are used for the routing purposes.

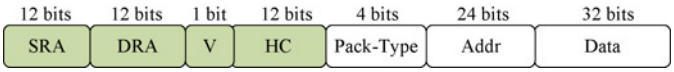

Fig. 9. Packet format.

The valid bit (V) indicates a valid or invalid packet. The hop count (HC) field is incremented at every hop in the network and is used to avoid the livelock. Packet Type differentiates among various types of packets in the system. The last two fields indicate the address/data of a memory transaction.

The platform also uses distributed locks in the synchronization handlers (SHs). The SH controls k 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 state is unlocked, then the acquire request changes its state to locked and an acknowledgement is sent back to the acquiring node. If the requested lock state 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. If the acquire and release requests arrive simultaneously, the lock remains in the locked status and the lock ownership is transferred from the releasing node to the acquiring node. A release request changes the lock's status to unlocked.

The platform uses a LEON3 processor [33] in each node of the network. The data cache system is disabled from the base processor for the independent implementation of memory models. The transaction controller (TCTRL) is a customized interface to integrate the processor with the rest of the system. It implements the key functions which may be required under any standard interface such as AXI [31] and OCP [32]. The TCTRL deals with the transactions from the processor and classifies them on the basis of address translation and memory mapping. It also communicates with the processor to control the flow of transactions. It transmits the transactions between the processor and memory system. One of the important functions of the interface is to implement the *memory consistency protocols*. It uses the hardware structures like (TC, AC, A-Stack) to realize the memory models. The TCTRL receives different types of transactions (read, write, memory barriers) with word granularity from the processor.

V. Realization of the Memory Consistency Models

A. SC Model

The SC model is realized [28] in the McNoC system by enforcing the required global orders as given in Fig. 3(a).

Program Order: is enforced by *stalling* the processor on the issuance of a memory operation till its completion. On the completion of a previously issued memory operation the next operation is issued in the program.

Sequential Order: is enforced by sequentially accessing the critical/shared memory locations among the multiprocessors by using the common lock [28].

B. TSO Model

The TSO model is realized [30] in the McNoC system by enforcing the required global orders as shown in Fig. 3(b).

***Write → Write*:** 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 incremented by the issuance of a write operation. It is decremented by the completion of a write operation. The WTC is not affected by the Sync and read operations. The issuance of a write operation is delayed by stalling the processor till the completion of previously issued outstanding write operation (e.g., $WTC = 0$). The processor is stalled by issuing an active low *keep-transaction-going* signal by the TCTRL in Fig. 8.

***Read → Read/Write*:** These global orders are enforced by stalling the processor on the issuance of a memory read operation till its completion by returned data. The issuances of the subsequent read and write operations are delayed till the completion of previously issued read operation.

Ordering constraints with respect to the Sync operations are also enforced. The memory operations are completed before the issuance of a Sync operation and vice versa. Upon the issuance of a Sync operation, 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 outstanding write operation, the issuance of a Sync operation is delayed till the completion of previously issued outstanding write operations (e.g., $WTC = 0$). Also, on the issuance of a Sync operation, the subsequent memory operations are delayed by stalling the processor till the successful completion of Sync operation.

C. PSO Model

The PSO model is realized [30] by enforcing the required global orders as given in Fig. 3(c). The ordering requirements under the PSO and TSO models are mostly similar except the PSO model further relaxes the ordering constraint on the memory operations in the case of a write followed by a write operation. According to the realization scheme of the PSO model, the issuance of a memory write operation is not delayed till $WTC = 0$. It allows multiple outstanding memory write operations in the network, which is not allowed under the TSO model. The independent memory write operations can be reordered with respect to each other. The rest of the realization scheme is similar to that described under the TSO model.

D. WC Model

The WC model is realized [28] in the McNoC platform by enforcing the required global orders as described in Fig. 4(a).

***Data → Sync*:** A transaction counter (TC) is used in each node of the network to keep track of the outstanding data (read, write) operations issued by a processor before a *Sync* operation. The TC is incremented by the issuance of a data operation. It is decremented by the completion of a data operation. The TC is not affected by the Sync operations. It is checked at the issuance of a Sync operation and the issuance of a Sync operation is delayed by stalling the processor till the completion of previously issued outstanding data operations.

***Sync → Data*:** To enforce this global order the processor is stalled upon the issuance of a Sync operation till its completion. The subsequent data operations are delayed for the completion of previously issued Sync operation.

***Sync → Sync*:** This global order is enforced by the sequential order on a lock in the multiprocessor system as discussed in the previous section. A Sync operation must be completed by a processor before the issuance of a next Sync operation.

E. RC Model

The RC model is realized [29] by enforcing the required global orders as illustrated in Fig. 4(b).

***Data → Release*:** A TC is used in each node of the network to keep track of outstanding data operations issued before a *release* operation. The TC is affected by the issuance and completion of the data operations. It is not affected by the acquire and release operations. The issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations (e.g., $TC = 0$). Note that, the TC under the RC model keeps track of the outstanding data operations issued between the two consecutive release points, while under the WC model; the TC keeps track of outstanding data operations issued between the two consecutive Sync points.

***Acquire → Data/Release*:** These global orders are enforced by stalling the processor on the issuance of an acquire operation till its successful completion. The subsequent data and release operations are delayed for the lock acquisition. The lock must be gained by a processor before entering to the critical section and also before releasing it.

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

F. PRC Model

The PRC model is realized [29] in the McNoC platform by enforcing the required global orders as given in Fig. 4(c). The ordering requirements under the PRC and RC models are similar except the release operation in the PRC model notifies only the preceding protected data operations.

***Protected Data → Release*:** A transaction counter (TC_{PD}) is used in each node to keep track of the outstanding protected data operations. The TC_{PD} is incremented on the issuance of a protected data operation. It is decremented by the completion of a protected data operation. The outstanding unprotected data operations are not tracked, and do not affect the TC_{PD}. The TC_{PD} is also not affected by the acquire and release operations. The issuance of a release operation checks the TC_{PD} and it is delayed till the completion of previously issued outstanding protected data operations.

***Acquire → Protected Data*:** The processor is stalled on the issuance of an acquire operation and the subsequent protected data operations are delayed for the lock acquisition. The rest of realization scheme is similar to that of the RC model.

***Classification of memory operations under the PRC model*:** The shared memory operations are categorized into local and remote operations using VTP address translation. Both the local and remote memory operations are further differentiated as data and synchronization operations on the basis of memory mapping. The synchronization operations are further distinguished as acquire and release operations by different commands/APIs. The classification of memory operations at this point is similar to that under the RC model. The PRC model further classifies the data operations as unprotected and protected data operations. An Acquire Counter (AC) is used in each node to this end. Initially, the AC is zero. The AC is incremented by the acquisition of a lock and decremented by releasing it. On the issuance of a data operation, the AC is checked. If it is zero, the data operation is unprotected, because either no lock is acquired yet or all prior acquired locks have been released. If it is non-zero, the data operation is protected, because either one or more lock(s) acquire exist already. Packet type field in the network protocol differentiates among the different types of memory operations (Fig. 9).

G. Operations to the Same Memory Location

In order to ensure the parallel program correctness, the operations to the same memory location are constrained to accomplish as per program order (refer to Section III-I). The TSO and PSO models use a hardware structure (write address stack) WA-Stack 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 memory write operations. Upon the issuance of a write operation, the address to be accessed by the write operation is pushed on the WA-Stack. Upon the completion of a write operation, the address is popped from the WA-Stack. The issuance of an operation checks the WA-Stack. If the address is on the WA-Stack, then there is an outstanding write operation issued to the same memory location. The issuance of the operation is then delayed until the same address is popped from the WA-Stack on the completion of previously issued write operations. Similarly, the WC, RC, and PRC models use Address Stack (A-Stack) in each node of the network to keep track of the addresses to be accessed by the previously issued outstanding data (read, write) operations. The A-Stack also tracks the addresses of outstanding read operations (in addition to the write operations) which are not tracked by the WA-Stack under the TSO and PSO models [30].

VI. Memory Models and system optimizations

A. Influence of the Compiler Optimizations

The compiler can reorder the memory operations to avoid the data dependencies. This could violate the sequentially consistent execution of a parallel program. The high level programming languages like Java, C, C#, and C++ use the volatile keyword in the declaration of variables which restricts the compiler optimizations on them. The WC, RC, and PRC models provide more space for the compiler to optimize the program. The WC model allows the compiler to statically reorder the data operations in between the two consecutive synchronization points. The RC model permits the compiler to optimize the program segment in between the two consecutive release points. The PRC model provides even more freedom to the compiler for the program optimization. The compiler can allocate the variables to the CPU registers. It can also eliminate the common sub-expressions by evaluating them to a single value which reduces the number of memory references. The compiler can also perform some operations on the loops. In brief, the PRC and RC models can exploit the most common compiler optimizations compared to the other memory models.

B. NoC Features and Performance of Memory Models

Different traffic patterns allow different overlapping and pipelining among the memory operations under relaxed memory models. The execution time of a program is dependent on the physical distance between the source and destination nodes in the network [30].

Different routing policies also affect the performance of memory models. For instance, the X–Y deterministic routing schemes do not allow reordering among the transactions of the same source to the same destination compared to the adaptive routing schemes. By using adaptive routing in our platform, we incorporate worst reordering situations. Thus, if an implementation of a memory model works well under an adaptive routing, it also works well under the deterministic routing. But note that, the benefits of relaxed memory models can be higher under adaptive routing, because more optimization potential can be exploited by the increasing reordering among the memory operations in the network.

The traffic is closed-loop under the implemented memory models; therefore, the injection rate cannot be arbitrary. It is under the control of the responses. Under the SC model, the injection rate will not have any impact on the performance improvement, because the issuance of an operation is controlled by the completion of a previously issued operation. Even with the outstanding transactions under the WC, RC and PRC models, there are synchronization points which will slow down the overall injection. However, increasing the injection rate will increase the outstanding transactions under a relaxed memory model and its performance could be further increased.

The network congestion can also affect the performance of different memory models. The memory operations could take longer time to complete in the congested network. Since we use the adaptive routing on-chip network, therefore, the memory operations might take alternative paths to reach their destinations. While increasing the system size, the network communication latency could also increase in general. This will somewhat produce similar effects as in more congested networks. The execution time under the SC model could be significantly increased under the congested network as it allows one outstanding operation issued by a processor at a time in the network. In contrast, the pipelining among the memory operations under PRC and RC models could be increased and the average memory access latency could be decreased.

C. Caches, Prefetching, and Transactional Memories

In order to get the benefits of data caches, the coherence protocols could be accommodated along with these independent

memory models. As a future work, the directory-based coherence protocols can be implemented on top of these memory models. This approach will allow for the independent optimal selection of the cache block size to reduce the coherence traffic, energy/power consumption and the directory overheads. However, the completion of memory transactions must be tracked via TC and A-Stack in the processor interface.

The modern processor systems prefer non-blocking caches (which deal with the multiple requests simultaneously) over the blocking caches (which deal one request at a time) [38]. Non-blocking caches can improve the system performance by servicing multiple outstanding cache miss requests. The WC, RC and PRC models allow outstanding operations which can exploit the architectures with non-blocking caches compared to the blocking caches. The PRC model could provide more space compared to the stricter models to utilize the systems using non-blocking caches.

The system efficiency might be increased by utilizing the processors in the time slots in which they are stalled due to the constraints imposed by the memory models. The DASH multiprocessor system [15] uses the non-binding prefetching, where the data is brought close to the processor vicinity somewhere in the data cache in advance which may be used by the processor later on. The fetched data are ensured to be coherent by using the directory-based coherence protocols.

Transactional memories simplify the parallel programming by atomic execution of a set of memory operations (e.g., transaction). Also, consistency and isolation requirements are ensured at the transaction level. Transactional memories suffer from the performance degradation due to the excessive amount of re-work as a result of frequent abortion of transactions. Consequently, the communication overhead is increased and the bandwidth is not utilized properly. This also limits the system scalability. Furthermore, as a future work, the transactional memories can be implemented along with the independent memory models. The ordering constraints on memory operations could be enforced by using the TC and A-Stack, while the transactional memories could target to simplify the parallel programming.

Each memory model maintains both the write atomicity and causality. Write atomicity over critical references is ensured by serial execution of the code segments in the protected sections. The writes are casually related as the read operations return the recently written values to the memory locations.

VII. Experiments and Results

A. Hardware Cost

The synthesis results of six designs in terms of NAND-gate equivalent and maximum frequency are given in Table I (optimized for the area). The difference in the area costs of the designs is mainly in the transaction controller (TCTRL), which uses transaction/acquire counter and address stack to implement the memory models. The TCTRL is synthesized with the stack depth of 16 up to 128 addresses each with 24 bits. The average result obtained with different configurations of TCTRL is shown in Table I. The transaction/acquire counter approximately consumes 635 gates each. In order to reduce the area overhead, size of the stack is kept optimal and is efficiently utilized. The addresses are popped from the stack continuously on the completion of operations in a pipelined manner. The area cost for the TSO, PSO WC, RC and PRC models are increased by 1.85% (373 gates), 0.65% (133 gates) 1.0% (210 gates), 0.65% (131 gates), and 3.8% (764 gates) over the SC model in the TCTRL. The NI and TCTRL under the PRC model consume the most area due to the additional logic needed for further classification of the data operations. The switch and synchronization handler under all the memory models consume 13.24 and 3.76 kilo-gates, respectively. In all the 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	
	A	F	A	F	A	F
NI	49.99	1.25	49.99	1.25	49.99	1.25
TCTRL	20.00	0.5	20.37	0.5	20.13	0.5
Total	69.99		70.36		70.12	
	WC Model		RC Model		PRC Model	
	A	F	A	F	A	F
NI	49.99	1.25	49.99	1.25	50.88	1.25
TCTRL	20.20	0.5	20.13	0.5	20.76	0.5
Total	70.19		70.12		71.64	

A: Area (Kilo Nand Gates), F: Frequency (GHz).

TABLE II
PLATFORM CONFIGURATION PARAMETERS

Sub-system	**Descriptions/Parameters**
Core processor	LEON3, synthesizable VHDL model, 32-bit, compliant with the SPARC V8 architecture
Network interface (NI)	Full duplex, packetization, de-packetization, buffering capacity 64 packets, message passing, network protocol (97 bits, 7 fields)
Network (Nostrum)	Buffer-less, on-chip, configurable, packet-switched, 2-D regular mesh topology, deflection routing policy (adaptive routing)
Memory (DSM)	DSM organization, 16 MB shared memory in each node, dual ported
Sync handler	Distributed, 256 locks in each node, dual ported
Transaction controller	Distributed, TC/AC each 32 bits, A/WA-Stack capacity of 64 virtual addresses each 24 bits

B. Experimental Setup

For the experiments, a configurable cycle true simulation McNoC platform is constructed in VHDL (Fig. 8). The caches are disabled from the LEON3 processors in the experiments, as they are neutral for the evaluation of the memory models. The platform configuration parameters are given in Table II.

For the McNoC systems, communication centric benchmarks are required to evaluate the communication aspect of the network. The SPEC, SPLASH-2, and PARSEC benchmarks suits [39] have computation intensive programs developed for the high-performance computing systems. These benchmarks cannot be utilized directly for the scalability analysis of memory models in the McNoC systems, because they are compute intensive and the potential parallelism of the network could not be fully exploited under the relaxed memory models. Thus, we have developed both the synthetic and application workloads to analyze the scalability of memory models in the McNoC systems [35]. The developed applications are light-weight, specific to NoC/embedded architectures, and communication centric. The input problem size and memory

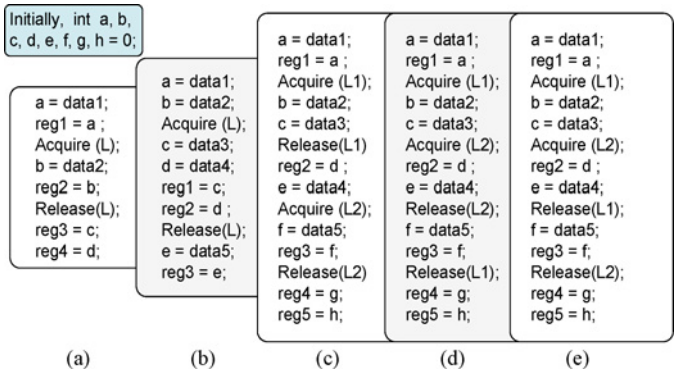

Fig. 10. Trans. Sequences: (a) WL1; (b) WL2; (c) WL3; (d) WL4; (e) WL5.

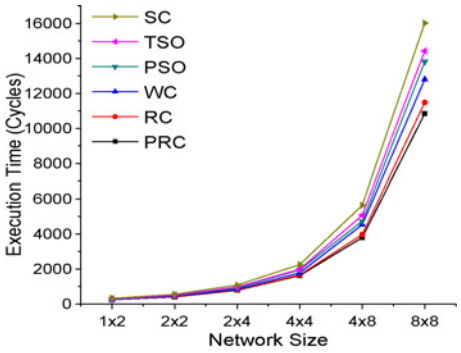

Fig. 11. Average SETs for WL1-WL5.

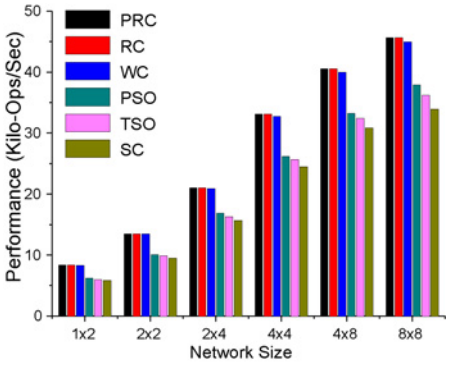

Fig. 12. Performance under angle conversion application.

instruction density can be varied. Different traffic patterns are generated for the memory operations. The developed workloads are ranging from a simple to complex workloads like wavefront computations and are chosen from different application areas. The synchronization intensive wavefront computations are widely applied in the scientific computing, dynamic programming algorithms, and particle physics [37]. We have chosen some representative workloads to derive meaningful conclusions from the experiments.

In the experiments, we study the scalability of six memory models in the McNoC systems (Fig. 8). The workloads are mapped on the networks of different sizes. The performance metrics like execution time, performance and speedup are evaluated against the network size. The execution time of a workload is the time from the start of execution on the first processor to the end of execution on the last processor. The performance is the reciprocal of the execution time (measured in kilo operations per second). The speedup is the ratio of execution time of the single core and multicore system.

C. Synthetic Workloads

The performance of six memory models is evaluated with different synthetic workloads (WL1-WL5) as shown in Fig. 10. 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; thus, when the number of cores is doubled, the total amount of work is also doubled. WL1 contains data and synchronization operations. It has both the cases of a write followed by a read and a read followed by a read operation. WL2 contains a write followed by a write, a write followed by a read and a read followed by a read sequences. WL3 has, in addition, a read followed by a write operation and uses two non-overlapped protected/critical sections. WL4 uses nested protected sections protected under two different locks. WL5 uses partially overlapped protected sections, which are protected under two different locks. For the synchronization and protected-data operations, hotspot traffic pattern is generated. For the unprotected-data operations, uniform random traffic pattern is used.

The synthetic workload execution times (SETs) are compared for the six memory models in Fig. 11. The SC model is used as the baseline model. The system size is increased from 2 to 64-cores. From the network perspective, 1×1 (single core) is meaningless, therefore, we have not considered it. As the system scales up, the SETs quickly increase under all the memory models because of the increasing network traffic, congestion and waiting time to acquire a lock. The SETs are lower under the relaxed memory models compared to the strict SC model due to reordering and relaxation in the memory operations. The average SETs under the PRC, RC, WC, PSO and TSO models in the 8×8 network are reduced by 32.3%, 28.3%, 20.1%, 13.8%, and 9.9% over the SC model, respectively. The SETs under the PRC and RC models are reduced more under the WL2 due to the issuance of more data operations before the release operation.

D. Application Workloads

1) *Angle Conversion:* The application converts the input data vector of degrees into output data vector of radians. The input data vector of 128 elements is used. The input and output vectors are stored in the DSM and uniform random traffic pattern is used.

As illustrated in Fig. 12, the performance of six memory models under the angle conversion application increases as the system size is scaled up. This is due to the division of computation cost in the larger networks. The computation cost is perfectly divided among the identical processors in the larger systems. The PRC and RC models further improve the performance compared to the stricter models by reordering and overlapping the memory operations in the network. The performance under the PRC, RC, WC, PSO, TSO, and SC models in the 64-cores system are 8.3, 8.3, 8.1, 6.8, 6.5, and 6.1 times of that in the single core system, respectively.

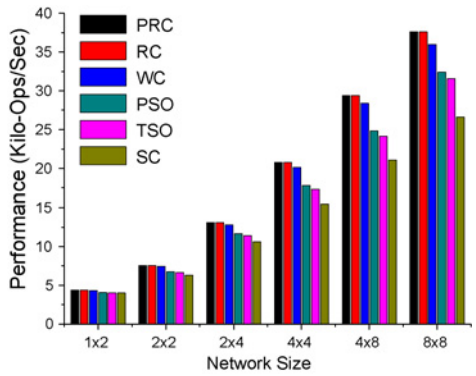

Fig. 13. Performance under bit count application.

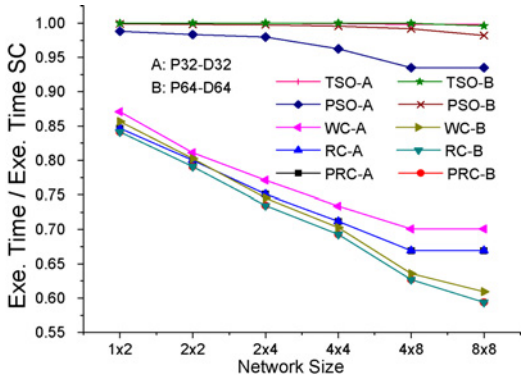

Fig. 14. Pattern search: Ratio (execution time of relaxed models/SC model).

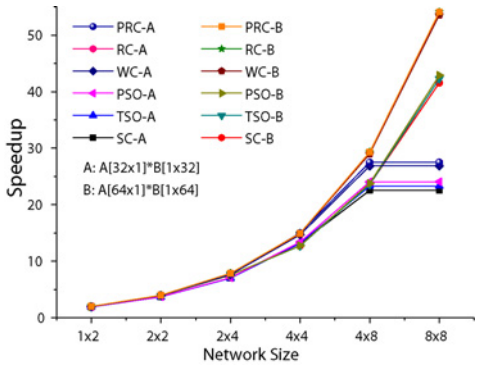

Fig. 15. Speedup under matrix multiplication applications.

2) *Bit Count:* The application analyzes the input data vector and calculates the number of set bits in each integer data item. The input data vector with 512 elements is initialized, read, analyzed and the resultant output vector is stored in the DSM. Each node operates on the data items in a randomly selected node.

For the bit count application, the increase in the performance (Fig. 13) under the relaxed memory models over the SC model is higher in contrast to the angle conversion application. This is because of the lower computation-to-communication ratio. The computation time per input data item under the bit count application is less (21 cycles) compared to the angle conversion application (31 cycles). Due to less computation-to-communication ratio, the communication becomes significant under the bit count application. The PRC and RC models efficiently handle the communication overhead by allowing more outstanding operations in the network which are pipelined and overlapped with each other. The increase in performance under the relaxed models over the SC model is 13.2% to 20.3% higher than the angle conversion application in the 64-cores systems (Fig. 12).

3) *Pattern Search:* The application searches data patterns (P) against the data elements (D), which are initialized in the DSM. Two different cases are simulated using different combinations of the patterns and data elements. The system size is increased from 2 to 64-cores. *P32-D32:* when 32 patterns and 32 data elements are mapped on the 8×8 network, only 32 nodes participate in the computations. *P64-D64:* For the 64 patterns and 64 data elements, one pattern and one data element is mapped on the 8×8 network and each node is involved in the computation. The outputs are the number of times that the patterns appear in the data elements, which are stored in the local node.

The execution times of relaxed memory models relative to the SC model under the pattern search application are given in Fig. 14. As the system scales up, relative execution times under the PRC and RC models decrease more due to the additional pipelining among the memory operations. At some point (e.g., network size) the decrease in the relative time flattens off. It depends on the problem size and its match to the architecture, when exactly this leveling off occurs. For the (*P64-D64*) problem, the relative execution time constantly decreases under all the memory models as the network size increases, because the problem size fits well into the increasing size of the network compared to the (*P32-D32*) problem. A better and scalable behavior (e.g., continuous performance gain) is observed under the (*P64-D64*) problem, where the computation is further divided in the 8×8 network.

4) *Matrix Multiplication:* The multiplication of two matrices (A, B) produces a matrix C. All the matrices are initialized in the DSM. Two different sized matrices are multiplied. The $A[32 \times 1]$ multiplies $B[1 \times 32]$ produces $C[32 \times 32]$, and $A[64 \times 1]$ cross $B[1 \times 64]$ resulting $C[64 \times 64]$. The network size is increased from 2 to 64-nodes. When $A[32 \times 1]$ and $B[1 \times 32]$ matrices are mapped in the 8×8 network, only 32 nodes perform the computations. When $A[64\times1]$ and $B[1\times64]$ are mapped in the 8×8 system, all the nodes are involved in the computation. Each node operates on that part of matrix-A which is in the local node, and the entire matrix-B which is distributed in the network. The output results are written into the shared memory of the same node.

Fig. 15 illustrates the speedup of memory models under the matrix multiplication application. The speedup is lower under case-A due to smaller problem compared to the case-B. For the case-A, the speedup levels off after 32 nodes up to 64 nodes, because the potential parallelism in the 8×8 network is not exploited by the chosen problem size. On the other hand, it is utilized under the case-B, where computation is further divided in the 8×8 network and all the nodes are involved in the computation. The problem size like case-B scales well and continuously obtain the benefit from the higher parallelism of the relaxed memory models compared to the SC model.

All these applications are data intensive and do not use the synchronization operations. Therefore, the performance of

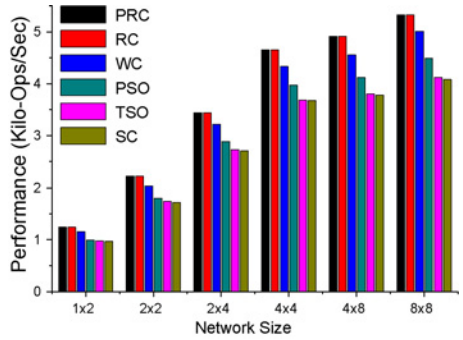

Fig. 16. Performance under WFCA application.

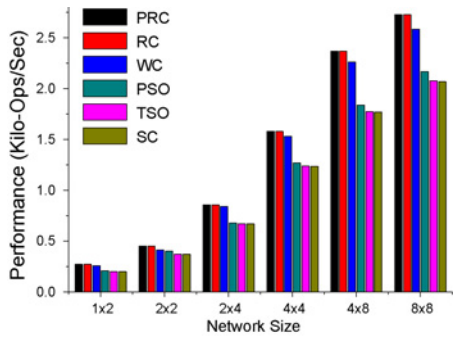

Fig. 17. Performance under WFCB application.

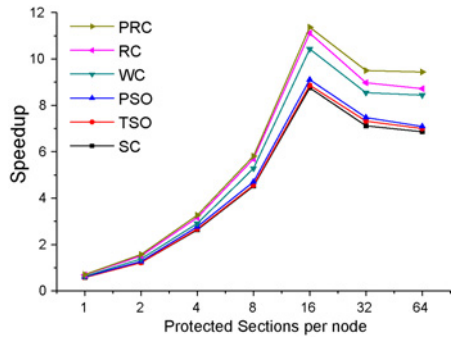

Fig. 18. Speedup under WFB-UPD application.

WC, RC and PRC models are almost the same, but more than the PSO, TSO, and SC models due to the additional reordering in the data operations. The next experiments use the wavefront computation applications which contain both synchronization and data operations compared to the previous applications.

5) *Wavefront Computation-A (WFCA):* The WFCA uses (16×64) data computations in 16 different *partially overlapped* protected sections. In two cores system, each node performs (16×32) data computations. Each computation uses the data produced in the same and previous protected sections on the same node, and also the data computed on the previous node. To ensure the data availability of the previous node, each node uses 16 additional lock-acquires at the start of each protected section to the same locks. When WFCA is mapped on the four cores system, each node performs half of the data computations (16×16). The number of protected sections remains the same. Likewise, this trend goes up to 64-cores system, where each core performs the (16×1) computations. The ratio of data/synchronization operations decreases as the network size is increased.

As given in Fig. 16, the performance gap between the PRC/RC and WC models is increased under the synchronization intensive WFCA application. This is due to the fact that the PRC and RC models allow the reordering and overlapping among the data and synchronization operations, which is not allowed under the WC model. The performance (Fig. 16) under the memory models increase quickly in the small networks up to 16 nodes due to a high ratio of the data/synchronization operations, while onward, the performance rises slowly due to the increasing synchronization overhead among the cores in the larger networks.

6) *Wavefront Computation-B (WFCB):* The WFCB uses (32×64) data computations in (32×64) different nested protected sections. In two cores system, each node performs (32×32) data computations. Each computation uses the data computed under the previous nested protected section on the same node and the data computed on the previous node. The previous node's data are made available by using nested acquires operations at the start of each protected section to the same locks. When WFCB is mapped on the four cores system, each node performs half of the data computations (32×16) in the nested protected sections. Similarly, this trend goes up to 64-cores system, where each core performs the (32×1) computations. The ratio of data/synchronization operations remains the same as the network size is increased.

For the WFCB application, the increase in the performance (Fig. 17) under the relaxed models over the SC model is slightly higher in contrast to the WFCA application. This is because of more protected sections (computations) per node which are used under the WFCB application. Note that, the WFCB uses 32, while WFCA uses 16 protected sections per node. The considerable amount of computation is parallelized in the larger network under the WFCB application. Also, the synchronization overhead does not arise in the larger networks. Therefore, the performance under the memory models increase quickly in the larger networks compared to the WFCA application. Both the WFCA and WFCB applications do not use the unprotected data operations. Thus, the performance gap between the PRC and RC models is insignificant.

In order to observe the performance gain of the PRC model over RC model, WFCB is supplemented with the unprotected data operations (WFCB-UPD) in the 8×8 system. The number of the unprotected data operations is varied in proportion to the number of protected sections per node from 1 to 64. For the single protected section per node, there are a total (1×64) unprotected data computations. For the 64 protected sections per node, there are a total (64×64) unprotected data computations. All these data computations in the unprotected sections are independent of each other.

As can be seen in Fig. 18, the performance gain of the PRC model over the RC model is clearly visible. This is due to the interleaving unprotected data operations among the protected sections. For the 64 protected sections per node, the execution time under the WFCB-UPD for the PRC model is reduced by 7.6% over the RC model and 27.3% over the SC model. The speedup increases until 16 protected sections per node

TABLE III
AVERAGE EXECUTION TIME RELATIVE TO SC IN PERCENTAGE

Applications	SC	TSO	PSO	WC	RC	PRC
Angle Conv.	100	93.68	89.61	75.49	74.34	74.34
Pattern Search	100	99.6	96.65	63.94	61.86	61.86
Bit Count	100	84.24	82.12	73.97	70.78	70.78
Matrix Multp.	100	97.74	95.77	78.23	77.13	77.13
WFCA	100	98.93	90.77	81.41	76.52	76.52
WFCB	100	99.54	95.41	79.92	75.79	75.79
WFCB-UPD	100	98.11	96.8	81.32	78.71	72.73
Average	100	95.98	92.44	76.32	73.59	72.73

and then decreases onward (Fig. 18). This is due to the fact, when the number of protected sections per node increases, the amount of computation also increases due to the incorporation of unprotected data operations.

7) *Average of All Applications:* The average execution times under all the applications for the relaxed memory models relative to the SC model in percentage are given in Table III. For the 64-cores systems, the average execution times under the PRC, RC, WC, PSO, and TSO models compared to the SC model are reduced by 25.9%, 23.2%, 19.9%, 4.9%, and 1.7%, respectively.

E. Summarizing the Scalability Analysis and Trends

To sum up, in all experiments, the execution time of the PRC and RC models has been between 50% and 100% of the SC model. The specific numbers are highly sensitive to the application and depend on how well it matches the platform. It also depends on the computation-to-communication ratio, problem size, traffic patterns and synchronization overheads in the system. However, the observed trends suggest that the PRC and RC scales inherently better with the network size than the stricter models. As shown in Fig. 19, the execution time under the PRC and RC models relative to the SC model decreases more compared to the other memory models as the network size grows. As long as the speedup increases, the benefits of the relaxed memory models over the SC model also increase, but when the nature of the problem makes it harder to exploit the additional parallelism, the benefits of the PRC and RC models over the SC model saturate as well. However, problems that scale well, like the B-matrix multiplication and B-pattern search problems continue to gain more benefits from the higher level of parallelism that the PRC and RC models offer compared to the rest of the memory models. Thus, we conclude that the performance increase of the PRC and RC models over SC model can be significantly higher than 50% as observed in the results when the system size is further increased.

From the experiments, it can also be observed that the reduction in relative execution times under WC, RC and PRC models over the SC model is expected to be 3.33 percentage points approximately for each doubling of core count up to 128-cores. These relaxed memory models could exploit the increasing parallelism of the larger networks beyond the 64-cores by reordering and pipelining the memory operations. Similarly, for the TSO and PSO models Fig. 19 suggests that this can be projected to 1.0 percentage points for each doubling of core count up to 128-cores. In general, the performance gain

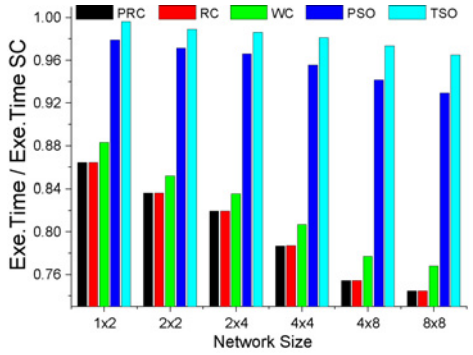

Fig. 19. Average relative execution time under all applications.

of the relaxed memory models can be even higher than the experimental values when the system size is further increased.

In the presence of caches and cache coherence protocols, the extra coherence traffic under a relaxed memory model could be pipelined in the network and the average latency of memory transactions might be significantly decreased compared to the strict SC model. However, the exact figure is very hard to calculate as it depends on several factors like cache write policy, hit/miss ratio, and the exercised cache coherence protocols. The execution time of different memory models would be scaled differently. A relaxed memory model will get more benefits by overlapping the coherence traffic compared to the strict SC model. Consequently, the same trend is expected to be observed in the relative execution times of the relaxed memory models while using both the cache coherence and consistency protocols.

VIII. CONCLUSION

We analyzed the scalability of memory models in the DSM-based McNoC systems up to 64-cores. These memory models were realized using a transaction counter and an address stack-based novel approaches. A comprehensive study of six different memory models was presented. The influence of system optimizations on the memory models was also discussed. The experimental results showed that under a set of synthetic workloads, the average execution time for the PRC, RC, WC, PSO, and TSO models in the 8×8 network is reduced by 32.3%, 28.3%, 20.1%, 13.8% and 9.9% over the SC model, respectively. For the application workloads, as long as the system size scales up, the execution time under the relaxed memory models decreases relative to the SC model. It depends on the scaling of the problem size and how efficiently the PRC and RC models are utilized compared to the SC model. The performance gains of the PRC and RC models over the SC model are expected to be higher than 50% as observed in the results, when the network size is further increased. The hardware cost for the relaxed memory models is increased by less than 4% over the SC model at the processor interface.

REFERENCES

[1] A. Jantsch, X. Chen, A. Naeem, Y. Zhang, S. Penolazzi, and Z. Lu, "Memory architecture and management in an NoC platform," in *Scalable Multicore Architectures: Design Methodologies and Tools*, A. Jantsch and D. Soudris, Eds. Berlin, Germany: Springer, 2011.

[2] S. V. Adve and K. Gharachorloo, *Shared Memory Consistency Models: A Tutorial*, Report no. 95/7, Digital Western Research Lab., Palo Alto, CA, 1995.

[3] L. Lamport, "How to make a multiprocessors computer that correctly executes multiprocessor programs," *IEEE Trans. Comput.*, vol. C-28. no. 9, pp. 690–691, Sep. 1979.

[4] D. E. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. San Mateo, CA: Morgan Kaufmann Publishers, 1999.

[5] A. Jantsch, *The Nostrum NoC* [Online]. Available: http://www.ict.kth.se/nostrum

[6] D. Weaver and T. Germond, Eds. *The SPARC Architecture Manual*, New Jersey: Prentice-Hall, 1994, SPARC Int., Ver. 9.

[7] P. Sewell, S. Sarkar, S. Owens, F. Z. Nardelli, and M. O. Myreen, "x86-TSO: A rigorous and usable programmer's model for x86 multiprocessors," *Commun. ACM*, vol. 53, no. 7, pp. 89–97, 2010.

[8] M. Dubois, C. Scheurich, and F. Briggs, "Memory access buffering in multiprocessors," in *Proc. Ann. Int. Symp. Comp. Arch.*, 1986, pp. 320–328.

[9] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy, "Memory consistency and event ordering in scalable shared memory multiprocessors," *Comput. Architecture News*, vol. 18, no. 2, pp. 15–26, Jun. 1990.

[10] D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, San Rafael, CA: Morgan and Claypool Publishers, 2011.

[11] J. B. Carter, J. K. Bennett, and W. Zwaenepoel, "Implementation and performance of Munin," in *Proc. 13th ACM SOSP*, Oct. 1991, pp. 152–164.

[12] P. Keleher, A. L. Cox, and W. Zwaenepoel, "Lazy consistency for software distributed shared memory," in *Proc. 19th Annu. Symp. Comput. Architecture*, May 1992, pp. 13–21.

[13] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon, "The midway distributed shared memory system," in *Proc. IEEE COMPCON Conf.*, Feb. 1993, pp. 528–537.

[14] L. Iftode, J. P. Singh, and K. Li, "Scope consistency: A bridge between release consistency and entry consistency," in *Proc. 8th Annu. Symp. Parallel Algorithms Archit.*, Jun. 1996, pp. 277–287.

[15] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam, "The Stanford dash multiprocessor," *IEEE Comput.*, vol. 25, no. 3, pp. 63–79, Mar. 1992.

[16] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *Proc. 42nd Int. Symp. MICRO*, 2009, pp. 423-434.

[17] M. Ferdman, P. L. Kamran, K. Balet, and B. Falsafi, "Cuckoo directory: A scalable directory for many-core systems," in *Proc. Int. Symp. High Performance Comp. Archit.*, Feb. 2011, pp. 169–180.

[18] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in *Proc. 30th Annu. ISCA*, 2003, pp. 182–193.

[19] G. R. Gao and V. Sarkar, "Location consistency—a new memory model and cache consistency protocol," *IEEE Trans. Comput.*, vol. 49, no. 8, pp. 798–813, Aug. 2000.

[20] B. F. Romanescu, A. R. Lebeck, and D. J. Sorin, "Address translation aware memory consistency," *IEEE Micro*, vol. 31, no. 1, pp. 109–118, Jan.–Feb. 2011.

[21] A. Arvind and J-Willem Maessen, "Memory model = instruction reordering + store atomicity," in *Proc. 33rd ISCA*, 2006, pp. 29–40.

[22] M. Herlihy and J. E. B. Moss, "Transactional memory: Architectural support for lock-free data structures," in *Proc. 20th Annu. ISCA*, 1993, pp. 289–300.

[23] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum, "Hybrid transactional memory," in *Proc. 12th Int. Cnf. ASPLOS*, Oct. 2006, pp. 336–346.

[24] J. Manson, W. Pugh, and S. Adve, "The Java memory model," in *Proc. ACM Symp. Principles Programming Languages*, Jan. 2005, pp. 378–391.

[25] F. Petrot, A. Greiner, and P. Gomez, "On cache coherence and memory consistency issues in NoC based shared memory multiprocessor SoC architectures," in *Proc. 9th Euromicro Conf. Digital Syst. Des.*, 2006, pp. 53–60.

[26] A. Hansson and K. Goossens, "An on-chip interconnect and protocol stack for multiple communication paradigms and programming models," in *Proc. CODES+ISSS*, 2009, pp. 99–108.

[27] J. W. Van den Brand and M. Bekooij, "Streaming consistency: A model for efficient MPSoC design," in *Proc. 10th Euromicro Conf. Digital Syst. Des.*, Aug. 2007, pp. 27–34.

[28] 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 *Proc.* 16th *Asia South Pacific Des. Autom. Conf.*, Jan. 2011.

[29] A. Naeem, A. Jantsch, X. Chen, and Z. Lu, "Realization and scalability of release and protected release consistency models in NoC based systems," in *Proc. Euromicro Conf. DSD*, Aug.–Sep 2011, pp. 47–54.

[30] A. Naeem, A. Jantsch, and Z. Lu, "Architecture support and comparison of three memory consistency models in NoC based systems," in *Proc. Euromicro Conf. DSD*, 2012, pp. 304–311.

[31] ARM Limited. (2004). *AMBA AXI Protocol Specification v1.0* [Online]. Available: http://infocenter.arm.com

[32] OCP International Partnership. (2007). *OCP Specification 2.2* [Online]. Available: http://www.ocpip.org/get_the_specifications.php

[33] A. Gaisler. *Leon3 32-Bit Processor Core* [Online]. Available: http://www.gaisler.com/doc/Leon3%20Grlib%20folder.pdf

[34] A. Naeem, X. Chen, Z. Lu, and A. Jantsch, "Scalability of relaxed consistency models in NoC based multicore architectures," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 5, pp. 8–15, Apr. 2010.

[35] C. Grecu, A. Ivanov, A. Jantsch, P. P. Pande, E. Salminen, U. Y. Ogras, and R. Marculescu, "Towards open network-on-chip benchmarks," in *Proc. NOCS*, May 2007, p. 205.

[36] A. Naeem, A. Jantsch, and Z. Lu, "Scalability analysis of release and sequential consistency models in NoC based multicore systems," in *Proc. IEEE Int. Symp. SOC*, Oct. 2012, pp. 1–7.

[37] E. Christopher Lewis and L. Snyder, "Pipelining wavefront computations: Experiences and performance," in *Proc. 15th IEEE Int. Workshop HIPS*, Apr. 2000, pp. 261–268.

[38] K. Aasaraai and A. Moshovos, "An efficient non-blocking data cache for soft processors," in *Proc. Int. Conf. Reconfigurable Computing FPGAs*, Dec. 2010, pp. 19–24.

[39] C. Bienia, S. Kumar, and K. Li, "PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *Proc. IEEE Int. Symp. Workload Characterization*, Sep. 2008, pp. 47–56.

Abdul Naeem received the B.Sc. degree in mathematics from the University of Peshawar, Peshawar, Pakistan, in 1999, the B.Sc. degree in electrical engineering from the University of Engineering and Technology (UET), Peshawar, in 2001, and the M.Sc. degree in electrical engineering from the UET, Taxila, Pakistan, in 2004. He is currently pursuing the Ph.D. degree with the Department of Electronic Systems, School of Information and Communication Technology, KTH-Royal Institute of Technology, Stockholm, Sweden.

His full biography can be found in *Marquis Who's Who in the World*, 30th Pearl Anniversary Edition 2013. His current research interests include memory consistency and coherence, systems-on-chip, and network-on-chip-based multicore systems.

Axel Jantsch (M'97) received the Dipl.Ing. and Dr.Tech. degrees from the Technical University of Vienna, Vienna, Austria, in 1988 and 1992, respectively. Between 1993 and 1995 he received the Alfred Schrödinger Scholarship from the Austrian Science Foundation as a Guest Researcher at the KTH-Royal Institute of Technology.

From 1995 to 1997, he was with Siemens Austria, Vienna, Austria as a System Validation Engineer. Since 1997, he has been with the KTH, Stockholm, Sweden. Since December 2002 he has been a Full Professor in electronic system design. At KTH, he heads a number of research projects involving 10 Ph.D. students, in the areas of system-level specification, design, synthesis, validation, and networks on chip. He has published over 200 papers in international conferences and journals and two books in the areas of VLSI design and synthesis, system-level specification, modeling and validation, HW/SW codesign, reconfigurable computing, and networks on chip.

Dr. Jantsch has served on a number of TPCs of international conferences such as FDL, DATE, CODES+ISSS, SOC, HDLCON, and others. He has been a TPC Chair of SSDL/FDL 2000, TPC Cochair of CODES+ISSS 2004, and General Cochair of CODES+ISSS 2005, and the TPC Cochair of NOCS 2009. From 2002 to 2007, he was a Subject Area Editor for the Journal of System Architecture.

Zhonghai Lu (M'05) received the B.Sc. degree in radio and electronics from Beijing Normal University, Beijing, China, in 1989, and the M.Sc. degree in system-on-chip design and the Ph.D. degree in electronic and computer system design from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2002 and 2007, respectively.

After completing the Ph.D. degree, he conducted two years of post-doctoral research with KTH. From 1989 to 2000, he worked extensively as an Engineer in the area of electronic and embedded systems. He is currently an Associate Professor with the Department of Electronic Systems, School for Information and Communication Technology, KTH. His current research interests include computing architectures, interconnection networks, performance analysis, and design automation. He has published over 100 peer-reviewed papers in these areas.