Scalability Analysis of Release and Sequential Consistency Models in NoC based Multicore 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 analyze the scalability of the Release Consistency (RC) and Sequential Consistency (SC) models which are realized in the Network-on-Chip (NoC) based distributed shared memory multicore systems. The analysis is performed on the basis of workloads mapped on the different sizes of networks with different data sets. The experiments use a configurable platform based on a 2D mesh NoC using deflection routing algorithm. The results show that under the synthetic workloads using different distributed locks, the performance of the RC model is increased by 17.6% to 54.6% over the SC model in the 64-cores system. For the application workloads, as the network size grows from 1 to 64 cores, the execution time under the RC model decreases relative to the SC model which depends on the application and its match to the architecture. The performance improvement of the RC model over the SC model tends to be higher than 50% observed in the experiments, when the system is further scaled up.**

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

I. INTRODUCTION

Parallelization as a key means to enhance performance and reduce power can be achieved at the computation, communication and memory architectures in the system [1]. The distributed nature of the Network-on-Chip (NoC) based systems can be exploited by using on-chip Distributed Shared Memory (DSM) architectures. Since the shared memory operations may be reordered in the network, the DSM system may show unexpected behavior. Memory consistency defines the execution order of the shared memory operations for the correct behavior of the DSM systems. Different memory consistency models [2] enforce different ordering constraints on the shared memory operations, implying different system performance. The *Sequential Consistency* (SC) model is a strict consistency model [3] and it cannot exploit the system optimizations. Therefore, several *relaxed* consistency models [2][4-7] are proposed to alleviate the ordering constraints on the memory operations to exploit these system optimizations.

Memory consistency and cache coherence are two main issues in the DSM systems. The former issue arises due to the unconstrained shared memory operations, while the later issue is due to the different cached copies of the same shared data in the DSM systems. Different memory consistency models and cache coherence protocols are proposed to handle these issues. In some situations, when a cache is not used like in the hard real time applications or when these two problems have different requirements on the size of the cache block and the consistency object, independent implementation schemes for the two problems are preferred [8][9].

This paper analyzes the scalability of the RC and SC models [8][9] which are realized in the Multicore NoC (McNoC) systems. The key performance metrics like *execution time, performance, speedup, overhead* and *efficiency* are evaluated as a function of the network size. The scaling behavior of both the RC and SC models are analyzed by mapping the workloads on the different sizes of the network with different data sets and also on the basis of application types and the system design perceptions (e.g. distributed locks and DSM architectures).

For the experiments, a configurable McNoC platform is used with distributed locks, DSM and 2D mesh *Nostrum* NoC [10] using a deflection routing policy. The *scalability study* of the RC and SC models is performed in the McNoC systems with 1 to 64-cores. The experimental results show the scaling behavior of the RC and SC models in the McNoC systems.

The rest of the paper is organized as follows. The next section overviews the related work. In section III, the SC and RC models, DSM based McNoC platform and the implementation of the SC and RC models are discussed. The simulation results and scalability analysis of the RC and SC models up to 64-cores systems are presented in section IV. In section V, our contributions are summarized.

II. RELATED WORK

In general multiprocessors DSM systems, several memory consistency models are discussed in the literature [2-7]. Adve et al. [2] discussed the memory consistency models from the system optimizations point of view. The SC model enforces a *total* order on the shared memory operations [3]. The *total store ordering* model relaxes the ordering constraint in the case of a *write followed by a read* operation. The *partial store ordering* model provides an additional relaxation among the *write* operations [2]. The *Weak Consistency* (WC) model [5] classifies the shared memory operations as *data* and *synchronization* operations. The data (read, write) operations issued between the two consecutive synchronization points can be reordered with each other. The RC model [6] further classifies the synchronization operations as *acquire* and *release* operations. The RC model is implemented in the DASH project [7] which depends on the *directory* based cache coherence protocol [11]. However, the directory based coherence protocols have some issues in the larger networks like, i.e., extra coherence traffic, directory overhead, additional latencies and complexities. In NoC based DSM systems, the proposed mechanism in [12] is very restrictive and allows one outstanding transaction of an initiator at a time in the network. The streaming consistency [13] is based on the software cache coherence protocol. However, polling the circular buffer at

each request level is not a scalable approach. A protocol stack for on-chip interconnects is proposed [14] at different levels of the SoC design. They briefly outline the mechanisms to implement the RC model at the memory-mapped stack. But, the implementation detail is not discussed. The AXI [16] and OCP [18] protocols enforce the *ordering models* by using transactions IDs and thread IDs, respectively. In [16], transactions of the same master with *different* IDs can be reordered, but transactions with the *same* ID are not allowed to be reordered. In [18], *tagged* transactions of the same master using thread IDs are allowed to be reordered, but *non-tagged* transactions are strictly ordered. In [8], the SC model is realized in the McNoC systems by *stalling* the processor on the issuance of an operation till its completion. In [15], two *Transaction Counters* (*TCs*) based approach is adopted to realize the RC model in the McNoC systems. The *TC1* and *TC2* are used to keep track of the outstanding data operations issued in the non-critical and critical sections, respectively. In [9], a single *TC* based approach is used to realize the RC model in the McNoC systems. In this paper, we further analyze the scalability of the RC model [9] and SC model [8] in the McNoC systems.

III. SC AND RC MODELS IN NOC BASED SYSTEMS

The ordering constraints to be enforced on the shared memory operations under the SC and RC models are given in Figure 1(a) and (b). An *arrow* between the two variables indicates an ordering constraint between the operations on these variables. For example, G→H indicates that an operation on variable G is followed by an operation on variable H in the program and these two operations are not allowed to be reordered with each other. The variables to the left side of the assignment operators are written and those to the right are read.

A. SC Model

According to the SC model [3][8] (Figure 1(a)), the shared memory operations are completed in the *program order*. The *sequential order* is maintained by interleaving operations on lock (L) among processors in the system. The SC model enforces the global orders (Figure 1(c)) on the shared memory operations. We refer to these global orders in the later part.

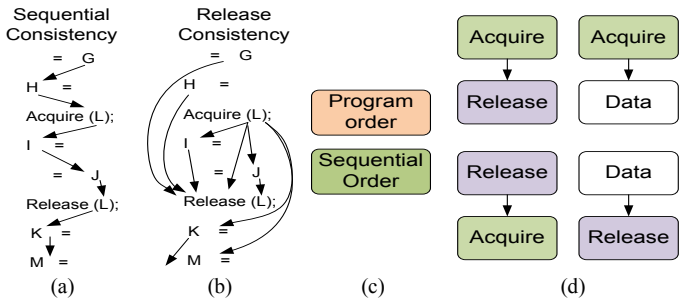

Figure 1. a) SC model b) RC model c) Global orders under SC model d) Global orders under RC model

B. RC Model

The RC model [6][9] is a refinement of the WC model [10]. It classifies the *synchronization* operations as *acquire* and *release* operations. The acquire operation delays the following data operations until the lock is obtained and does not wait for the completion of the previously issued data (read, write) operations. The release operation is to inform about the completion of previously issued data operations and does not delay the subsequent data operations. According to the RC model (Figure 1(b)), the independent data (read, write) operations on (G, H) are allowed to be reordered with each other, with the acquire operation on lock (L) and with the data operations on (I, J) in the critical section. They are not permitted to be reordered with respect to the release operation on lock (L). The data operations (I, J) 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 (L). The data operations on (K, M) are allowed to be reordered with each other, with the prior outstanding release operation on lock (L) and with the prior outstanding data operations on (I, J). However, they are not permitted to be reordered with respect to the prior acquire operation on lock (L). The global orders to be enforced on the shared memory operations under the RC model are given in Figure 1(d).

C. Platform Architecture

A homogenous McNoC platform is shown in Figure 2(a). As demonstrated in Figure 2(b), each Processor-Memory (PM) node consists of a processor, transaction controller (TCTRL), Synchronization Handler (SH), Network Interface (NI) and the local memory. The platform uses 2D mesh packet switched *Nostrum* NoC [10] with an adaptive routing algorithm. It is a buffer-less network and only buffers are used at the NIs to store the packets before injection into and after ejection from the network. The NI connects a PM node to the network. It performs packetization, de-packetization, queuing, arbitration and communication over the network. The platform uses the DSM in the network. All shared parts in the local memories constitute the DSM in a single global address space. The local memory is connected to the local processor within the node and to the remote processors via the network.

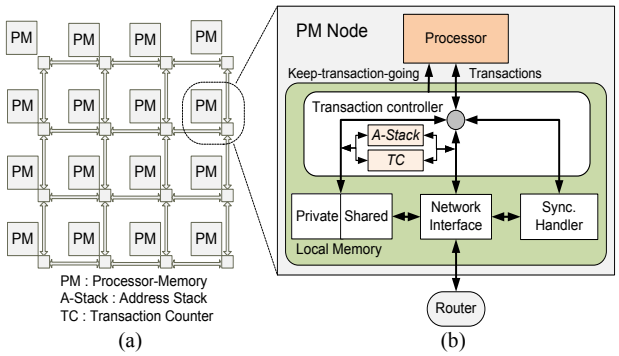

Figure 2. a) Homogeneous McNoC b) PM node

The platform also uses the *distributed* locks in the network. 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. The synchronization (acquire, release) requests to the SH either come from the local processor or from the remote processor via the network. If the requested

lock is available then it is acquired. Otherwise, a negative acknowledgement is sent back and the source node sends again the same request until the lock is gained. A release request makes the lock available for the next acquire on it. The data (read, write) operations to the local shared memory and synchronization (acquire, release) operations to the local SH are accomplished within the node. For the remote accesses, message passing is carried out to the remote node via the network. The customized interface (TCTRL) like any standard interface [15][17] integrates the processor with the rest of the system. It also implements the *memory consistency protocols* using the hardware structures (*TC, Address-Stack*). The TCTRL is developed specifically for the LEON3 IP-core [17] which is used in each node of the network.

D. Implementation of the SC Model

The SC model is implemented [8] by enforcing the required global orders (Figure 1(c)) on the shared memory operations.

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

Sequential Order: The multiple processors mutually agree on a common lock to sequentially access the critical resource.

E. Implementation of the RC Model

The RC model is implemented [9] by enforcing the required global orders (Figure 1(d)) on the shared memory operations.

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* (read, write) operations issued before the release operation. The *TC* is incremented by the issuance of a data operation. It is decremented by the completion of a data operation. The issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations, i.e., (*TC=0*).

Acquire → Data/Release: To enforce these global orders, the processor is *stalled* on the issuance of an acquire operation till the acquisition of the lock. The lock is gained by a processor before entering to the critical section and before trying to release it.

Release → Acquire: This global order is enforced by sequential ordering on a lock in the multiprocessor system. The lock is released by a processor before the next acquire on it.

Data operations to the same location: are constrained for the purpose of correctness. To that end, an *address stack (A-Stack)* is used in each node of the network to ensure the parallel program correctness [9].

IV. Experiments and Results

A. Experimental Setup

We experimented on a configurable cycle-accurate McNoC simulation platform constructed in VHDL (Figure 2). The LEON3 processor [17] is used in each node of the network. The size of the shared memory in each node is 16 MB. The SH maintains 256 locks in each node of the network. The *TC* is 32 bits and the *A-Stack* can stack up to 64 addresses each with 24 bits. The size of the *A-Stack* is kept small and it is utilized efficiently. The addresses are popped from the *A-Stack* continuously by the completion of operations in a pipelined manner. The packet formation in the NI uses 7 fields (96 bits). The buffering capacity at the NI is 64 packets. The *Nostrum* NoC [10] uses 2D mesh regular topology and deflection routing policy.

We have developed some in-house synthetic and application workloads to evaluate the performance of the McNoC systems. Synthetic workloads are small in size and used to test a particular aspect of the system, while application workloads give accurate and deeper evaluations of the system [19][20]. The developed applications are light weight, specific to NoC/embedded architectures and communication centric.

B. Performance Metrics for the Scalability Analysis

To study the scalability of the RC and SC models in the McNoC systems (Figure 2), application workloads are mapped on the different sized networks. The performance metrics like *execution time*, *performance*, *speedup*, *overhead* and *efficiency* are evaluated as the network scales up. The *execution time* (*ET*) of a workload is the time from the start of the execution on the first processor to the end of the execution on the last processor in the system. The *performance* is the reciprocal of the *ET*. The *speedup* (*SP*) is the ratio of the single core execution time (*Ts*) and the execution time of the multicore (*Tm*) system. We define the *communication overhead* (*OH*) of the multicore system as: *Nc*Tm-Ts*, where *Nc* is the number of cores in the system. The *efficiency* (*EF*) of the multicore system is defined as the ratio *SP*/*Nc*. For the synthetic workloads, the ET of the multicore normalized to the single core is defined as (*Nor-ET1C)*. In the experiments, the effects of the network size on the *ET,* performance, *SP, OH* and *EF* are investigated under the RC and SC models in the McNoC systems.

C. Synthetic Workloads

To demonstrate the benefits of the *distributed locks*, we evaluate the RC and SC models with synthetic workloads manually mapped on the LEON3 processors in the systems (Figure 3(a)). The same sequence of transactions is generated by the processor in each node of the network. The workloads have both data and synchronization operations. For the lock and protected (critical section) data operations, hotspot traffic pattern is generated (Figure 3(b)). We consider an 8x8 network. For *k* locks, the network is divided into *k* equal segments. All nodes within a segment synchronize over a common lock in a node that belongs to the same segment.

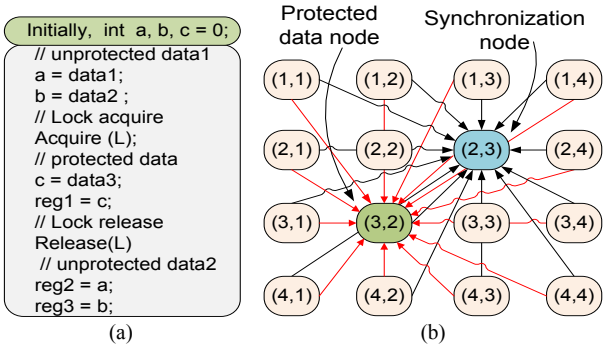

Figure 3. a) Sequences of transactions generated b) Traffic Patterns

The performance of the RC and SC models are compared using different number of segments/locks in the 8x8 network (Figure 4(a)). As the number of segments/locks increase in the network, the performance quickly increases due to the fact that different segments synchronize over different distributed locks in the network. The average lock acquire wait time is reduced as the network traffic/congestion decreases. The performance is higher under the RC model compared to the SC model due to reordering and relaxation in the shared memory operations. The average performance under the RC model for 1 to 32 locks is increased by 17.6% to 54.6% over the SC model. The ET of the 64-cores normalized to the single core (*Nor-ET1C*) is shown in Figure 4(b). The ideal *Nor-ET1C* is 1 by assuming zero communication overhead in the 8x8 system and the ET of the 64-cores system is equal to that in the single core system. Note that, the same sequence of transactions is generated by identical processors in the system. The deviation of the actual *Nor-ET1C* under both the memory models from the ideal case decreases as the number of the segments/locks increases in the network.

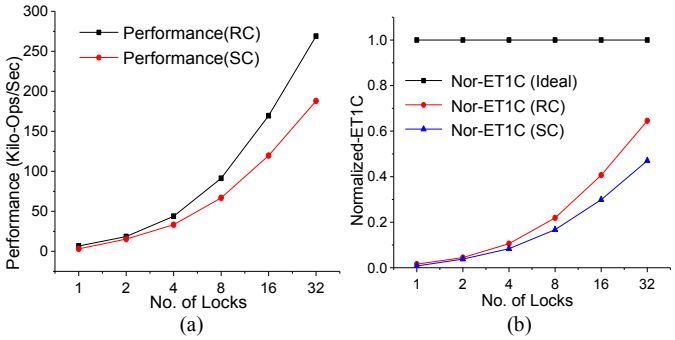

Figure 4. (a) Performance (b) Normalized ET of 64-cores to single core

D. *Application Workloads*

1) *Bit Count*

Bit count application analyzes a data vector and calculates the number of set bits in each integer data item. After initialization these data items are read, analyzed and the output values are stored in the DSM. Different sizes vectors are used with (16, 32, 64, 128, 256, 512 and 1024) data elements. The network size is increased from 1 to 64 nodes. When a data vector of 16 elements is mapped on the 4x8 and 8x8 networks, only 16 nodes are involved in the computations. Similarly, when a 32 data vector is mapped on the 8x8, only 32 nodes perform the computations. For the rest of the data vectors all nodes perform the computation in the 8x8 network. Each node operates on the data items in the *randomly* selected node and also writes the output results into the same node.

Figures 5~8 illustrate the ET, speedup, communication overhead and efficiency of the RC and SC models under different sizes of networks and different data sets. As illustrated in Figure 5, the Application workload ET (AET) is decreased as the system size is increased from 1 to 64 cores. This is due to the division of computation cost in the network. The RC model further decreases the AET compared to the SC model by reordering and overlapping the shared memory operations in the network.

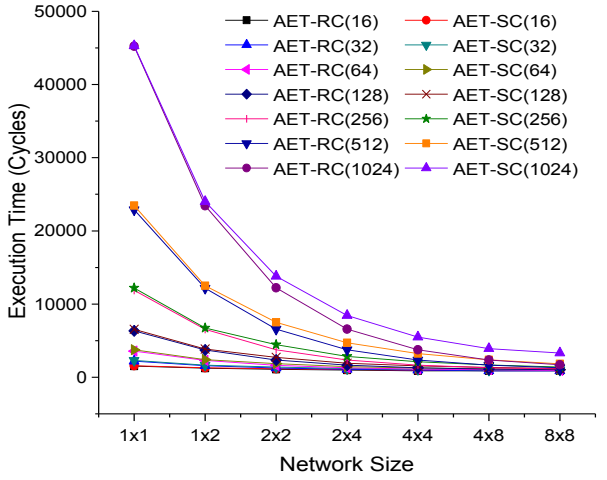

Figure 5. Execution Time under Bit Count Application

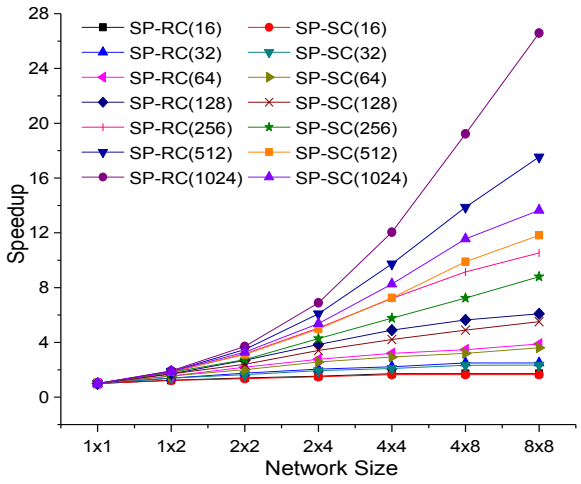

Figure 6. Speedup under Bit Count Application

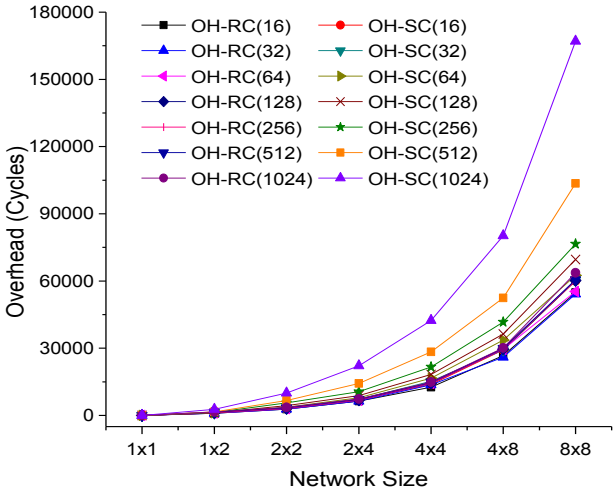

Figure 7. Communication Overhead under Bit Count Application

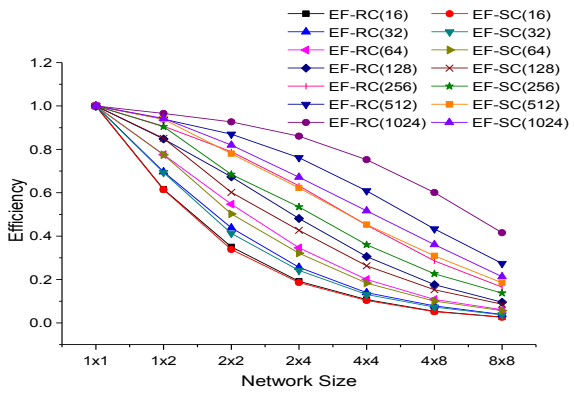

Figure 8. Efficiency under Bit Count Application

The AET reduction also depends on the data set size. It is high under the larger data set due to the parallelization of the significant amount of computation cost in the system. For the 1024 data set, the AETs in the single core system are 26.6 and 13.6 times of that in the 64-cores system under the RC and SC models, respectively. Likewise, the speedup (Figure 6) grows faster under the RC model compared to the SC model as the system size is increased. The speedup under the RC model is even higher under the larger data set. It is because of the efficient handling of the communication overhead (Figure 7) under the RC model by allowing *more* outstanding operations in the network which are overlapped and pipelined with each other. For the 1024 data set, the overhead in the 64-cores system compared to the two-core system is 39.6 and 60.9 times under the RC and SC models, respectively. The efficiency (Figure 8) is maintained high under the RC model compared to the SC model under different data sets when the network size grows up. In general, the RC model demonstrates better scalability and can efficiently utilizes the system resources in the larger networks. The execution time and overhead are lower and the speedup and efficiency are higher compared to the SC model.

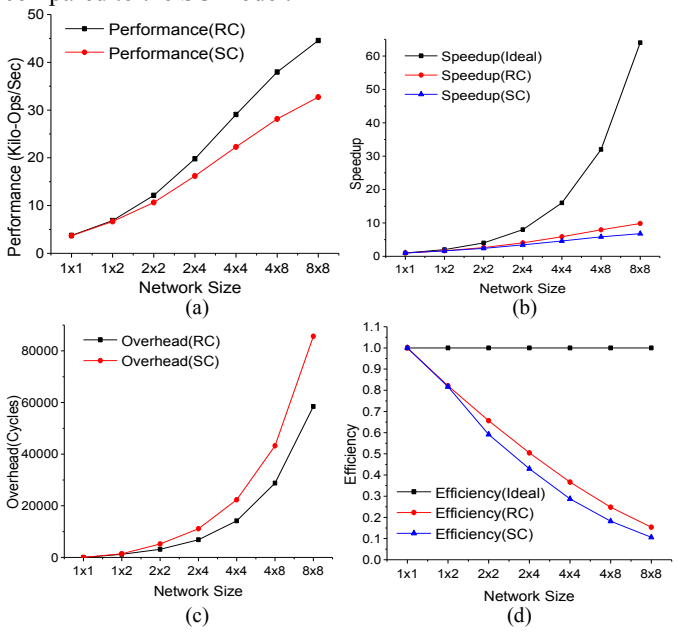

Figure 9. Bit Count: a) AET b) Speedup c) Overhead d) Efficiency

The *average* performance, speedup, overhead and efficiency under the RC and SC models are given in Figure 9. As shown in Figure 9(a), on average the performance in the 64-cores systems compared to the single core systems are 12 and 8.8 times higher under the RC and SC models, respectively.

2) Pattern Search

The application searches data patterns (P) against the data elements (D), which are initialized in the shared memory across the network. Four different cases are simulated using different combinations of the patterns and data elements. The system size is increased from 1 to 64-cores. *P32-D32:* when 32 patterns and 32 data elements are mapped on the 8x8 network, only 32 nodes participate in the computations. *P32-D64:* For 32 patterns and 64 data elements, one pattern each in the 32 nodes, while one data element each in the 64 nodes is initialized in the 8x8 network. Also, 32 nodes are involved in the computations. *P64-D32:* one pattern each in the 64 nodes, while one data element each in the 32 nodes is initialized in the 8x8 network. All 64 nodes perform the computation. *P64-D64:* one pattern and data element each is mapped in the 8x8 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.

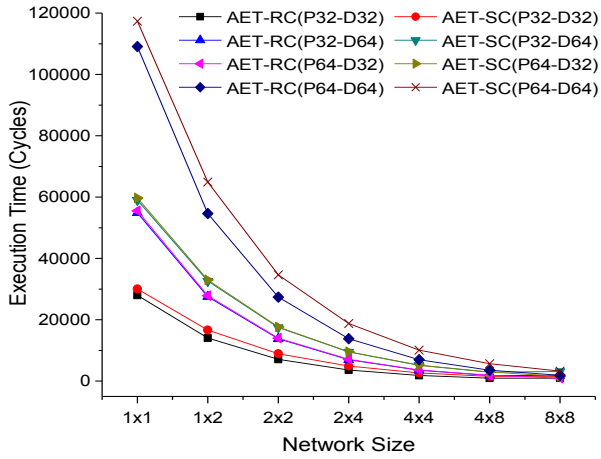

Figure 10. Execution Time under Pattern Search Application

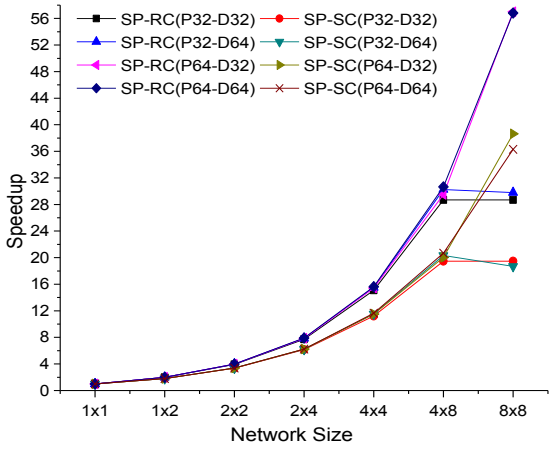

Figure 11. Speedup under Pattern Search Application

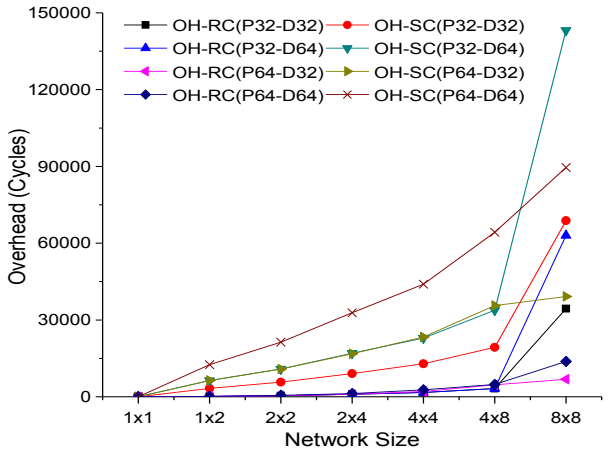

Figure 12. Communication Overhead under Pattern Search Application

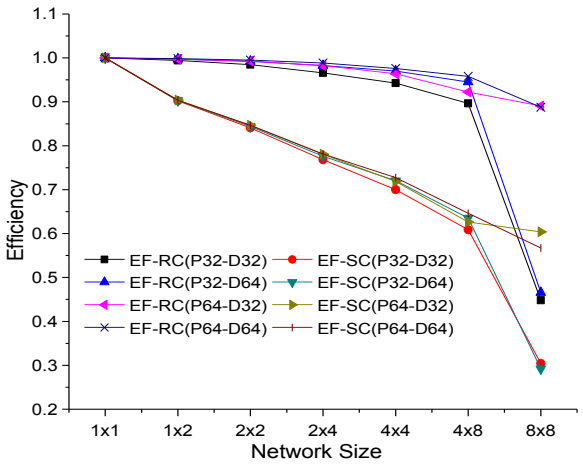

Figure 13. Efficiency under Pattern Search Application

For the pattern search application, Figures 10~13 demonstrate the AET, speedup, communication overhead and efficiency of the RC and SC models under different sizes of networks and different data sets. As the system scales up from 1 to 64-cores, the AET reduction is high under both the memory models for the (P64-Dx) cases, because these problem sizes fit well in to the increasing size of network compared to the (P32-Dx) cases (Figure 10). For instance, better scaling behavior can be observed under the (P64-D32) case, where the problem size fits well into the 8x8 network and each node is involved in the computation. Due to parallelization of more computation time among the nodes, the AETs are reduced more as the system size is scaled up. The AETs in the single core systems are 57 and 38.6 time of that in the 64-cores systems under the RC and SC models, respectively. The AET reduction under the RC model over the SC model is high by pipelining and overlapping the shared memory operations. Similarly, the speedup (Figure 11) grows quickly under the RC model compared to the SC model. After 32 nodes the speedup levels off up to 64 nodes under the (P32-Dx) cases, since the same amount of computation is performed in the 4x8 and 8x8 networks. The communication overhead (Figure 12) under both the memory models significantly increases under the (Px-D64) configurations when the network size is increased. It is due to the fact that for each pattern all 64 data elements are searched which are distributed across the network. Also, as the network size is increased, the efficiency (Figure 13) is maintained high under the RC model compared to the SC model. Overall, the RC model again maintains low execution time and high speedup compared to the SC model.

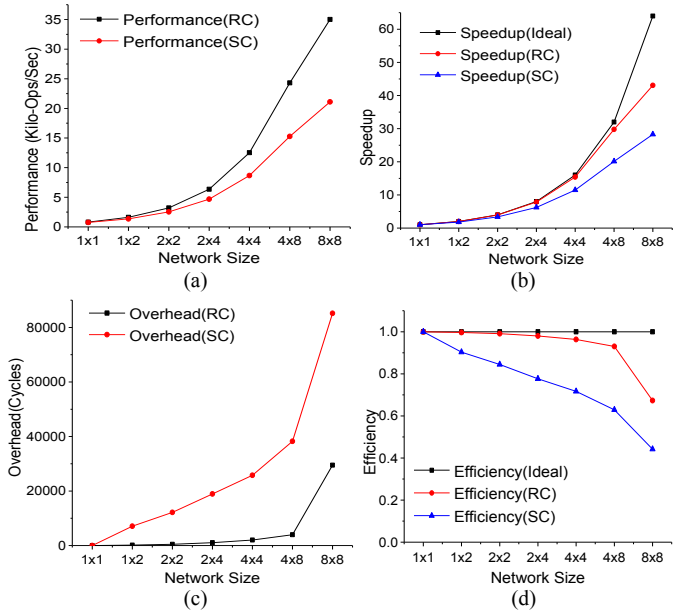

Figure 14. Pattern Search: a) AET b) Speedup c) Overhead d) Efficiency

The average performance, speedup, overhead and efficiency under the pattern search application are given in Figure 14. The increase in the performance and speedup for both the memory models in the 64-cores systems over the single core systems are higher in contrast to the bit count application. This is because of the *low* computation to communication ratio. The computation time per input data is less (9 cycles) under the pattern search application compared to that under the bit count application (21 cycles). In addition, the communication is significant under the pattern search application, because the numbers of input and output data items are more compared to the bit count application. The increase in the average performance (Figure 14(a)) under the RC and SC models is 31.4% and 19.2% higher than that in the bit count application (Figure 9(a)). The average speedup (Figure 14(b)) for the RC model is 43.1 (almost ideal), while for the SC model it is 28.2 in the 64-cores system. The RC model compared to the SC model shows even better and more scalable behavior by allowing *more* outstanding data operations on the network which are reordered and overlapped with each other. The average communication overhead (Figure 14(c)) is controlled efficiently under the RC model with the increasing size of the network. The overhead reduction under the RC model over the SC model is quite high compared to the (Figure 9(c)). As long as the system size increases, the average efficiency (Figure 14(d)) is maintained high (close to the ideal case 1) compared to the (Figure 9(d)). The average efficiency in the 64-cores system for the RC and SC models is 0.67 and 0.44, respectively.

E. *Summary of the Scalability Analysis of RC and SC models*

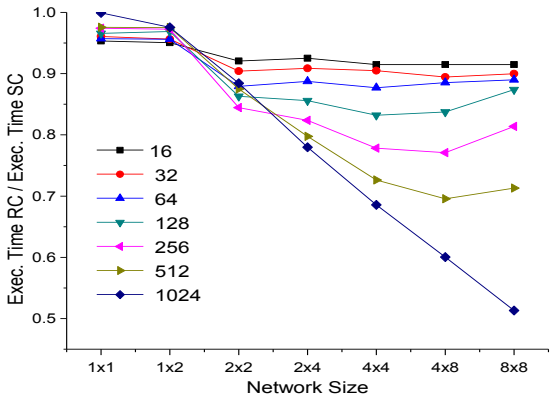

Figure 15. Bit Count: Ratio of AETs (RC/SC)

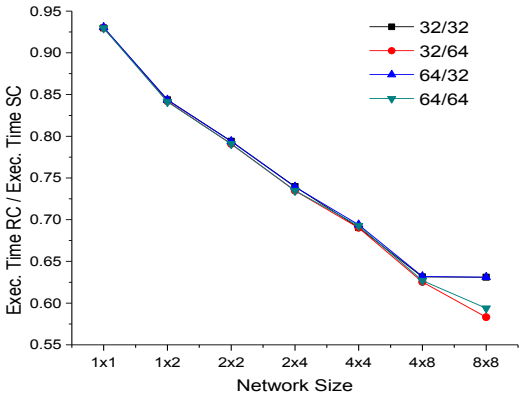

Figure 16. Pattern Search: Ratio of AETs (RC/SC)

In all our experiments, the execution time of RC model 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 to the platform. However, the observed trends suggest that the RC model scales inherently better with the network size than the SC model. As shown in Figure 15, the execution time is very similar for the small networks. As the network size grows, the execution time under the RC model decreases relative to the SC model and at some point (network size) the decrease flattens off. It depends sensibly on the application and its match to the architecture, when exactly this leveling off occurs. As long as the speedup increases, the benefits of the RC model over the SC model also increase, but when the nature of the problem makes it harder to utilize the additional parallelism, the benefits of the RC model over the SC model saturate as well. However, problems that scale well, like the 1024-bit count problem, continue to obtain increased benefits from the higher level of parallelism that the RC model offers compared to the SC model. We expect the trend shown in Figure 15 to continue for larger networks, which means that the performance benefits of the RC model continue to increase for well matched problems as long as the network size grows. Exactly, the same trend is visible in Figure 16. Thus, we conclude that the performance increase of the RC model over SC model can be significantly higher than 50% as observed in our experiments.

V. CONCLUSION

The scalability of the RC and SC models is analyzed in the NoC based DSM systems with 1 to 64 nodes based on the workloads mapping on the various sizes of networks with different data sets. The results show that under the synthetic workloads, the performance of the RC model is increased by 17.6% to 54.6% over the SC model using distributed locks in the 8x8 network. Under the application workloads, as long as the system size scales up, the execution time under the RC model decreases relative to the SC model. It depends on the scaling of the problem size and how efficiently the RC model is utilized compared to the SC model. The performance gain for the RC model over the SC model is expected to be higher than 50% as observed in the results, when the network size is further increased.

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

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

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

[8] A. Naeem et al., "Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multicore Systems," in: Proc. of the 16th (ASP-DAC), 2011.

[9] A. Naeem et al., "Architecture Support and Comparison of Three Memory Consistency Models in NoC based Systems," in: Proc. of Euromicro Conference on Digital Systems Design (DSD), 2012.

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

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

[12] 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 (DSD), pp. 53-60, 2006.

[13] J.W. van den Brand and M. Bekooij, "Streaming consistency: a model for efficient MPSoC design," in: Proc. of 10th Euromicro (DSD), 2007.

[14] 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.

[15] 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.

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

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

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

[19] C. Grecu, A. Ivanov, A. Jantsch, P.P. Pande, E. Salminen, U.Y. Ogras, R. Marculescu, Towards Open Network-on-Chip Benchmarks, First Int. Symposium on Networks-on-Chip (NOCS'07), May 2007.

[20] Zhonghai Lu, A. Jantsch, E. Salminen, and C. Grecu. Network-on-chip benchmarking specification part 2: Micro-benchmark specification. Technical Report Version 1.0, OCP-IP, May 2008.