
# Architecture Support and Scalability Analysis of Memory Consistency Models in Network-on-Chip based Systems

**Abdul Naeem**

Doctoral thesis in Electronic Systems
KTH- Royal Institute of Technology
Stockholm, Sweden 2013

**TRITA-ICT/ECS AVH 12:11**
**ISSN 1653-6363**
**ISRN KTH/ICT/ECS/AVH-12/11-SE**
**ISBN 978-91-7501-617-7**

**KTH, School of Information and Communication Technology**
**Department of Electronic Systems**
**SE-164 40 Stockholm, Sweden**

Academic dissertation for the Degree of Doctor of Philosophy in Communication Systems at Kungliga Tekniska Högskolan to be publicly defended on 13 March 2013 at 09:00 in Sal E, Forum, Isafjordsgatan 39, Kista.



Tryck: Universitetsservice US AB

# Abstract

The shared memory systems should support parallelization at the computation (multi-core), communication (Network-on-Chip, NoC) and memory architecture levels to exploit the potential performance benefits. These parallel systems supporting shared memory abstraction both in the general purpose and application specific domains are confronting the critical issue of memory consistency. The *memory consistency* issue arises due to the unconstrained memory operations which leads to the unexpected behavior of shared memory systems. The memory consistency *models* enforce ordering constraints on the memory operations for the expected behavior of the shared memory systems. The intuitive *Sequential Consistency (SC)* model enforces strict ordering constraints on the memory operations and does not take advantage of the system optimizations both in the hardware and software. Alternatively, the *relaxed* memory consistency models relax the ordering constraints on the memory operations and exploit these optimizations to enhance the system performance at the reasonable cost. The purpose of this thesis is twofold. First, the novel architecture supports are provided for the different memory consistency models like: SC, Total Store Ordering (TSO), Partial Store Ordering (PSO), Weak Consistency (WC), Release Consistency (RC) and Protected Release Consistency (PRC) in the NoC-based multi-core (McNoC) systems. The PRC model is proposed as an extension of the RC model which provides additional reordering and relaxation in the memory operations. Second, the scalability analysis of these memory consistency models is performed in the *McNoC* systems.

The *architecture supports* for these different memory consistency models are provided in the McNoC platforms. Each configurable McNoC platform uses a packet-switched 2-D mesh NoC with deflection routing policy, distributed shared memory (DSM), distributed locks and customized processor interface. The memory consistency models/protocols are implemented in the customized processor interfaces which are developed to integrate the processors with the rest of the system. The realization schemes for the memory consistency models are based on a *transaction counter* and an *address stack*-based novel approaches. The transaction counter is used in each node of the network to keep track of the outstanding memory operations issued by a processor in the system. The address stack is used in each node of the network to keep track of the addresses of the outstanding memory operations issued by a processor in the system. These hardware structures are used in the processor interface to enforce the required *global orders* under these different memory consistency models. The realization scheme of the PRC model in addition also uses *acquire counter* for further classification of the data operations as *unprotected* and *protected* operations.

The *scalability analysis* of these different memory consistency models is performed on the basis of different workloads which are developed and mapped on the various sized networks. The scalability study is conducted in the McNoC systems with 1 to 64-cores with various applications using different problem sizes and traffic patterns.

The performance metrics like *execution time, performance, speedup, overhead* and *efficiency* are evaluated as a function of the network size. The experiments are conducted both with the synthetic and application workloads. The experimental results under different application workloads show that the average execution time under the relaxed memory consistency models decreases relative to the SC model. The specific numbers are highly sensitive to the application and depend on how well it matches to the architectures. This study shows the performance improvement under the relaxed memory consistency models over the SC model that is dependent on the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio and the problem size. 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.

***Keywords:*** *Memory consistency, Protected release consistency, Distributed shared memory; Network-on-Chip, Scalability*

# Acknowledgements

The doctoral study is a long process full of many ups and downs. The foremost thing(s) required under such studies are patience, focus and target oriented approach.

First of all, I thank almighty *Allah*, the most gracious and beneficent, for giving me the strength and the capacity to accomplish my doctoral thesis.

Throughout the course of time several peoples helped me in various ways. First, I would like to thank my main supervisor Prof. *Axel Jantsch* the most, whose scientific advices and guidance at every step during the course of time leads to my technical development and better understanding of the topic. Then, I am thankful to my Co. supervisor Assoc. Prof. *Zhonghai Lu* for the guidance, help and cooperation. I also thank Prof. Ahmed Hemani for some small talks, useful discussions and for reviewing my thesis. I am thankful to the anonymous reviewers of my research work whose feedback helped me in the orientation toward the subject. I am indebted to all other faculty members in the department of Electronic Systems from whom I learned while taking their courses. I thank the administrative staff (Alina Munteanu) at our department for the help.

I present my appreciation to all the colleagues in the department for their help, cooperation and valuable discussions. I thank Xiaowen Chen for the help and discussions.

I express my gratitude to all my friends from Pakistan whose company supported me during the completion of my thesis. The wonderful time I spend with them while playing cricket in the summer are the great occasions in my life to remember.

I would like to thank the Higher Education Commission (HEC), Pakistan for giving me the opportunity and financial support for my PhD study. I also acknowledge KTH-Royal Institute of Technology for providing the financial support to present my research work/results at various international forums.

Thanks to Prof. Jari Nurmi for being my opponent. I would like to thank Prof. Eric Martin Törngren, Prof. Krzysztof Kuchcinski, and Assoc. Prof. Juha Petteri Plosila for being members of the grading committee.

Finally, I give my deepest gratitude to my family (Wife: Madiha Zeb, Sons: Abdullah Khan, Ebadullah Khan and Daughter: lovely little Ajwa Naeem). I also thank my wife for proof reading my research papers. I express many thanks to my parents from the core of heart (Abdul Jalil, Bibi Habibun) for their everlasting love, praying for my success and health and exceptional support despite of the old age and illness. I thank my brothers and sisters in Pakistan for their love and endless concerns. My elder sister Bibi Naheed was died in August 2006 in the heart surgery. I loved her the most and dedicate my PhD thesis to my *late* sister and to my parents.

During the study period, I lost some of my close relatives, but they are still in my memory. I feel regret for their families.

Abdul Naeem
August 2012, Stockholm

# Table of Contents

# List of Publications

- **Abdul Naeem**, Xiaowen Chen, Zhonghai Lu, and Axel Jantsch, "Scalability of Weak Consistency in NoC based Multicore Architectures," In *Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2010)*, pages 3497 – 3500, Paris, France, June 2010.

- **Abdul Naeem**, Xiaowen Chen, Zhonghai Lu, and Axel Jantsch, "Scalability of Relaxed Consistency Models in NoC based Multicore Architectures," *ACM SIGARCH Computer Architecture News*, vol. 37, no.5, pages 8 – 15, April 2010.

- **Abdul Naeem**, Xiaowen Chen, Zhonghai Lu, and Axel Jantsch, "Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems," In *Proceedings of 16th ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC 2011)*, pages 154 – 159, Yokohama, Japan, January 2011.

- Axel Jantsch, Xiaowen Chen, **Abdul Naeem**, Yuang Zhang, Sandro Penolazzi, and Zhonghai Lu, "Memory Architecture and Management in an NoC Platform," In Axel Jantsch and Dimitrios Soudris, editors, *Scalable Multi-core Architectures: Design Methodologies and Tools*, Eds. Berlin, Germany: Springer, 2011, pages 3 – 28.

- **Abdul Naeem**, Axel Jantsch, Xiaowen Chen, and Zhonghai Lu, "Realization and Scalability of Release and Protected Release Consistency Models in NoC based Systems," In *Proceedings of 14th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2011)*, pages 47 – 54, Oulu, Finland, August-September 2011.

- **Abdul Naeem**, Axel Jantsch, and Zhonghai Lu, "Architecture Support and Comparison of Three Memory Consistency Models in NoC based Systems," In *Proceedings of 15th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2012)*, pages 304 – 311, Cesme, Izmir, Turkey, September 2012.

- **Abdul Naeem**, Axel Jantsch, and Zhonghai Lu, "Scalability Analysis of Release and Sequential Consistency Models in NoC based Multicore Systems," In *Proceedings of IEEE International Symposium on System-on-Chip (SOC 2012)*, pages 1 – 7, Tampere, Finland, October 2012.

- **Abdul Naeem**, Axel Jantsch, and Zhonghai Lu, "Scalability Analysis of Memory Consistency Models in NoC-based Distributed Shared Memory SoCs," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 2013, accepted for publication. doi: 10.1109/TCAD.2012.2235914

# Other Publications:

- **Abdul Naeem**, "Shared memory consistency models evaluation in NoC based multicore systems," In *Proceedings of PhD Forum, Design Automation and Test in Europe (DATE 2012)*, Dresden, Germany, March 2012.

- **Abdul Naeem**, Axel Jantsch, and Zhonghai Lu, "Scalability and Performance Evaluation of Memory Consistency Models in NoC based Multicore SoCs," *Poster in ICES 5$^{th}$ Annual Conference: World-wide Trends and Challenges in Embedded Systems (ICES 2012)*, Stockholm, Sweden, August 2012.

- Zhonghai Lu, Xiaowen Chen, Yuang Zhang, **Abdul Naeem**, Axel Jantsch, Iraklis Anagnostopoulos, Sotirios Xydis, Alexandros Bartzas, Dimitrios Soudris, Christos Baloukas, *FP7 EU MOSART project Deliverables* D1.2 – "Final version of the Specification of platform interfaces and services," Deliverables D1.3/ D1.6 – "First/Final versions of Nostrum Extensions for Distributed Memory and Abstract Data Types," EU IST project contract number: 215244. Available from www.mosart-project.org/.

# List of Figures

# List of Tables

# Abbreviations and Acronyms

| | |
|---|---|
| SC | Sequential Consistency |
| TSO | Total Store Ordering |
| PSO | Partial Store Ordering |
| WC | Weak Consistency |
| RC | Release Consistency |
| PRC | Protected Release Consistency |
| PC | Processor Consistency |
| RMO | Relaxed Memory Ordering |
| TC | Transaction Counter |
| WTC | Write Transaction Counter |
| A-Stack | Address Stack |
| WA-Stack | Write Address Stack |
| TCTRL | Transaction Controller |
| NoC | Network-on-Chip |
| SoC | System-on-Chip |
| McNoC | Multi-core NoC |
| MPSoCs | Multi-Processor Systems-on-Chip |
| CMPs | Chip Multi-Processors |
| SMMPs | Shared Memory Multi-processors |
| IP | Intellectual Property |
| DSM | Distributed Shared Memory |
| Sync | Synchronization |
| PS | Protected Section |
| SH | Synchronization Handler |
| NI | Network Interface |
| PM | Processor Memory |
| DME | Data Management Engine |
| CICU | Core Interface Control Unit |
| NICU | Network Interface Control Unit |
| V2P | Virtual To Physical |
| AXI | Advanced eXtensible Interface |
| OCP | Open Core Protocol |
| VCI | Virtual Component Interface |
| DTL | Device Transaction Level |
| $TC_{PD}$ | Transaction Counter for the protected data |
| AC | Acquire Counter |
| WL | Workload |
| SWL | Synthetic Workload |
| 2-D | Two Dimensional |
| 3-D | Three Dimensional |
| ET | Execution Time |

| | |
|---|---|
| SET | Synthetic workload Execution Time |
| AET | Application workload Execution Time |
| WFC | Wave Front Computation |
| Reg | Register |
| AMBA | Advanced Microcontroller Bus Architecture |
| AHB | Advanced High-performance Bus |
| DRAM | Dynamic Random Access Memory |
| TSV | Through Silicon Via |
| SP | Speedup |
| $T_S$ | Execution Time of Single Core System |
| $T_M$ | Execution Time of Multi-core System |
| OH | Communication Overhead |
| $N_C$ | Number of cores |
| EF | Efficiency |
| HTM | Hardware Transactional Memory |
| STM | Software Transactional Memory |
| HyTM | Hybrid Transactional Memory |
| JVM | Java Virtual Machine |
| MSI | Modified-Shared-Invalid |
| MESI | Modified-Exclusive-Shared-Invalid |
| MBs | Memory Barriers |
| WMBs | Write Memory Barriers |
| STBAR | Store Barrier |
| MEMBARs | Memory Barriers |
| SFENCE | Store Fence |
| LFENCE | Load Fence |
| MFENCE | Memory Fence |
| PAMC | Physical Address Memory Consistency |
| VAMC | Virtual Address Memory Consistency |
| DVMC | Dynamic Verification of Memory Consistency |
| SCNF | Sequential Consistency Normal Form |
| DRF | Data Race Free |
| FIFO | First-In-First-Out |
| CAM | Content Addressable Memory |

# Chapter 1

# Introduction

This chapter briefly states the general trends and developments in the shared memory parallel systems. The problem of memory consistency is introduced. The difference between the cache coherence and memory consistency is discussed. We describe the scope of the thesis and the research methodology which has been adopted. The research work conducted under this thesis is overviewed and the main contributions are summarized. The final part of the chapter focuses on the outline of thesis.

## 1.1 Trends and Developments

There are several limitations in shrinking the size of logic gates and microelectronics systems. Therefore, the performance of a processor-based system is enhanced by using different ways in the past several years. The system performance is enhanced by increasing the parallelism at the instruction level (e.g., Instruction Level Parallelism, ILP). Another method which has been adapted to increase the performance of the system is to parallelize the computation at the thread level (e.g., Thread Level Parallelism, TLP). The high demand for the TLP has led towards the multi-cores, MPSoCs (Multi Processor Systems-on-Chip) or CMPs (Chip Multi Processors) systems. The processor development has been shifted from a single sequential processor to the parallel *multi-core* systems. Most of the computer companies, for instance, AMD, Intel, Sun, ARM, and IBM have shifted their next generation designs to be based on the multi-core systems [2][3][4][57]-[60]. The multiple cores are integrated on a single die (e.g., Intel: i7, Xeon E7-2820. AMD: Phenom II X6, FX-8150, etc.). The integrated cores in a system may be either of one type (homogeneous systems) or of different types (heterogeneous systems).

Another trend of parallelism in the communication (e.g., interconnection networks) has been driven by the poor scaling properties of the classical global wires and buses. This trend has guided towards the *Network-on-Chip* (NoC) paradigm. The NoC is an emerging solution for the parallel communication. It can be used as a reliable and scalable communication medium among the multiple cores in the system [5][6][7][8][61][109]. The on-chip communication could be parallelized over the network up to a great extent and high bandwidth could be offered compared to the traditional global buses. The IP-cores and distributed memories can be integrated in a NoC-based system using a specific topology and routing algorithm. However, in the NoC-based systems which support the shared memory abstraction, the memory access latency

depends on the *physical distance* among the masters and slaves where they are located on the chip. The average memory access latency should be kept optimal and minimal in the NoC-based systems. Thus, to minimize the average memory access latency, the shared memory organization can play a vital role in the NoC-based systems.

The *Distributed Shared Memory* (DSM) organization connects the physically distributed memories in a single global address space. The DSM in a single address space constitutes the virtual memory. It has non-uniform memory access time compared to the centralized shared memory organization. The DSM is preferred over a single centralized memory organization as it can significantly improve the system performance. However, it depends on the application types, problem sizes, applications mapping and the data locations in the memory. For a set of applications, which confines most memory accesses to the local region in the network, the DSM systems could be exploited more compared to the centralized shared memory systems. Within the DSM systems, the processor operates on a single global address space and scales better when the number of processors is increased in the system. The DSM systems accommodate larger problems compared to the centralized shared memory systems, and scale better as the number of processors grows in the system. As shown in Figure 1.1, the multi-core NoC (McNoC)-based system supporting DSM can combine the benefits of parallel computation, parallel communication, and at the memory architecture levels. The PM (Processor Memory) nodes can be integrated in a scalable manner in the McNoC system.

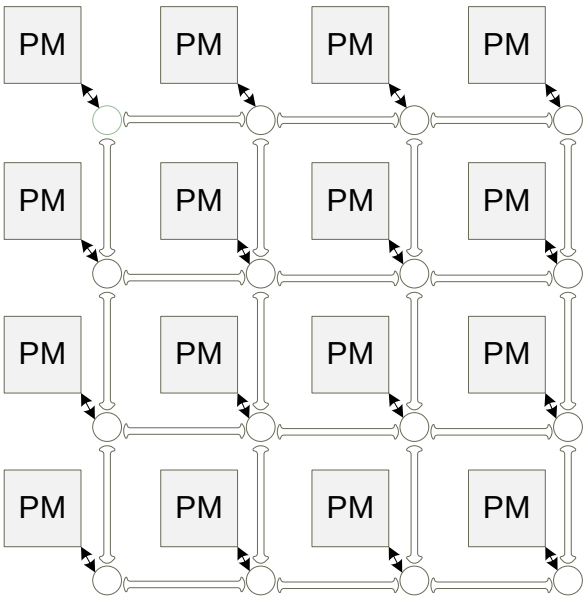



**Figure 1.1.** DSM-based McNoC system.

## 1.2 Problem Statement

In the uni-processor systems, a read operation issued by a processor to a memory location returns the most recent and correct value of the memory location. Since the last

value written to a memory location in the uni-processor systems is clearly defined. It is because that only one processor operates on the same memory location. On the other hand, the last value written to a shared memory location in the multi-processors systems is not clearly defined compared to the uni-processor systems, since more than one processor can access the same memory location at the same time. The operations issued by different processors to the same memory location are *conflicting* or *competing* with each other when at least one of the processor is involved in a write operation. The conflicting accesses cause the *data races* [62][63] among the multiple processors in the system. The data race (or race condition) occurs where the fetched value of a read operation is dependent on the timing and sequence of the operations which are issued by another processor to the same memory location in the system. A processor may observe the old or inconsistent value of the shared memory location due to these data races, which could lead to the incorrect behavior of the shared memory systems. In order to make the parallel program free of data races and to serialize the conflicting write operations to the critical/shared memory locations, the system must have support for the synchronization among the multiple processors. The *synchronization* support must be provided both at the hardware and software levels in the shared memory multi-processor systems. However, the provision of synchronization support still does not guarantee the *consistent* behavior of the shared memory systems. For example, an acquire synchronization operation on a lock must be completed before entering to the critical section or the execution of operations in the critical section must be completed before the issuance of a release operation on a lock.

The shared memory operations issued by a processor can be reordered due to several reasons both in the hardware (e.g., write buffer, cache, interconnection network) and in the software (e.g., compiler reordering, register allocation). Due to the reordering of memory operations by these system optimizations these operations which are issued by a processor may execute out-of-order in the system. The reordering of memory operations sometimes may lead to the unexpected behavior of the shared memory system for the programmer [10]. Even if a system has a proper support for the synchronization among the cores both at the hardware and software levels, reordering of operations issued by a processor under some situations may still lead to the unexpected behavior of shared memory multi-processor systems. The processor could observe the old or inconsistent data from the memory system and the entire multi-processor system may fail. As discussed earlier, it is illegal to enter the critical section without a lock, and also a lock must not be released until the execution of operations in the critical section is completed.

The shared memory multi-processor systems have the critical problem of *memory consistency*. The memory consistency problem arises due to the unconstrained memory operations which sometimes produce an illegal execution trace or unexpected behavior of the shared memory system. The memory consistency model specifies the execution order of the memory operations for the expected behavior of the shared memory systems. It is related to the enforcement of the ordering constraints on the shared memory operations. Memory consistency enforces restrictions on the order of shared memory operations to ensure the parallel program correctness. Intuitively, a read must return the most recent or correct value of a shared memory location in the context of multi-processor systems.

The memory consistency *models* are used to resolve the issue of the memory consistency. A memory consistency model specifies the ordering rules for the memory

operations. These rules must be followed by a system for the consistent behavior of the shared memory. There are several memory consistency *models* which are available in the literature [10]. These memory models correspond to the enforcement of different ordering constraints on the shared memory operations. The Sequential Consistency (SC) model [9] is an intuitive consistency model which requires the operations of an individual processor to be accomplished in the order specified by the program (*program order*). The *sequential order* is maintained among the multiple processors on the shared memory locations in the system. However, due to the strict nature, the SC model could not utilize the hardware and software optimizations in the systems. Therefore, several *relaxed* memory consistency models [10][11][12][13][14][51] have been emerged. These memory models enforce less ordering restrictions on the memory operations and also ensure the parallel program correctness. The *relaxation* is defined as the reordering, overlapping or pipelining among the independent shared memory operations which are issued by a processor in the system. It does not lead to the incorrect execution of the parallel program, i.e., the parallel program correctness is guaranteed. The parallel program correctness means that a read operation issued by a processor must return the correct or most recent value of a memory location in the multi-processor system. The relaxation or reordering among the shared memory operations could be allowed statically by the compiler or dynamically by the system hardware. The relaxed memory consistency models (e.g., Total Store Ordering, Partial Store Ordering, Weak Consistency, Release Consistency, etc.) exploit these system optimizations and the system performance is enhanced at the reasonable cost. Some of the commercial architectures support the relaxed memory consistency models such as: Digital Alpha [54], SPARC V9 Relaxed Memory Order (RMO) [73], and IBM PowerPC [52][53].

The relaxed memory consistency models require the *classification* of operations which are issued by a processor in the multi-processor systems. The memory operations are classified under the relaxed consistency models to specify the ordering requirements for different kind of memory operations in the system. This classification of operations helps to enable additional reordering and relaxation among the memory operations under these relaxed memory consistency models. The categorization of memory operations could be accomplished both at the software and hardware levels. The compilers and run time systems at the software level, while processor *interfaces* at the hardware level, can be used to distinguish between different types of memory operations. At the hardware level, sophisticated interfaces can be developed to integrate the processor with the rest of the system. There are some standard and well-defined interfaces available to integrate the IP-cores with each other in the system. For instance, Advanced Microcontroller Bus Architecture (AMBA), Advanced High-performance Bus (AHB) and Advance eXtensible Interface (AXI) from ARM limited [44], Open Core Protocol (OCP) from OCP-IP [45], Virtual Component Interface (VCI) [64], CoreConnect [65] and Device Transaction Level (DTL) from Philips Semiconductors [66] are the commonly available interfaces. These interfaces enforce the *ordering models* by assigning the IDs to the memory transactions issued by the same processor in the system. Based on these IDs, the memory transactions are either allowed or not allowed to be reordered with respect to each other. Apart from these standard interfaces, *customized* interfaces can also be developed depending upon the needs, requirements and setups. But, a customize processor interface should be equipped with the key features and functionalities which may be required under

any standard interface such as AXI [44] and OCP [45]. All these interfaces either standardized or customized should handle the transactions from a processor and must deal with the issues like: classification of memory operations, address translation, memory mapping, flow control of transactions from a processor and communication of transactions between a processor and memory system. More importantly, these interfaces can play a vital role to enforce the ordering constraints on the memory operations for the implementation of various memory *consistency protocols*.

The *objective* of this thesis is to explore the memory consistency issue specifically in the NoC-based multi-core (McNoC) systems. We provide a description of the architecture support and scalability analysis of the SC model and some relaxed memory consistency models, e.g., Total Store Ordering (TSO), Partial Store Ordering (PSO), Weak Consistency (WC), Release Consistency (RC) and Protected Release Consistency (PRC) in the McNoC systems. We illustrate the novel realization schemes of different memory consistency models in the McNoC systems. These memory consistency models are implemented by using a *transaction counter* and an *address stack*-based novel approaches. The synthetic and application workloads are developed and manually mapped on the processors to analyze the scalability of these memory consistency models in the McNoC systems. More details can be found in the later parts (Chapters 3 & 4) of the thesis which are compiled on our research works [1][17]-[25].

The cache coherence and memory consistency are two different problems of similar nature in the shared memory multi-processor systems. The cache coherence problem is observed in the multi-processor systems which use the data caches. While the memory consistency issue could be observed in the multi-processor systems that may or may not use the data caches. The cache coherence problem arises due to the different cached copies of the same shared data in the multi-processors system. The caches may follow the write-through or the write-back policy. The write-through policy always keeps the main memory updated, but it consumes more memory bandwidth compared to the write-back policy. The data caches either using the write-through or the write-back policy have the problem of coherence. The cache coherence protocols are used to resolve the issue of cache coherence. Without a cache coherence protocol, a processor may observe a *stale* data of a memory location. The cache coherence protocol maintains the coherence among all the data caches in the system. The sno*oping*-based cache coherence protocol [15] relies on the broadcasting and bus snooping by the cache controllers. It is suitable for the small scale bus-based multi-core systems. Since broadcasting is very expensive in the network-based multi-core systems. Hence, the snooping-based cache coherence protocol is almost impractical in the network-based multi-core systems. On the other hand, the *directory*-based cache coherence protocol [16] is used in the network-based multi-core systems. The directory-based protocol maintains the status information about the cache blocks and uses point-to-point communication to *invalidate* or *update* the cached copies of the requested block. Various cache coherence protocols like: MSI (Modified-Shared-Invalid) and MOSI (Modified-Owner-Shared-Invalid), etc. are used in the multi-processor systems. However, the directory-based coherence protocols also have some confronting issues like: extra coherence traffic, directory overhead, additional latencies and complexities. Memory consistency issue in contrast to the cache coherence is related to the enforcement of ordering constraints on the shared memory operations in order to ensure the parallel program correctness (e.g., the correct behavior of the shared memory

systems is guaranteed). As discussed earlier, our research works [1][17]-[25] mainly focus on the memory consistency issue by implementing different memory consistency models which are independent of the cache coherence protocols in the McNoC systems. This study targets the hard real time applications, applications specific systems, customized systems and some other application domains where the caches may not be employed. We have decoupled the consistency and coherence issues for the better and independent optimization. The cache coherence protocols could still be accommodated along with these independent memory consistency models in the McNoC systems. As a future work, the cache coherence schemes can be implemented on top of our memory consistency protocols in the McNoC systems.

## 1.3 Research Overview

The thesis mainly concentrates on the memory consistency issue and various memory consistency models are realized which are independent of the cache coherence protocols. The memory consistency issue is addressed in the customized application specific McNoC systems. Our study focuses on the implementation and scalability analysis of different memory consistency models. This thesis targets the following application domains:

- *Decoupling memory consistency and cache coherence protocols:* The memory consistency and cache coherence are two critical issues in the shared memory systems. Both these problems are of similar nature and target to achieve the consistent view of the memory system, but at different levels of abstraction. The memory consistency problem arises due to the unconstrained memory operations which sometimes leads to the unexpected behavior of the shared memory systems. Different memory consistency models are proposed to resolve the issue of memory consistency. In contrast, the cache coherence problem is due to the different cached copies of the same shared data in the shared memory systems. Various coherence protocols are used to handle the problem of cache coherence. The scope of the thesis is to target the hard real time applications, application specific systems and some other applications where data caching is not used. The main focus of our study is to distinguish the memory consistency and cache coherence from each other in the shared memory systems. Decoupling these two critical issues would help to optimize them separately. Also, for some systems which have different requirements on the size of the cache block and the consistency object, independent implementation schemes of the memory consistency and cache coherence problems are commonly accepted and preferred [1] [17]-[25].

- *Memory consistency models covered under this thesis:* The thesis focuses on the architectural support and scalability analysis of some well known memory consistency models, e.g., SC, TSO, PSO, WC and RC. We have also introduced a new memory consistency model called *Protected Release Consistency* (PRC) model as an extension of the RC model which provides additional reordering

and relaxation in the memory operations. There are several other memory consistency models available in the research. For example, the IBM-370 model [71], Processor Consistency (PC) model [69][72], Digital Alpha [54], SPARC V9 Relaxed Memory Order (RMO) [73], IBM PowerPC [52][53], *Eager RC* model [28], *Lazy RC* model [29], *Entry RC* [30], *Scope consistency* [31] and some other memory consistency models are presented in the literature. We have concentrated on the implementation and scalability analysis of the SC, TSO, PSO, WC, RC, and PRC models in due course of time. Our study does not consider the implementation and analysis of other memory consistency models, but we provide a brief overview of these memory models.

- *Customized NoC-based systems:* The heterogeneous and customized NoC-based shared memory systems have different requirements and design constraints compared to the general multi-processors shared memory systems. The customized NoC-based systems have less power consumption, require less but heterogeneous storage, make less or no use of caches and have often soft or hard real-time constraints. In contrast, the general multi-processor systems have more power consumptions, require more memory, make use of data caches and have no hard real-time constraints. The main focus of the thesis is on the memory consistency issue in the NoC-based customized shared memory systems. Different memory consistency models are realized [1][17]-[25] in the McNoC systems and their scalability study is performed.

We present a brief sketch of the methods, procedures and approaches which are followed to accomplish this research work. First, the literature review is conducted on the memory consistency issue. The NoC-based multi-core platforms supporting different memory consistency models are developed. The synthetic and application workloads are developed and mapped on the increasing size of the network to analyze the scalability of various memory consistency models. The research documents are compiled based on the experimental results and analysis. In the following, we briefly discuss the approaches and procedures which have been adopted to conduct this research work:

- *Literature review and problem understanding*: The literature review is carried out to explore the on-going research and challenges in the area of memory consistency. The existing research on the memory consistency models is mainly performed in the general multi-processors domain while quite less research is conducted [41][42][43] in the NoC-based multi-processors domain. The available research is fairly helpful to understand the memory consistency problem. But, for an average technical person to understand the core issue of the memory consistency is still difficult due to the critical nature of the topic. Along with the literature study, thinking in the right and specific direction, regular meeting and discussion with the supervisors and the comments of anonymous reviewers on the research work also assisted me in understanding and orientation towards the topic.
- *Platforms developments:* The NoC-based multi-core *platforms* were developed for various memory consistency models. All the nodes in a platform are

integrated with each other by a packet-switched on-chip network [46] in the 2-D mesh regular topology. Each node consists of a processor, customized interface to the processor, synchronization handler, network interface and the local memory. Each platform uses distributed shared memory organization and provides a support for the memory synchronization among the multiple cores both at the software and hardware levels. The customized interface is developed for the processor which integrates it with the rest of the system. Different memory consistency models (protocols) are implemented in the processor interface.

- *Benchmarks developments:* In order to assess the performance of various memory consistency models which are implemented in the McNoC systems, we have developed some benchmarks due to the non-availability of standard NoC benchmarks. The experiments are conducted by using these benchmarks. We have developed both the synthetic and application workloads to evaluate the performance of different memory consistency models in the McNoC systems. The small sized synthetic workloads are used to test a particular aspect of the system. On the other hand, the application workloads give accurate and deeper evaluations of the systems [48][49]. The developed benchmarks are parameterized so as to map and run them on the different number of cores with different memory layout and problem sizes. The performance of different memory consistency models is evaluated and compared under these developed applications.
- *Experiments and scalability analysis:* The experiments are planned and conducted on the various platforms supporting different memory consistency models. The experiments are performed by mapping the application workloads on different sized networks. The scalability study of various memory consistency models is conducted and the performance metrics like: *execution time*, *performance*, *speedup*, *overhead* and *efficiency* are evaluated when the network size is scaled up (Chapter 4).
- *Documentations:* Based on the analysis of the experimental results and understanding of the subject, the research documents like: project deliverables, technical report and research papers are complied. These research documents have reported our research results at various international forums.

The main concentration of our research work is on the memory consistency issue in the customized McNoC systems. We have identified and formulated problems related to the memory consistency issue in the McNoC systems. The memory consistency models are implemented in the McNoC systems and their performance is analyzed. In the following, we provide a brief overview of the main research work which is accomplished under this thesis:

- *Architectural support for memory consistency models:* We have proposed the architectural support for the memory consistency models independent of the cache coherence protocols in the McNoC systems. Different memory consistency models are realized in the McNoC systems using *novel* approaches. The

realization schemes of these memory consistency models use the *Transaction Counter* and *Address Stack*-based mechanisms. These hardware structures are used to enforce the required ordering constraints (global orders) under these memory consistency models. The SC model is realized by *stalling* the processor on the issuance of an operation in the program till its completion. The processor issues the next operation on the completion of a previously issued operation in the program. This enforces the *program order* under the SC model. The *sequential order* is enforced under the SC model by interleaving operations among the multiple processors in the system. The TSO and PSO models are realized by using the *Write Transaction Counter* (*WTC)* and *Write Address Stack* (*WA-Stack)*-based approaches. The *WTC* keeps track of the outstanding write operations issued by a processor in the system. The *WA-Stack* is used to keep track of the addresses of the outstanding write operations issued by a processor in the system. These hardware structures are used at the processor interface in each node of the network to enforce the required global orders (e.g., ordering constraints on the memory operations) under these memory consistency models. The WC model is realized by using a *Transaction Counter* (*TC*) and an *Address Stack* (*A-Stack*) at the processor interface in each node of the network. The *TC* keeps track of the outstanding data (read, write) operations issued by a processor in the system. The *A-Stack* keeps track of the addresses of outstanding data operations which are issued by a processor in the system. The required global orders under the WC model are enforced on the memory operations by using these hardware structures. The RC model is realized by using two different approaches. First, it is realized by using two *TCs* in each node of the network. Later, the RC model is realized by using a single *TC* in each node of the network. The realization scheme of the RC model is further enhanced by using the *TC* and *A-Stack* in each node of the network to enforce the required global orders on the memory operations. The *A-Stack* is used in each node of the network to ensure the parallel program correctness by constraining the operations issued by a processor to the *same* memory location to accomplish as per program order. It is due to the fact that the operations issued by a processor to the same location in the memory must be constrained for the purpose of correctness, i.e., a read operation must always fetch the most recently written or correct value of a memory location. The PRC model is proposed as an extension of the RC model. The realization scheme of the PRC model also uses an *Acquire Counter* (*AC*) in addition to the *TC* and *A-Stack* in each node of the network. The *AC* is used for further categorization of the data operations as *unprotected* and *protected* data operations under the PRC model.

- *Scalability analysis of memory consistency models:* Based on the realization schemes of various memory consistency models in the McNoC systems, we have analyzed the scalability of these memory consistency models on the basis of different application workloads. The scalability analysis is performed in the systems with 1 to 64-cores. The application workloads are developed and mapped on the different sizes of networks with different problem sizes. The performance metrics like *execution time, performance, speedup, overhead* and *efficiency* are evaluated as a function of network size under different memory consistency

models. The scalability analysis of different memory consistency models is presented in chapter 4 with more details. The scalability study highlights some interesting and key factors which affect the performance gain under the relaxed memory consistency models over the stricter memory consistency models, for example, the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio and the problem size. When the system size is scaled up, a significant performance improvement is observed under the relaxed memory consistency models like PRC and RC models over the strict SC model due to the additional reordering and relaxation among the shared memory operations.

## 1.4 Contributions and Outline of Thesis

This section summarizes the contributions of the conducted research work and presents an outline of the thesis. In general, the memory consistency issue in the McNoC systems is addressed and the scalability analysis of different memory consistency models is analyzed. All the reported results have been accepted or published in various international conferences and journals. The main contributions of our research work are summarized in the following:

The architectural support of the SC, TSO, PSO, WC, RC, and PRC models are provided in the McNoC systems. The realization scheme of the SC model in the McNoC systems is discussed in [19]. According to the realization scheme of SC model, the *program order* is enforced by stalling the processor on the issuance of each memory operation till its completion. A processor in each node of the network issues the next operation in the program on the completion of previously issued operation. The memory operations are executed as per program order. The *sequential order* is enforced by sequentially accessing the critical/shared memory locations by using the common lock among the multi-processors in the system.

The realization scheme of the TSO models in the McNoC systems is introduced in [21]. The required global orders under the TSO model are enforced on the memory operations by using the *Write Transaction Counter* (*WTC)* and *Write Address Stack* (*WA-Stack)* hardware structures in each node of the network. The *WTC* is used to keep track of the outstanding write operations issued by a processor in the system. The *WTC* is checked at the issuance of each write operation and the issuance of a write operation is delayed by stalling the processor till the completion of previously issued outstanding write operation, which is indicated by the zero value of *WTC*. This enforces the ordering constraints on the memory operations in the case of *a write followed by a write* operation. The processor is stalled on the issuance of a read operation till its completion. This enforces the global orders under the TSO model in the cases of *a read followed by a read* operation and *a read followed by a write* operation. The *WTC* is also checked at the issuance of each synchronization operation and the issuance of a synchronization operation is delayed by stalling the processor till the completion of previously issued outstanding write operation ( e.g., *WTC=0)*. The *WA-Stack* is used to keep track of the addresses of the outstanding write operations issued by a processor in the system. The *WA-Stack* constrains the outstanding operations issued by a processor to the same location in the memory. The outstanding operations issued by a processor to the same location in the memory are executed as per *program order*. This ensures the parallel program correctness, i.e., a read

operation must always fetch the most recent value of a memory location. Overall, the global orders required under the TSO model are enforced by stalling the processor and using *WTC* and *WA-Stack* hardware structures in each node of the network.

The realization scheme of the PSO models in the McNoC systems is presented in [21]. The required global orders under the PSO model are also enforced on the memory operations by using the *WTC* and *WA-Stack* hardware structures in each node of the network. In contrast to TSO model, the PSO model allows reordering and relaxation among the write operations which are issued by a processor in the system. Therefore, the *WTC* is not checked at the issuance of each write operation and the issuance of a write operation is not delayed till the completion of previously issued outstanding write operation. The processor is stalled on the issuance of a read operation in the program till its completion. This enforces the required global orders under the PSO model in the cases of *a read followed by a read* operation and *a read followed by a write* operation. The *WTC* is checked at the issuance of each synchronization operation and the issuance of a synchronization operation is delayed by stalling the processor till the completion of previously issued outstanding write operation. The *WA-Stack* is used to constrain the outstanding write operations which are issued by a processor to the same location in the memory. In contrast to the TSO model, the *WA-Stack* is also checked at the issuance of each write operation. This is due to the fact that the PSO model allows additional reordering and relaxation among the memory write operations. Hence, the outstanding write operations issued by a processor to the same location in the memory are constrained to complete in the order specified by the program. In general, the global orders required under the PSO model are enforced by stalling the processor and by using the *WTC* and *WA-Stack* hardware structures in each node of the network.

The realization scheme of the WC model in the McNoC systems is discussed in [1][17]-[19]. The realization scheme of the WC model uses a *Transaction Counter* (*TC*)-based approach. However, the outstanding operations of a processor to the same location in the memory are not constrained as per program order. Later on, the realization scheme of the WC model is enhanced in [23] to constrain the data operations issued by a processor to the same memory location by using an Address Stack (*A-Stack*) in each node of the network. A *TC* is used in each node of the network to keep track of the outstanding data (read, write) operations issued by a processor in between the two consecutive synchronization points. The *TC* is checked at each synchronization point and the issuance of a synchronization operation is delayed by stalling the processor till the completion of previously issued outstanding data operations, which is indicated by a zero value of *TC*. This enforces the global order under the WC model in the case of a data operation followed by a synchronization operation. The processor is stalled on the issuance of a synchronization operation till its completion. This enforces the required global orders under the WC model in the case of a synchronization operation followed by a data operation and a synchronization operation followed by a synchronization operation. The other global orders required under the WC model among the synchronization operations are enforced by sequentially accessing the synchronization variables among the multi-processors in the system. The *A-Stack* is used in [23] to keep track of the addresses of the outstanding data (read, write) operations issued by a processor in the system. The *A-Stack* constrains the outstanding operations issued by a processor to the same location in the memory. The outstanding data operations issued by a processor to the same location in the

memory are executed as per program order for the purpose of correctness. To sum up, the global orders required under the WC model are enforced by stalling the processor and by using the *TC* and *A-Stack* hardware structures in each node of the network.

The realization scheme of the RC model in the McNoC systems is described in [1] [18]. The RC model in [1][18] is realized by using two *TCs*-based approaches. In [18], the *TC1* and *TC2* are used in each node of the network to keep track of the outstanding data operations issued by a processor in the non-critical and critical sections, respectively. The *TC2* is checked at the issuance of each acquire operation and the issuance of an acquire operation is delayed by stalling the processor till the completion of previously issued outstanding data operations in the critical section. The *TC1* and *TC2* both are checked at the issuance of each release operation and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations both in the non-critical and critical sections, which is indicated by the zero value of *TC1* and *TC2*. This enforces the required global order on the memory operations in the case of a data operation followed by a release operation. The processor is stalled on the issuance of an acquire operation till its successful completion. This enforces the required global orders under the RC model in the cases of an acquire operation followed by a data operation and an acquire operation followed by a release operation. The global order required under the RC model in the case of a release operation followed by an acquire operation is enforced by sequentially accessing the synchronization variables among the multi-processors in the system. In contrast to [18], the issuance of an acquire operation does not check *TC2* in [1]. It is due to the fact that the completion of previously issued outstanding data operations in the critical section is already ensured at the previous release points, because the issuance of a release operation checks both the *TC1* and *TC2* to be zero. The realization scheme of the RC model is further enhanced in [21][23] and it is realized by using a *TC* and *A-Stack*-based approach. A *TC* is used in each node of the network to keep track of the outstanding data (read, write) operations issued by a processor in between two consecutive release points. The *TC* keeps track of the data operations which are issued both in the non-critical and critical sections. The *TC* is checked at the release point and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations both in the non-critical and critical sections, which is indicated by the zero value of *TC*. This enforces the global order under the RC model in the case of a data operation followed by a release operation. The processor is stalled on the issuance of an acquire operation till its completion. The *A-Stack* is used to keep track of the addresses of the outstanding data (read, write) operations issued by a processor in the system. The *A-Stack* works in the same way as described above under the WC model. In brief, the global orders required under the RC model are also enforced by stalling the processor and by using the *TC* and *A-Stack* hardware structures in each node of the network.

The realization scheme of the PRC model in the McNoC systems is presented in [1]. The PRC model is proposed as an extension of the RC model which allows further reordering and relaxation among the memory operations. The realization scheme of the PRC model is based on an *Acquire Counter* (*AC*), a *Transaction Counter for protected data* ($TC_{PD}$) and an *A-Stack*-based approach. The *AC* is used in each node of the network to classify the data operations issued by a processor as *unprotected* and *protected* operations. Protected data operations are protected under a lock acquire and release

operation, while the rest are the unprotected data operations. A $TC_{PD}$ is used in each node of the network to keep track of the outstanding *protected* data (read, write) operations issued by a processor before the release operation. The $TC_{PD}$ is checked at the issuance of a release operation and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding protected data operations, which is indicated by the zero value of $TC_{PD}$. This enforces the global order under the PRC model in the case of a protected data operation followed by a release operation. The issuance of a release operation is not delayed for the completion of previously unprotected data operations. The processor is stalled on the issuance of an acquire operation till its successful completion. This enforces the required global orders under the PRC model in the case of an acquire operation followed by a protected data operation and an acquire operation followed by a release operation. The other global orders required under the PRC model in the case of a release followed by an acquire operation are enforced by sequentially accessing the synchronization variables among the multi-processors in the system. The *A-Stack* is used to keep track of the addresses of the outstanding data operations issued by a processor in the system. The *A-Stack* works in the same way as previously discussed under the WC and RC models. In summary, the global orders required under the PRC model are enforced by stalling the processor and by using the $TC_{PD}$ and *A-Stack* hardware structures in each node of the network.

The performance and scalability analysis of different memory consistency models are analyzed in [22][23]. The scalability of the RC and SC models is analyzed in [22]. 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 these RC and SC models are analyzed by mapping different application workloads on the different sizes networks using different problem sizes. The scalability is studied on the basis of different types of applications and also from the perspectives of system design.

In contrast to our previous works [1][17]-[21], a comprehensive and better analysis of the ordering constraints under six different memory consistency models is presented in [23]. The scalability analysis of these memory consistency models is carried out in the McNoC systems with 1 to 64-cores. The scalability analysis is conducted by developing and mapping more application workloads on the increasing size of the network. The previous works [1][17]-[21] mainly focus on the architecture support of six different memory consistency models in the McNoC systems. In [22], the scalability of two memory consistency models (RC and SC) is analyzed, while the scalability study of all these six memory consistency models is presented in [23]. The scalability analysis and performance evaluation of these memory consistency models are carried out with new experiments to study more aspects. This study underlines some interesting and key factors which affect the performance gain under the relaxed memory consistency models over the stricter memory consistency models, for instance, the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio and the problem size.

The thesis comprises of five chapters. Chapter 1 summarizes our research work and the main contributions of the conducted research are discussed. Chapter 2 elaborates the memory consistency issue with the help of some examples and different memory consistency models are compared with respect to each other. Chapter 3 discusses the architectural support of different memory consistency models in the McNoC systems. Chapter 4 focuses on the scalability analysis of different memory consistency models in

the McNoC systems. The summary of the thesis and future directions are described in chapter 5. In the following, we further elaborate on the structure of the thesis.

**Chapter 1**

- This chapter briefly discusses the general trends and developments in the parallel computing systems. The issue of memory consistency is introduced. The main difference between the cache coherence and memory consistency is described. The scope of the thesis and the research methodology is also discussed. The research work accomplished under this thesis is summarized. The final part of the chapter, describes the main contributions and the structure of the thesis.

**Chapter 2**

- This chapter introduces the background and motivation for the memory consistency problem. The cache coherence and memory consistency protocols are briefly described. Some well known memory consistency models like: SC, TSO, PSO, WC, and RC are discussed. The ordering constraints under these memory consistency models are analyzed using different program segments in an easy and understandable way. A new memory consistency model (Protected Release Consistency, PRC) is proposed as an extension of the RC model which provides further reordering and relaxation in the shared memory operations. The ordering constraints under all these memory consistency models are analyzed and compared with respect to each other. A comprehensive analysis of the ordering constraints under the WC, RC, and PRC models is also presented with different program segments using non-overlapped, nested, and partially overlapped protected sections. Under the relaxed memory consistency models, the requirements of the ordering constraints on the outstanding operations which are issued by a processor to the same location in the memory are also described. The ordering constraints and the relaxation offered under these different memory consistency models are summarized. The memory models which are specified at the high level programming languages are also discussed. In addition, the compiler optimizations and its influence on the memory consistency models are briefly described. The final part of the chapter reviews the state of the art research on the memory consistency issue. The related work on the memory consistency is reviewed comprehensively both in the general purpose multi-processors and customized McNoC systems.

**Chapter 3**

- Chapter 3 discusses the architectural support of different memory consistency models in the customized McNoC systems. Different McNoC platforms are developed which support various memory consistency models, for example, SC, TSO, PSO, WC, RC, and PRC. The composition and functionality of each sub-system in the platforms are described. Each platform uses *Nostrum* NoC as a

communication backbone with 2-D mesh topology and deflection routing policy. Each platform supports the DSM, memory synchronization and customized processor interface. The processor interfaces are developed to provide the support for address translation, classification of memory operations, flow control and to facilitate the communication between the processor and the rest of the system. These different memory consistency models are realized in the McNoC systems by enforcing the required global orders on the memory operations by *stalling* the processor and by using the *transaction counter* and *address stack*-based *novel* approaches. The hardware structures *WTC* and *WA-Stack* are used for the enforcement of the ordering constraints on the memory operations under the TSO and PSO models. The *TC* and *A-Stack* structures are used to enforce the global orders on the memory operations under the WC and RC models. The realization scheme of the PRC model also uses *AC* at the processor interface in addition to the *TC* and *A-Stack.* The *AC* is used for further classification of data operations under the PRC model. The global orders related to the synchronization operations are also enforced under all these memory consistency models by using the sequential accesses among the multiple processors over the locks maintained in a single global address space. The ordering constraints on the memory operations which are issued by a processor to the same memory location are also described under these memory consistency models. At the end of the chapter, the impact of traffic patterns, routing algorithms, injection rate, network congestion, non-blocking caches and pre-fetching on the performance of the memory consistency models is also discussed.

Chapters 2 and 3 include some research results which are reported in the following publications:

- Abdul Naeem, Xiaowen Chen, Zhonghai Lu, and Axel Jantsch. Scalability of Weak Consistency in NoC based Multicore Architectures. *In Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2010*), pages 3497 – 3500, Paris, France, June 2010.

  *Author's contributions:* The author performed the research, experimental work and wrote the manuscript. The rest of the authors provided feedback and helped out in technical and language corrections.

- Abdul Naeem, Xiaowen Chen, Zhonghai Lu, and Axel Jantsch. Scalability of Relaxed Consistency Models in NoC based Multicore Architectures. *ACM SIGARCH Computer Architecture News*, vol. 37, no. 5, pages 8 – 15, April 2010.

  *Author's contributions:* Abdul performed the research and wrote the manuscript. The rest of the authors provided feedback and helped in editing the manuscript.

- Abdul Naeem, Xiaowen Chen, Zhonghai Lu, and Axel Jantsch. Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems. *In Proceedings of 16th*

*ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC 2011),* pages 154 – 159, Yokohama, Japan, January 2011.

*Author's contributions:* The author contributed to the problem formulation and wrote the manuscript. The rest of the authors assisted in editing the manuscript.

- Axel Jantsch, Xiaowen Chen, Abdul Naeem, Yuang Zhang, Sandro Penolazzi, and Zhonghai Lu. Memory Architecture and Management in an NoC Platform. *In Axel Jantsch and Dimitrios Soudris, editors, Scalable Multi-core Architectures: Design Methodologies and Tools,* Eds. Berlin, Germany: Springer, 2011, pages 3 – 28.

  *Author's contributions*: Axel wrote the major part of the chapter and analyzed the results. Abdul performed experiments and helped in editing a part of the research on the memory consistency models.

- Abdul Naeem, Axel Jantsch, Xiaowen Chen, and Zhonghai Lu. Realization and Scalability of Release and Protected Release Consistency Models in NoC based Systems. *In Proceedings of 14th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2011),* pages 47 – 54, Oulu, Finland, September 2011.

  *Author's contributions:* The author developed the idea and performed the research and wrote the manuscript. Discussion with the Axel helped in the development and refinement of the idea. Axel also helped in editing the manuscript. The rest of the authors helped in reviewing the manuscript.

- Abdul Naeem, Axel Jantsch, and Zhonghai Lu. Architecture Support and Comparison of Three Memory Consistency Models in NoC based Systems. *In Proceedings of 15th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2012),* pages 304 – 311, Izmir, Turkey, September 2012.

  *Author's contributions:* The author contributed to the problem formulation, planned and conducted experiments and wrote the manuscript. Axel provided help in improving and polishing the idea and technical contents. Zhonghai assisted in the language corrections.

**Chapter 4**

- This chapter focuses on the scalability analysis of different memory consistency models which are realized in the McNoC systems. The scalability analysis is performed in the systems with 1 to 64-cores. The experimental setup and the platform configuration parameters are discussed. For the experiments, we have developed the synthetic and application workloads. These workloads are used to assess and compare the performance of these different memory consistency models. Specifically, the scalability of various memory consistency models is

analyzed by mapping these applications on the increasing size of the network. The performance metrics like *execution time*, *performance*, *speedup*, *overhead* and *efficiency* are evaluated and compared for different memory consistency models as the network size is scaled up. The scaling behavior of these memory consistency models is analyzed by mapping different types of applications with different problem sizes. The study underlines some interesting and key factors which affect the performance and scalability of the relaxed memory consistency models over the stricter memory consistency models. For instance, the computation-to-communication ratio, traffic patterns, ratio of data-to-synchronization operations, problem sizes and the types and natures applications, which affect the performance improvement statistics of relaxed memory consistency models over the stricter memory consistency models.

Most of the results in chapter 4 are reported in:

- Abdul Naeem, Axel Jantsch, and Zhonghai Lu. Scalability Analysis of Release and Sequential Consistency Models in NoC based Multicore Systems, *In Proceedings of IEEE International Symposium on System-on-Chip (SOC 2012)*, Tampere, Finland, October 2012.

  *Author's contributions:* The author performed the research, conducted experiments and wrote the manuscript. The rest of the authors helped in editing the manuscript.

- Abdul Naeem, Axel Jantsch, and Zhonghai Lu. Scalability Analysis of Memory Consistency Models in NoC based Distributed Shared Memory SoCs, *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 2013, accepted for publication*. doi: 10.1109/TCAD.2012.2235914

  *Author's contributions:* The author performed the research, conducted experiments and wrote the manuscript. The rest of the authors assisted in editing the manuscript.

**Chapter 5**

- Finally, chapter 5 summarizes the thesis and the future directions are outlined.

## 1.5 Summary

This chapter has described the problem of memory consistency in the shared memory multi-core systems. Some general trends and developments in the parallel systems, for instance, parallel computations (multi-core); parallel communication (NoC) and distributed shared memory (DSM) are discussed. These trends, developments and optimizations both in the hardware and software have led to some critical issues of memory consistency and cache coherence in the multi-processor systems. The memory

consistency issue and different memory consistency models are briefly discussed in the shared memory systems. The study is mainly confined to the memory consistency issue by making it independent of the cache coherence issue in the shared memory systems. Some of the well known memory consistency models are realized in the customized McNoC systems. The chapter also describes the methods, procedures and approaches for conducting this research work. Our research mainly focuses on the architectural support and scalability analysis of different memory consistency models in the McNoC systems. The main contributions of the conducted research work are also summarized.

# Chapter 2

# Memory Consistency

This chapter presents the background and motivation for the memory consistency issue. The main difference between the cache coherence and memory consistency is briefly discussed. The memory consistency issue is mainly focused and various memory consistency models are described. The ordering constraints which are enforced on the memory operations under six different memory consistency models are analyzed and compared with respect to each other. The ordering constraints under the WC, RC, and PRC models are further analyzed comprehensively under different program segments. The ordering constraints on the memory operations issued by a processor to the same location in the shared memory are also described. Different memory consistency models are summarized on the basis of relaxation they offer among the memory operations. The memory consistency models proposed at the high level programming languages are also discussed. Finally, we review the state of the art research on the memory consistency issue both in the general purpose multi-processors and McNoC systems.

## 2.1 Background and Motivation

The parallel systems supporting shared memory are widely accepted in many areas of computing. Within the parallel computing systems, the computation is parallelized among the many-cores. These standalone computation elements in the parallel systems work simultaneously to solve the larger problems. The parallelism in computation is driven by the physical limit of semiconductor which does not allow further frequency scaling due to the high power consumption. Thus, the parallel computing has attained much popularity in the form of multi-cores, multi-processors, MPSoCs, CMPs, clusters, and MMPs. The IP-cores in the multi-core systems can be integrated by a common communication infrastructure. The classical shared busses face several issues like poor scalability and reliability when they are used as a communication medium among the IP-cores in the systems. Alternatively, Network-on-Chip (NoC) is a promising solution to integrate the IP-cores in a scalable and reliable manner in the modern Systems-on-Chip (SoCs). The NoC can follow the theories applied to the networking and can act as the on-chip communication medium among the IP-cores in the system. In the NoC-based systems, the performance can be tremendously enhanced due to the parallel communication compared to the systems based on the classical buses. The NoC-based multi-core systems can use the shared memory abstraction to exchange the values/data among the IP-cores. The message passing abstraction can also be used to initiate the communication among the IP-cores in the system. However, it is preferable to use both

the shared memory and message passing to combine the benefits of both these approaches in the systems. The systems supporting the shared memory abstraction must use a suitable organization for the shared memory. The shared memory organization can play a very important role from the scalability and performance perspective of the systems. The Distributed Shared Memory (DSM) is preferred over a single centralized shared memory organization due to several reasons. Within the DSM systems, the computational cores can operate on the physically distributed memories which are maintained in a single global address space. The system which combines the benefits of the parallel computation (multi-core), parallel communication (NoC) and memory architecture (DSM) could be used as a promising solution in the future SoCs designs.

The parallel systems which support the shared memory abstraction are facing the critical/challenging issue of memory consistency. The cache coherence issue can also be observed in the systems which use the data caches to reduce the average memory access time. The cache coherence problem is due to the different cached copies of the same shared data in the multi-processor systems. Several cache coherence protocols such as MSI (Modified-Shared-Invalid), MESI (Modified-Exclusive-Shared-Invalid), MOSI (Modified-Owner-Shared-Invalid), etc., are used to resolve the issue of cache coherence in the multi-processor systems. In contrast, the memory consistency problem is due to the unconstrained memory operations which lead to the unexpected behavior of the shared memory multi-processor systems. In the remaining chapter, we discuss the memory consistency issue in more details.

## 2.2 Coherence vs. Consistency

The memory consistency and cache coherence are two different problems of similar nature in the shared memory multi-processor systems. As discussed earlier, the cache coherence problem arises when multiple processors are using the data caches which sometimes retain different versions of the same shared data. As a result, a processor fetches the wrong data from the cache instead of the most recently written value from a memory location. It is mandatory both for the memory coherence and consistency that the memory operations should be accomplished in the program order and a read operation must return the correct or most recent value of a memory location. In the multi-processor systems, a value written to a memory location by a processor must be propagated to all other processors before they access this value/location (e.g., write propagation or write atomicity). Another important aspect is that all the processors must agree on the same value for a given memory location, i.e., they should not observe the different values for the same memory location (e.g., write serialization). The coherence problem could be observed in the data caches either using the write-through or write-back policy. The advantage of the write-through caches is that they propagate every write of a processor and make it visible on the shared communication medium and the main memory is kept updated. However, it consumes more memory bandwidth due to the more traffic offered compared to the write-back policy. The write-back policy reduces the traffic as it does not allow every write to propagate on the shared communication medium. The cache coherence problem is resolved by using the coherence protocols. There are two different types of mechanisms or protocols (e.g., snooping-based cache coherence, directory-based

cache coherence) which are used to keep the data caches coherent and updated. The *snooping*-based cache coherence protocol relies on the bus snooping by the cache controllers in the system. It is based on broadcasting in the system. Therefore, it is suitable for the traditional shared bus-based multi-core systems, but it is un-scalable in the network-based multi-core systems. Alternatively, the *directory*-based cache coherence protocol is used which is suitable for the modern multi-processor systems using sophisticated interconnection network as the communication medium. The directory-based cache coherence protocol maintains the state information about the cache blocks and sends messages to invalidate or update the cached copies of the requested block. However, the directory-based coherence protocols have some challenging issues like extra coherence traffic, directory overhead, additional latencies and complexities.

We consider an example of the directory-based cache coherence protocol in the network-based multi-processor systems. The write-back caches with write-allocate and invalidation-based MSI cache coherence protocol is considered. The *requesting* node issues the request and the *directory* node hosts the main memory of the block. The *dirty* node has the cached copy of the block in the modified or dirty state, while the *sharer* node has the copy in the shared or valid state. On the *read miss*, the request is forwarded to the directory node to check the state information on the requested block. If the main memory has a *clean* copy of the block, then it is provided to the requesting node. If it has the *dirty* (stale) copy, then the request is forwarded to the dirty node which maintains a copy of the requested block in the modified state. The dirty node then flushes the data to provide the data to the requesting node (cache-to-cache transfer) and updates the main memory. In the case of a *write hit*, when the cache block is in the *shared* state, the state is then updated to the modified state. If there are some sharer(s) of the same block, then their cached copies are invalidated. On the *write miss*, if the main memory has a *clean* copy and there is *no sharer*, then the block is allocated to the requesting node. If the sharer(s) are available, then cached copies of all the sharers are invalidated and the block is allocated to the requester. In case that the main memory has a *dirty* copy, the dirty node flushes the data by updating the cache of the requesting node and the main memory is updated as well. Also, the directory information is updated.

In contrast to the cache coherence issue, the memory consistency issue is related to the ordering constraints on the shared memory operations. When the memory operations are not properly controlled then it leads to the inconsistent or unexpected behavior of the shared memory systems. The memory consistency models enforce the ordering constraints on the shared memory operations to ensure the parallel program correctness in the system. Our study targets the hard real time applications and some other applications where caches are not used or when these two problems have very different requirements on the size of the consistency object and the cache block. In such situations, independent implementation schemes for these two problems are preferred. Therefore, the main focus of our study is on the memory consistency problem and we are decoupling it from the cache coherence problem. The memory consistency models are realized independent of the cache coherence protocols in the multi-core systems. We argue that decoupling will allow for better optimization, treatment, and concentration on these two critical problems (coherence and consistency) in the shared memory systems.

## 2.3 Memory Consistency

The system designers must know about the exact behavior of the shared memory systems both at the hardware and software levels. The correct behavior of shared memory system must be ensured from the perspective of read and write operations which are issued by multiple processors in the system. The 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 reordering, register allocation). These optimizations are the key means to enhance the performance of the systems by efficient utilization of the available resources. As an example of the hardware optimization, the write buffer can overlap the latency of a write operation with the subsequent read operation. The data caches can make the memory access time faster on average using temporal or spatial locality of references. The interconnection network can parallelize the memory transactions in the system. On the software side, the compiler can reorder and transform the code for the efficient execution. The compiler can also allocate and assign the memory variables to the fast access registers of the processor and the processor then fetches the value from these registers instead of accessing the memory locations.

On the other hand, reordering of the memory operations due to these system optimizations sometimes might lead to the incorrect or unexpected behavior of the shared memory systems for the programmer [10]. We illustrate this by an example. For instance, Table 2.1 considers two different program segments mapped on the processors $P_1$ and $P_2$, respectively in the system. We assume that each processor update (writes to) the shared memory location in the remote node and reads from the shared memory location within the same node. We further assume that the system supports the Distributed Shared Memory (DSM) organization and the outstanding memory transactions are allowed in the system. The memory operations issued by a processor can be reordered in the *interconnection network* and a read operation might not observe the value updated by another processor. Due to the reordering of the memory operations, inconsistent result (e.g., Reg1=0, Reg2=0) is produced, which is never expected by the programmer. Alternatively, the *compiler* in each node can also reorder the independent instructions, because there is no data dependencies. As a result, the read operations can be executed before the updated data are available in the memory, which also produces the surprising result (e.g., Reg1=0, Reg2=0). The read operations issued by the processors could not return the most recently updated values from the shared memory. Hence, the parallel program correctness could be violated. The parallel program correctness must be ensured for the correct behavior of the shared memory systems. The parallel program must be executed correctly according to its specification, i.e., the correct results must be produced by the parallel program for a given set of the input data. In terms of memory operations, a read operation must always fetch the most recent and correct value written to a memory location.

**Table 2.1.** Motivation for the memory consistency

| Initially, X = Y = 0; | |
|---|---|
| **Processor $P_1$** | **Processor $P_2$** |
| Y = data1;<br>Reg1 = X; | X = data2;<br>Reg2 = Y; |

The memory operations issued by a processor must be constrained carefully for the expected behavior of shared memory systems. As discussed earlier, a read operation of a processor must always return the most recent or correct value of the memory location in the multi-processor systems. The parallel program correctness must be ensured up to the expectation of a programmer. The *memory consistency* specifies the execution order of the memory operations for the expected behavior of the shared memory systems. It is related to the enforcement of the ordering constraints on the shared memory operations. These constraints are imposed on the order of memory operations for the legal outcome (or execution) of the parallel program. There are various memory consistency models available in the literature [9]-[14]. These memory consistency models correspond to the enforcement of the different ordering constraints on the memory operations. The SC model [9] is one of the intuitive memory consistency models which impose more restrictions on the order of memory operations. The SC model does not allow reordering and overlapping among the memory operations which are issued by a processor in the system. Due to the strict nature, the SC model cannot exploit the system optimizations both in the hardware and software. Therefore, *relaxed* or weaker consistency models [10]-[14] are proposed to enhance the system performance by enforcing fewer restrictions on the order of shared memory operations. The relaxed memory consistency models exploit the system optimizations compared to the stricter memory consistency models.

## 2.3.1 Interfaces

A memory consistency model can be implemented at the interface between a processor and shared memory system. Different types of interfaces could be developed for the processors in the shared memory systems. The processor interface could communicate with the memory controller to control the flow of transactions from the processor. This can also help in the enforcement of the ordering constraints on the memory operations under the implementation of a memory consistency model. Different interfaces can be used at various levels to implement the consistency and coherence protocols. These hardware interfaces can be used along with the semantics of programming languages to implement different memory consistency protocols. The enforcement of the ordering constraints on the memory operations under a given memory consistency model is the local property of a processor in the system. Thus, the processor interface in each core of the system can be used to impose the ordering constraints on the memory operations which are issued in the core. We illustrate in Figure 2.1, a conceptual view for the different types and levels of interfaces between a programmer and shared

memory system. The processor issue the memory operations/transactions in the order in which they appear in the program. The first level or the high level processor interface can be used to communicate with the processor via the memory controller to control the flow of operations for the enforcement of the ordering constraints under a specified memory consistency model. We call this as memory consistency interface. The address translation, classification of memory operations and flow control can also be implemented at this level of interface. The next level of interface can be used to implement the cache coherence protocols (cache coherence interface). A directory could be maintained for the implementation of the directory-based cache coherence protocols. Also, the additional controllers/mechanisms can be implemented to handle the extra cache coherence protocols in the parallel fashion. The interconnect interface can be used for specifying the interconnection network protocols, arbitrations, buffering the shared memory requests and responses.

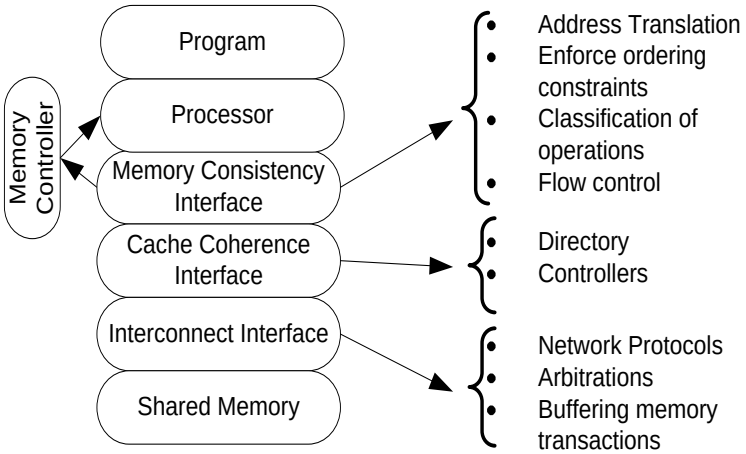



**Figure 2.1.** Different interfaces between a processor and the shared memory system.

## 2.4 Memory Consistency Models

A memory consistency model (often called memory model) determines the order of execution of the shared memory operations for the correct behavior of multi-processor systems. A memory consistency model is a contract between the programmer and system that specifies a set of rules under which the memory operations are allowed or not allowed to be reordered with respect to each other. These rules must be followed by the processors for the consistent behavior of shared memory systems. A memory consistency model has a system wide relationship and it simultaneously affects the hardware designers, software developers, programming languages and compiler designers.

Different memory consistency models are proposed which are available in the literature. These various memory consistency models enforce different ordering constraints on the shared memory operations. The memory consistency models have evolved from the stricter to weaker or relaxed memory consistency models. The stricter models come first in

the evolutionary process and then the relaxed memory models are introduced by rectifying the shortcoming in the earlier restrictive memory models. The strict memory consistency model (also called *atomic* model or *linearizability*) does not allow any kind of reordering among the memory operations. Under the strict memory consistency model, the memory operations of a processor are executed atomically in the system. In other words, a memory operation issued by a processor is executed at once and it seems that all other processors in the system have suspended their activities. The atomic operation can be ensured either by sequentially accessing a memory location or by using the read-modified-write operation. The operations of multiple processors are interleaved under the atomic or strict memory consistency model to serially access a memory location. This serializability does not allow any kind of overlapping among the memory operations and the parallelism of the concurrent systems cannot be exploited. The SC model is also a strict memory consistency model. It enforces a *total order* on the memory operations by enforcing the program order and sequential order on the memory operations. The SC model constrains the operations issued by a processor so that these must be executed in the order in which they are issued in the program (program order). The operations of different processors are also constrained to access the critical memory locations in a sequential order. The *stricter* memory consistency models enforce more restrictions on the order of shared memory operations. Therefore, system optimizations both in the hardware and software cannot be exploited due to the strict nature of these memory models. As a result, *relaxed* or *weaker* memory consistency models are proposed in the literature. These relaxed memory consistency models enforce less ordering restrictions on the shared memory operations. Under the relaxed memory consistency models, the memory operations can be reordered with respect to each other and the parallel program correctness is also ensured. The relaxed memory consistency models effectively utilize the system resources compared to the stricter memory consistency models and the system performance is enhanced at the reasonable cost. The IBM-370 model [71] allows reordering and relaxation in the case of a *write followed by a read* operation to different locations in the memory. This is a similar kind of relaxation which is enabled under the TSO model. The IBM-370 model uses serialization instructions as the *safety net* in the program among the memory operations. The IBM-370 model is a strict model which restricts a processor to read the value of its own write operation before propagating it to all other processors in the system. The *causal consistency* model is a weaker model than the SC model. The memory operations are constrained, which are causally related to each other. Those memory operations which are *not* causally related are called concurrent operations and they can be reordered with respect to each other. The memory system is causally consistent, when the causal write operations are observed in the same order by all the processors in the system, while the concurrent write operations are observed in a different order. The pipelined random access memory (PRAM) consistency model (also called FIFO consistency) is another relaxed memory model. According to the PRAM consistency model, the write operations issued by one processor are observed by all processors in the system in the same order in which they are issued. Also, the write operations issued by different processors can be observed by a processor in the different order. The TSO model is also a relaxed memory consistency model which allows reordering and relaxation among the memory operations in the case of *a write followed by a read* operation. The PSO model in addition allows reordering among the memory

operations in the case of *a write followed by a write* operation. The TSO and PSO models utilize the write buffer optimization which hides the latency of write operations by pipelining these operations with the subsequent read operations. The *processor consistency* (PC) model [69][72] relaxes the ordering constraints in the case of a *write followed by a read* operation. This relaxation is similar to that which is allowed under the IBM370 and the SPARC V8 TSO models. According to the PC model, each processor in the system has its own copy of the entire memory. Multiple copies of a single memory are replicated in the processing nodes of the system. Two conditions must be satisfied on a given processor under the PC model. First, all the previously issued read operations must be completed before a read operation is allowed to be performed. Second, all the previously issued read or write operations must be completed before a write operation is allowed to be performed.

The WC model categorizes the memory operations as *data* and *synchronization* operations. The data (read, write) operations issued by a processor in between the two consecutive synchronization points can be reordered and overlapped with respect to each other. The data and synchronization operations cannot be reordered with respect to each other. The operations to the special synchronization variables are sequentially consistent. The RC model classifies the synchronization operations as *acquire* and *release* operations. The data operations issued by a processor in between the two consecutive release points can be reordered and overlapped with respect to each other. The data and acquire operations can be reordered with respect to each other under certain conditions. Also, the data and release operations can be reordered with respect to each other under certain conditions. We discuss these ordering constraints under the RC model with more details in the later part of this thesis (Chapter 3). The PRC model classifies the data operations as *unprotected* and *protected* operations. The protected data operations are protected under the acquire and release operations on a lock, while the unprotected data operations are not protected under the acquire and release operations. The unprotected data operations issued by a processor can be reordered and overlapped with respect to each other, with the subsequent release and unprotected data operations. Also, the data and acquire/release operations can be reordered with respect to each other under certain conditions. The execution of the acquire and release operations to the synchronization variables must be sequentially consistent.

There are some variants of the RC model [28]-[31] which provide additional pipelining and overlapping among the memory operations. For example, the *eager RC* model [28], *Lazy RC* model [29], *Entry RC* model [30] and *Scope consistency* model [31] permit further reordering and relaxation in the memory operations in contrast to the RC model. In the *Eager RC* model [28], the processor delays the propagation of all its modifications to the shared data until the release points. A processor sends all the updates belonging to the same destination in a single message at the release points in the program. This reduces the number of messages and the memory access time under the eager RC model is also decreased compared to the RC model. In the *Lazy RC* model [29], the propagation of modifications by a processor is further delayed till the acquisition of a lock by another processor. The processor observes all the required and needed modifications at the time of lock acquire. The number of messages is reduced under the lazy RC model compared to the eager RC model. The *Entry RC* [30] further categorizes the lock acquires into two different modes (exclusive and non-exclusive). When a processor acquires a lock

in an exclusive mode, no other processor can acquire the same lock at the same time. When a processor acquires a lock in a non-exclusive mode, another processor can acquire the same lock at the same time only when it performs the read operations in their critical sections. But note that, when a lock is acquired in an exclusive mode, the next non-exclusive acquire of that lock by any other processor is allowed only after the lock owner performs the release operation. The *Scope consistency* [31] simplifies the implementation of the Entry RC model. A scope is defined by all the critical sections guarded by the same lock in the system. Within a scope, the global order is enforced. In contrast to the Lazy RC model, the scope consistency reduces the false sharing. *False sharing* occurs when multiple processors in the system write to the disjoint fields of the same coherence block and unnecessarily invalidate each other. The full block seems to be shared although only a part of it is shared in reality. The scope consistency uses the page as the coherence atom compared to the Entry RC model.

Some other memory consistency models are implemented in the commercial architectures which allow reordering and relaxation among the data operations issued by a processor to the different locations in the memory. Among these include: Digital Alpha [54], SPARC V9 relaxed memory order (RMO) [73] and IBM PowerPC [52][53]. The WC and RC models classify the memory operations based on the *popper labeling* of these operations [62], while the Alpha, RMO and IBM PowerPC models use different types of explicit *fence* instructions to impose the ordering constraints on the memory operations. The DEC Alpha memory consistency model [54] provides two different types of fence instructions for this purpose. The Memory Barrier (MB) enforces the program order among all the read and write operations which are issued by a processor in the system. The Write Memory Barrier (WMB) enforces the ordering constraints only among the write operations of a processor in the system. The Alpha memory consistency model also enforces the ordering constraints among the read operations issued by a processor to the same location. The SPARC V9 RMO model [73] further refines the TSO and PSO models. In contrast to the PSO model, the RMO model further relaxes the ordering constraints in the cases of a *read followed by a read* and *a read followed by a write* operation. The RMO model supports four different types of fence instructions which enforce the ordering constraints among the memory operations. A single instruction is used to specify various types of fences (memory barriers) by using the corresponding bits in an op-code. The RMO model also enforces the ordering constraints in a situation when a *read is followed by a write* operation and among the writes operations issued to the same location in the memory. The semantics of the IBM PowerPC model [52][53] is identical to the WC model compared to the Alpha and RMO models. The PowerPC model supports a single fence instruction (SYNC) to enforce the ordering constraints on the memory operation. The conflicting operations from different processors are also constrained under the PowerPC model. The PowerPC model also enforces the ordering constraints on the write operations issued to the same location in the memory. For more details on the Alpha, RMO and PowerPC models, we refer you to the [62][63].

## 2.5 Memory Consistency Models Covered under this Thesis

This thesis focuses on the architectural support and scalability analysis of some well known memory consistency models: SC, TSO, PSO, WC and RC models in the McNoC systems. The PRC model is a newly proposed memory consistency model. The PRC model is introduced as an extension of the well known RC model. It further classifies the data operations as *unprotected* and *protected* operations. The PRC model offers more reordering and relaxation among the shared memory operations compared to the RC model. We do not consider the implementation of other variants of all these memory consistency models which are available in the literature. However, we have provided a brief overview of these memory consistency models in the previous section. We provide a detailed description of all these six different memory consistency models in this section. The ordering constraints under these memory consistency models are analyzed with easy and understandable examples. The ordering restrictions under these models are compared with respect to each other under different program segments. The global orders (e.g., the ordering constraints to be enforced on the memory operations) under these different memory consistency models are formulated which are then used in the realization schemes of these models in the McNoC systems. The novel realization schemes of all these six different memory consistency models in the McNoC systems are discussed in the next chapter.

### 2.5.1 Sequential Consistency Model

The Sequential Consistency (SC) (also called *strong ordering*) model is a natural choice for the multi-processor systems. The SC model is defined by Lamport [9] as:

"A multiprocessor system is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program"

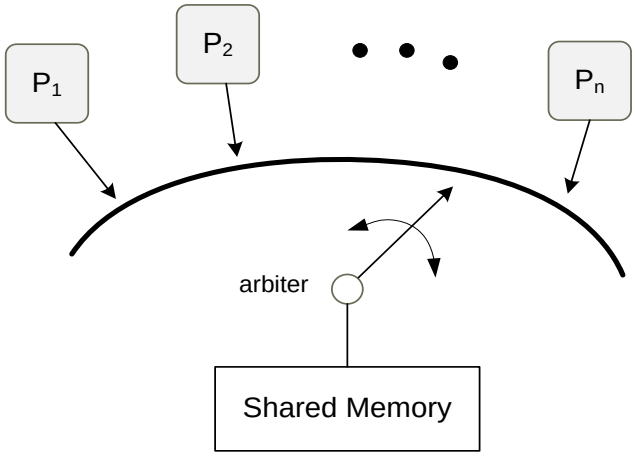



**Figure 2.2.** Abstract view of the SC model.

The SC model enforces the *program order* among the operations of an individual processor. Each processor executes the memory operations in the order specified by the program. The *sequential order* among the multiple processors on the critical resource (shared memory) is also enforced. The memory operations issued by different processors should access the critical memory locations in a single sequential order. Due to the enforcement of the program order and sequential order, the SC model ensures a *total order* on all the memory operations which are issued in the system.

As given in Figure 2.2, the SC model provides a straightforward view of the shared memory system to the users/processors. All the processors in the system can access the shared memory one at a time by following an arbitration policy. The memory operations of an individual processor must be issued and completed in the program order. A centralized arbiter can synchronize all these processors over the shared memory access in the system.

We consider an example under the SC model as given in Table 2.2. The global variables X and Y are initialized to zero. The processor $P_1$ writes to the X and Y memory locations, while the processor $P_2$ reads these locations. The possible interleaving of operations could produce several outcomes/executions of this parallel program. When all the read operations of $P_2$ occur before the write operations of $P_1$, then the possible outcome returns the initial values of X and Y locations (e.g., Reg1=0, Reg2=0). The execution is sequentially consistent. Also, the outcomes (e.g., Reg1=0, Reg2=data1) and (e.g., Reg1=data2, Reg2=data1) do not violate the semantic of sequential consistency. However, the outcome (Reg1=data2, Reg2=0) is not a sequentially consistent execution, because this is not possible under the SC model. The SC model enforces the program order on the operations issued by an individual processor. Therefore, when the processor $P_2$ returns the Y value in Reg1, this implies that the processor $P_1$ must have completed the write operation on X before the write operation on Y in the program order. Hence, the processor $P_2$ must observe the updated value of X under the SC model.

**Table 2.2.** Example for the SC model

| Initially, X = Y = 0; | |
|---|---|
| **Processor $P_1$** | **Processor $P_2$** |
| X = data1;<br>Y = data2; | Reg1 = Y;<br>Reg2 = X; |

Figure 2.3(a) is adopted from [20][43][67]. It illustrates the enforcement of the ordering constraints on the memory operations under the SC model. The variables (*U, V, W, X, Y, Z* and *S*) are the global (shared) variables. The read and write operations are

performed by the processors on these global variables in the system. The variables on the left side of the assignment operators are updated (written) and those on the right side are read. The operations on synchronization variable lock (*S*) are used to achieve a sequential order on the critical memory references among the multiple processors in the system. The synchronization operations under the SC model are considered in order to make it comparable to the other relaxed memory consistency models. An *arrow* between the 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 the variable *V* in the program and an operation on *U* is completed before the issuance of an operation on *V*. The two operations are not permitted to be reordered and overlapped with respect to each other.

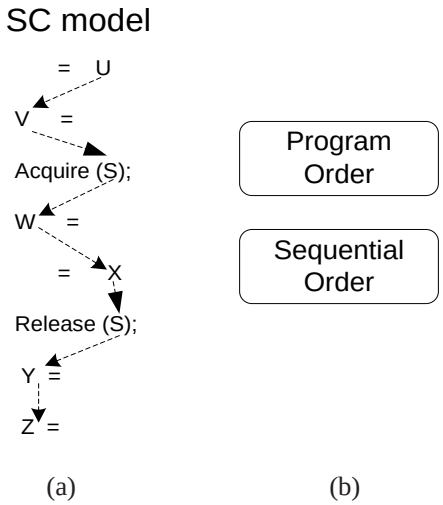



**Figure 2.3.** The SC model: a) Ordering requirements; b) Global orders.

According to the SC model, the memory operations are completed in the *program order* (e.g., order specified by the program). On the completion of a previously issued operation, the next operation is issued by a processor in the program. In other words, the issuance of a memory operation is delayed till the completion of a previously issued operation in the program. The *sequential order* is maintained by interleaving the operations on lock (*S*) among the multiple processors in the system. The global orders to be enforced under the SC model are shown in Figure 2.3(b). We refer to these global orders in chapter 3 under the realization scheme of the SC model in the McNoC systems.

The SC model is implemented in the multi-processor arrangement of the MIPS R10000 [79][80] commercial RISC microprocessor. The MIPS is a superscalar processor of degree four using super pipelining and speculative execution. The multi-processor arrangement uses write-back data caches with both the snooping and directory-based cache coherence protocols.

## Limitations of the SC model

The SC model enforces the strict ordering constraints on the memory operations. It does not allow reordering and overlapping among the memory operations issued by a processor

in the system. Consequently, the SC model cannot utilize the system optimizations both in the hardware (e.g., write buffer, cache, interconnection network) and in the software (e.g., compiler reordering, register allocation) due to its strict nature. The SC model restricts even those reordering and overlapping among the memory operations which do not lead to the incorrect behavior of the shared memory systems. The SC model cannot take advantage of the potential performance benefits in the shared memory systems. Therefore, various *relaxed* or *weaker* memory consistency models are proposed to rectify the shortcomings of the SC model. In contrast to the SC model, the relaxed memory consistency models enforce less ordering constraints on the memory operations which are issued by a processor in the system. The relaxed memory consistency models do not enforce the strict ordering constraints on the memory operations issued by a processor in the system. These memory models allow reordering and overlapping among the memory operations issued by a processor in the system. The memory operations which are issued by a processor can be executed out-of-order. But, it should not violate the parallel program correctness. A number of relaxed memory consistency models are available in the literature. The relaxed memory consistency models offer relaxation among the memory operations at different levels, i.e., different relaxed memory consistency models offer various relaxations among the memory operations. In the following sub-sections, we present some of the relaxed memory consistency models like: TSO, PSO, WC, RC, and PRC. We analyze the ordering constraints which are enforced on the memory operations under these memory consistency models. We also compare the ordering constraints under these weaker memory consistency models with respect to each other.

### 2.5.2 Total Store Ordering Model

The Total Store Ordering (TSO) model is relaxed model compared to the SC model. It is implemented in the SPARC and x86 architectures [27][73]. The TSO model was proposed to exploit the potential performance benefits of the write buffers (or store buffers) as a hardware optimization in the in systems architectures. The write buffer can reduce the memory access time by overlapping the latency of write operations with the latency of subsequent read operations. The SC model could not exploit the write buffers, since the architectures using the write buffer can reorder the memory operations in the case of *a write followed by a read* operation. In such systems, the execution of a parallel program may not be sequentially consistent under some situations.

The TSO model compared to the SC model allows reordering and relaxation among the memory operations in the case of a *write followed by a read* operation. The subsequent read operation can be reordered and overlapped with the outstanding write operation issued earlier in the program. Under the TSO model, a read operation can fetch the data of a previously issued outstanding write operation from the write buffer instead of referring to the memory. Hence, the memory access latency is reduced under the TSO model compared to the SC model by pipelining the latency of write operation with the latency of a subsequent read operation.

We consider an example under the TSO model as given in Table 2.3. The code snippet is adopted from [10]. It represents the Dekker's algorithm which is used to solve the problem of critical sections. The ordinary global variables (e.g., Flag1, Flag2) are used for the purpose of synchronization over the execution of critical sections under the $P_1$ and $P_2$, respectively. Both these flag variables are initialized to zero. We assume that

the system supports the write buffers to hide the latency of the write operations. The write buffers are safe to use in the uni-processor systems, because in such systems by passing the value from the write buffer by the subsequent read operation does not affect the correctness of a program. However, in the multi-processor systems, it could violate the parallel program correctness. On the issuance of a write operation on the flag variable, a processor stores the data in the write buffer and proceeds without waiting for the completion of the write operation. Incase of both the processors, the subsequent read operation on the flag could be reordered with respect to the previously issued write operation on the flag which is pending in the write buffer. We consider a situation, when both the processors could insert their write operations on the flags in the write buffers and the subsequent read operations bypass the earlier write operations on the flags. Hence, both these processors could observe the initial values of their flags (e.g., Flag1=0, Flag2=0) and they could enter into their critical sections simultaneously. This is illegal and the correctness of parallel program could be violated.

**Table 2.3.** Example for the TSO model

| Initially, Flag1 = Flag2 = 0; | |
|---|---|
| **Processor $P_1$** | **Processor $P_2$** |
| Flag1 = 1;<br>If (Flag2 = 0)<br>critical section | Flag2 = 1;<br>If (Flag1 = 0)<br>critical section |

Alternatively, as we argued earlier in chapter 1 that the synchronization supports among the multiple processors must be provided both at the hardware and software levels in the shared memory multi-processor systems. The underneath hardware must provide a back up for the synchronization primitives which are used at the software level.

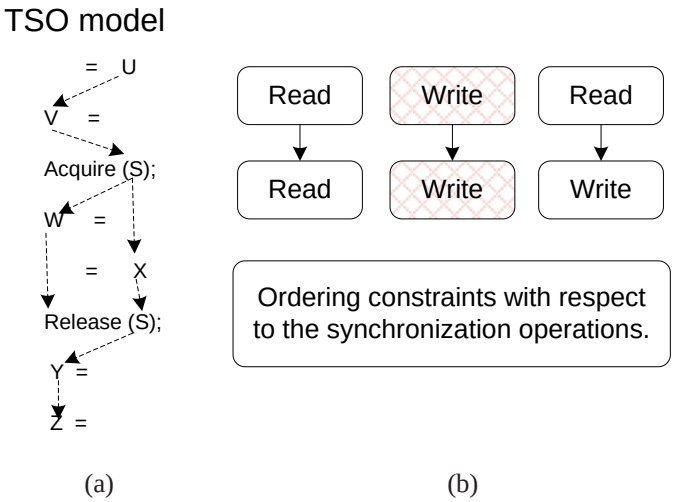



**Figure 2.4.** The TSO model: a) Ordering requirements; b) Global orders.

As demonstrated in Figure 2.4(a), the TSO model allows the write operation on (*W*) to be *reordered and overlapped* with respect to the subsequent read operation on (*X*), while this is not allowed under the SC model as shown in Figure 2.3(a). Both the TSO and SC models enforce the ordering constraints in the cases of *a read followed by a write* operation (*U→V*), *a write followed by a write* operation (*Y→Z*) and *a read followed by a read* operation. In addition, the ordering constraints with respect to the synchronization operations must also be enforced. The global orders to be enforced on the memory operations under the TSO model are given in Figure 2.4(b).

### Fences

The ordering constraints on the memory operations under the TSO model in the commercial systems are enforced by using the fence instructions [26][73]. The fence instructions are the non-memory reference instructions and are only used for imposing the ordering constraints on the memory operations. These fence instructions are inserted in between the read and write operations in the program by a programmer or could be inserted by the compiler to enforce the required global orders on the memory operations. Apart from the TSO model, the fence instructions are also used in the other relaxed memory models. The system supporting TSO model allows relaxation in the case of *a write followed by a read* operation compared to the SC model. However, the programmer can make these operations in order by inserting a fence instruction in between these operations. A fence instruction establishes the program order among the memory operations which appears before and after it. We consider an example in Table 2.4 which uses the fence instructions in the program segments. The fence instruction is inserted in between the read and write operations of $P_1$, while another fence instruction is placed between the two consecutive write operations of $P_2$. On processor $P_1$, the issuance of write operation on *X* is delayed till the completion of the earlier read operation on *Y*; similarly, on $P_2$ the two write operations on (*Y, Z*) are also constrained to be executed in the program order.

**Table 2.4.** Fence instructions

| Initially, X = Y = Z = 0; | |
|---|---|
| **Processor P1** | **Processor P2** |
| Reg1 = Y;<br>Fence<br>X = data1; | Y = data2;<br>Fence<br>Z = data3; |

## 2.5.3 Partial Store Ordering Model

The Partial Store Ordering (PSO) model is another relaxed memory consistency model. In contrast to the TSO model, the PSO model allows reordering and relaxation

among the memory write operations which are issued by a processor in the system. The memory access latency could further be reduced by pipelining multiple write operations under the PSO model. The PSO model like TSO model also exploits the write buffer optimization in the system architecture. The PSO model is proposed as an extension of the TSO model which is also implemented in the SPARC architectures [26][73]. Both the TSO and PSO models enforce the ordering constraints on the memory operations by using different kind of *fence* instructions. As discussed earlier, these fence instructions are supplemented in the program to enforce the ordering constraints on the memory operations. For example, store barrier (STBAR) and memory barrier (MEMBARs) are the fence instructions which are used in the SPARC architectures [26][73] to enforce the required ordering constraints under the TSO and PSO models. The STBAR enforces the ordering constraints on the atomic load and store operations and all the store operations which appear before and after this barrier. The MEMBAR is used both for the memory synchronization purposes and for the enforcement of the ordering constraints on the memory references. The x86 architectures [27] on the other hand uses memory fence (MFENCE), store fence (SFENCE) and load fence (LFENCE) instructions to enforce the ordering constraints on the memory operations. The SFENCE enforces the ordering constraints among the store operations which occur before and after it. Similarly, LFENCE is used for the load operations, while MFENCE enforces the ordering constraints on all the load and store operations that appear prior and later to it.

As shown in Figure 2.5(a), the PSO model further eliminates the ordering constraint in the case of a *write operation on (Y) followed by a write operation on (Z),* while this is not allowed under the TSO model as given in Figure 2.4(a). In contrast to the SC model as shown in Figure 2.3(a), the PSO model allows additional reordering among the *write* operations ($Y \rightarrow Z$) and also in the case of a *write followed by a read* ($W \rightarrow X$) operation. The PSO model enforces the global orders on the shared memory operations as given in Figure 2.5(b). More details on these global orders are given in chapter 3 under the realization scheme of the PSO model in the McNoC systems.

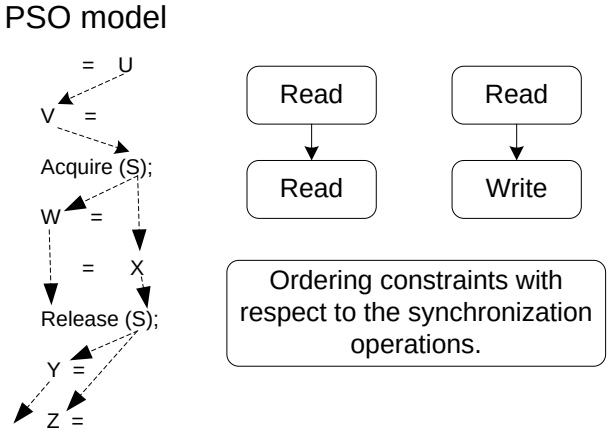



**Figure 2.5.** The PSO model: a) Ordering requirements; b) Global orders.

In the following sub-sections, we consider other relaxed memory consistency models which provide additional reordering and relaxation among the memory operations. These memory consistency models further exploit the system optimizations to reduce the memory access latency by allowing additional pipelining and overlapping among the memory operations. We consider the two well known relaxed memory consistency models (WC and RC) and a new memory model (PRC model) is proposed by extending the idea of RC model.

### 2.5.4 Weak Consistency Model

The Weak Consistency (WC) model (often called *Weak Ordering*) is introduced by Dubois et al. [12]. It classifies the memory operations as *synchronization* and *data* operations. The synchronization operations are related to the special synchronization variables (locks, semaphores) maintained in the shared address space. The multiple processors are synchronized over the critical memory references using these synchronization variables. The data (read, write) operations are related to the ordinary shared variables in the system. The WC model enforces the ordering constraints at the synchronization points in the program. The WC model compared to the PSO model further allows reordering and overlapping among the *read* operations and also in the case of a *read followed by a write* operation. According to the WC model, the independent data operations issued by a processor in between the two consecutive synchronization points can be reordered and pipelined with respect to each other. Hence, the memory access latency can be significantly reduced compared to the stricter memory consistency models (e.g., SC, TSO and PSO). The WC model does not allow reordering among the data and synchronization operations in order to ensure the parallel program correctness. The issuance of a synchronization operation must be delayed for the completion of previously issued outstanding data operations and also the issuance of a data operation must be delayed for the successful completion of a previously issued synchronization operation.

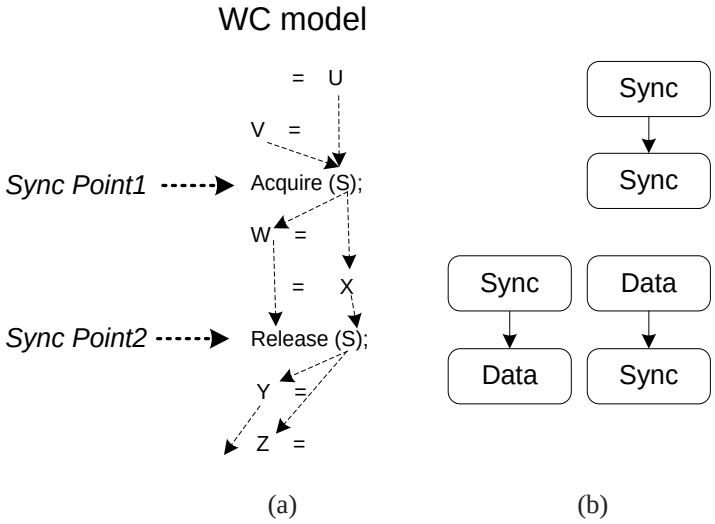



**Figure 2.6.** The WC model: a) Ordering requirements; b) Global orders.

As can be seen in Figure 2.6(a), according to the WC model, the independent data operations issued by a processor in between the two consecutive Synchronization (Sync) points can be reordered and overlapped with respect to each other. This is not allowed under the previously discussed stricter memory consistency models. According to the WC model, the data operations on (*U, V*) are allowed to be reordered with respect to each other, but they are not permitted to be reordered with respect to the data operations on (*W, X*) or (*Y, Z*), because they are not issued in between the two consecutive Sync points. Also, the data operations on (*U, V*) are not allowed to be reordered with respect to the Sync operations and vice versa. The WC model introduces the notion of synchronization in the parallel programs. Therefore, the ordering constraints among the Sync operations are also enforced. For instance, the previously issued Sync operation must be completed before the issuance of a next synchronization operation in the program. These global orders to be enforced on the memory operations under the WC model are shown in Figure 2.6(b).

### 2.5.5 Release Consistency Model

The Release Consistency (RC) model [13][20][21][69] is a refinement of the WC model, which further classifies the synchronization operations as *acquire* and *release* operations as shown in Figure 2.7. An acquire operation delays the future (subsequent) data operations until the lock is obtained. It is due to the fact that the lock must be obtained before entering to the critical section. The issuance of an acquire operation does not wait for the completion of previously issued outstanding data operations of a processor in the system. It is because, an acquire operation does not pass information to any other processor about the completion of previously issued outstanding data operations. This allows further relaxation among the data and acquires operations. Also, the data operations issued by a processor *before* and *after* an acquire operation can be reordered and overlapped under *certain conditions*. However, this is not allowed under the WC model, which does not allow the reordering and relaxation among the data operations that are issued before and after a synchronization point in the program. According to the RC model, a release operation notifies the completion of previously issued outstanding data operations to the other processors in the system. It does not delay the future (subsequent) data operations as they are independent of the release operation. The release operation does not pass any information about the issuance and completion of the *future* data operations. This also enables additional reordering and relaxation among the release and data operations. The classification of synchronization operation under the RC model further permits reordering and relaxation among the memory operations compared to the WC model. The RC model distinguishes the ordering requirements for different types of synchronization operations.

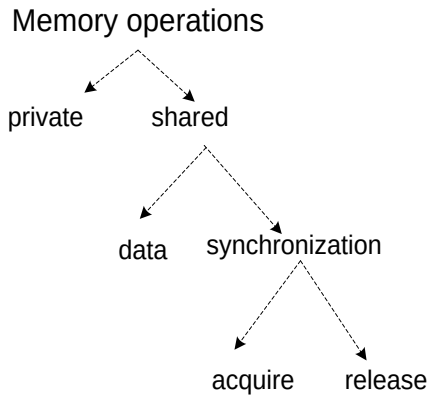



**Figure 2.7.** Classification of memory operations under the RC model.

The ordering constraints to be enforced under the RC model are given in Figure 2.8(a). According to the RC model, the independent data operations on (*U, V*) are allowed to be reordered with respect to each other. The RC model like WC model allows all possible reordering and overlapping among the data operations (e.g., a read followed by a read relaxation, a read followed by a write relaxation, a write followed by a read relaxation, and a write followed by a write relaxation). The data operations on (*U, V*) can also be reordered with respect to the subsequent acquire operation on lock (S) and with the data operations on (*W, X*) in the critical section, while this is not allowed under the WC model. However, the data operations on (*U, V*) are *not* permitted to be reordered with respect to the release operation on lock (*S*). This is because, the RC model enforces the global order in the case of a *data operation followed by a release* operation in order to ensure the parallel program correctness. The data operations (*W, X*) which are issued in the critical section can also be reordered and overlapped with respect to each other, but they are not allowed to be reordered with respect to the acquire and release operations on lock (*S*). This is due to the enforcement of the ordering constraints under the RC model in the cases of an *acquire operation followed by a data* operation and a *data operation followed by a release* operation. The data operations on (*Y, Z*) are allowed to be reordered with respect to each other. They are also allowed to be reordered with the prior outstanding release operation on lock (*S*) and with the prior outstanding data operations on (*W, X*). *These reordering and relaxation are not allowed under the WC model*. However, the data operations on (*Y, Z*) are not permitted to be reordered with respect to the prior acquire operation on lock (S). This is due to the enforcement of the global order under the RC model in the case of an acquire operation followed by a data operation. The data operations on (*U, V, Y* and *Z*) outside the acquire-release operations can be reordered with respect to the data operations on (*W, X*) which are issued inside the critical section, *while this is not permitted under the WC model*. But note that, the data operations on (*W, X*) cannot be moved outside the critical section. Apart from these, the ordering constraints among the acquire and release synchronization operations are also enforced under the RC model. The global order is enforced in the case of an *acquire operation followed by a release* operation and vice versa. For example, a lock must be acquired by a processor

before attempting to release it. Also, a lock must be released by a processor before the next acquire on it. The global orders to be enforced on the memory operations under the RC model are given in Figure 2.8(b).

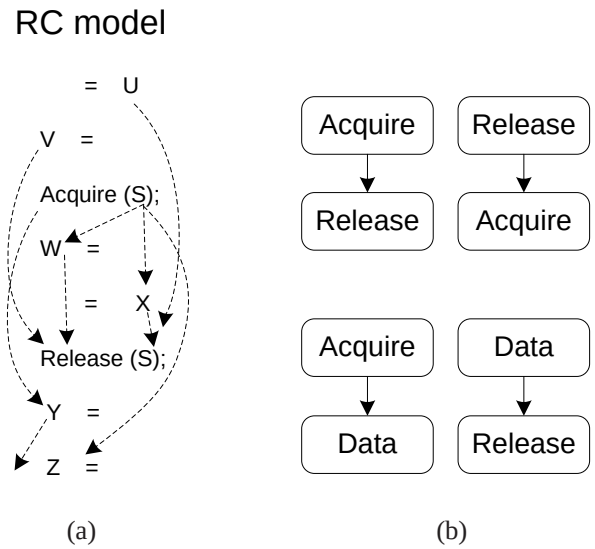



**Figure 2.8.** The RC model: a) Ordering requirements; b) Global orders.

### 2.5.6 Protected Release Consistency Model

The Protected Release Consistency (PRC) model [20] is an extension of the RC model, which further categorizes the data operations as *unprotected* and *protected* operations. The unprotected data operations are not protected under acquire-release operations on a lock. While the protected data operations are protected under acquire-release operations on a lock. According to the PRC model, the issuance of a release operation is *not* delayed till the completion of previously issued *unprotected* data operations. A release operation does not pass information to any other processor in the system about the completion of previously issued unprotected data operations. This allows additional reordering and relaxation among the unprotected data operations with the subsequent release and unprotected data operations under the PRC model compared to the RC model. The issuance of a release operation is only delayed till the completion of previously issued protected data operations. The release operation *only* notifies the completion of previously issued *protected* data operations to any other processor in the system. While in the case of RC model, the release operation unnecessarily notifies the completion of previously issued unprotected data operations which are independent of each other. This additional categorization of the data operations under the PRC model allows more reordering and relaxation among the memory operations compared to the RC model. The main difference with the RC model is that the PRC model distinguishes the ordering requirements for different types of data operations.

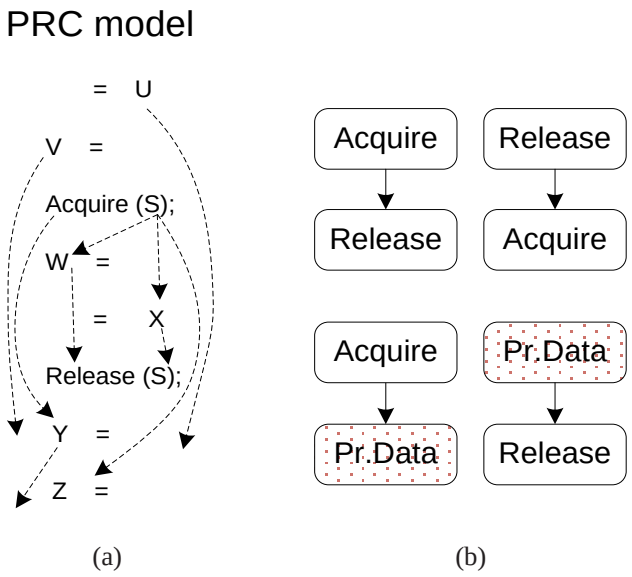



**Figure 2.9.** The PRC model: a) Ordering requirements; b) Global order; Pr. Data: Protected data.

The ordering restrictions on the memory operations under the PRC model are given in Figure 2.9(a). The data operations on (*U, V, Y* and *Z*) are *unprotected* data operations as they are not protected under acquire-release operations on lock (*S*). The data operations on (*W, X*) are *protected* data operations under acquire-release operations on lock (*S*). According to the PRC model, the independent unprotected data operations on (*U, V*) are allowed to be reordered with respect to each other. The PRC model like RC and WC models allows all possible reordering and overlapping among the data (read, write) operations (e.g., a read followed by a read relaxation, a read followed by a write relaxation, a write followed by a read relaxation, and a write followed by a write relaxation). The unprotected data operations on (*U, V*) can also be reordered with respect to the subsequent acquire operation on lock (*S*) and with the protected data operations on (*W, X*) in the critical or protected section. This is similar to the reordering and relaxation allowed under the RC model while this is not allowed under the WC model. The unprotected data operations on (*U, V*) are permitted under the PRC model to be reordered with respect to the release operation on lock (*S*) and with the subsequent unprotected data operations on (*Y, Z*). *However, this is not allowed under the RC model*. The PRC model enforces the global order *only* in the case of a *protected data operation followed by a release operation* while the global order in the case of unprotected data operation followed by release operation is not enforced. The protected data operations (*W, X*) issued in the critical or protected section can be reordered and overlapped with respect to each other, but they are not allowed to be reordered with respect to the acquire and release operations on lock (*S*). This is due to the enforcement of the ordering constraints under the PRC model in the cases of an *acquire operation followed by a protected data operation* and a *protected data operation followed by a release operation*. The unprotected data operations on (*Y, Z*) are allowed to be reordered with respect to each other. They are also allowed to

be reordered with the prior outstanding release operation on lock (*S*) and with the prior outstanding data operations on (*W, X*). This reordering and relaxation is similar to that which is also permitted under the RC model. The unprotected data operations on (*Y, Z*) are also permitted to be reordered and overlapped with respect to the prior outstanding acquire operation on lock (*S*) and with the prior outstanding unprotected data operations on (*U, V*). However, this pipelining and overlapping is not allowed under the RC model. The PRC model does not enforce the global order in the case of an acquire operation followed by unprotected data operation. The unprotected data operations on (*U, V, Y* and *Z*) outside the acquire-release operations can be reordered with respect to each other *before* the issuance of an acquire operation on lock (*S*) and *after* the issuance of a release operation on lock (*S*), *while this is not permitted under the RC model*. The PRC model enforces similar ordering constraints among the acquire and release synchronization operations as discussed under the RC model. The global orders to be enforced on the shared memory operations under the PRC model are shown in Figure 2.9(b). These global orders under six different memory consistency models (Figures 2.3 to 2.6, 2.8 and 2.9) are further discussed in chapter 3 with more details under the realization schemes of these models in the McNoC systems.

### 2.5.7 Comparison of the Memory Consistency Models

As discussed earlier, the stricter memory consistency models enforce more ordering constraints on the memory operations and allow less reordering and relaxation among the memory operations. In contrast, the relaxed memory consistency models allow more reordering and relaxation among the memory operations and take advantage of the system optimizations both in the hardware and software. Figure 2.10 compares six different memory consistency models with respect to each other on basis of ordering constraints they impose on the memory operations. In other words; these memory models are compared on the basis of relaxation they offer among the memory operations. Among these memory models, the SC model offers the least relaxation, while the PRC model offers the highest relaxation in the memory operations. The relaxation offered by the TSO model is a subset of the relaxation which is offered under the PSO model. Similarly, the relaxation offered by the WC model is covered as a part of the relaxation which is offered under the RC model. The PRC model gets the maximum benefits due to the additional reordering and relaxation among the memory operations compared to all these other memory consistency models.

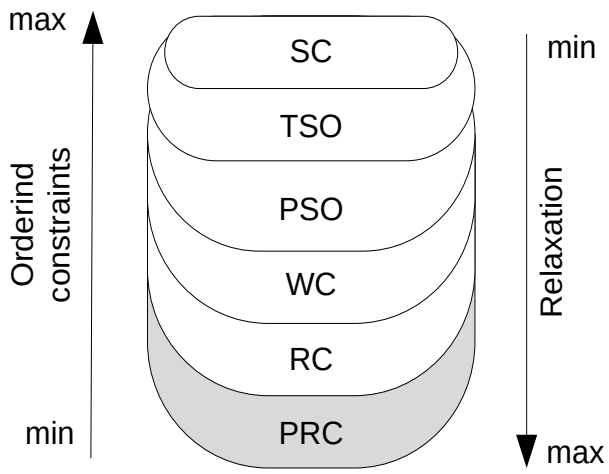



**Figure 2.10.** Comparison of different memory consistency models.

## 2.6 Further Analysis of the Memory Consistency Models

For an extensive analysis and deeper understanding of the ordering constraints under the WC, RC, and PRC models, we consider further cases by using different program segments. We analyze the ordering constraints under these memory consistency models by using different program segments with the non-overlapped, nested and partially overlapped protected sections. All these programs segments consider more than one protected section.

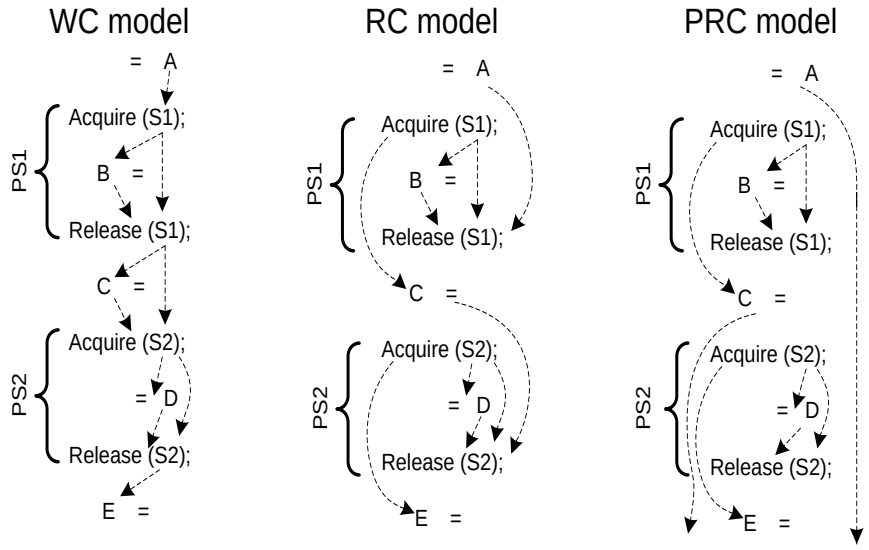



**Figure 2.11.** Non-overlapped protected section, protected data: B, D and unprotected data: A, C, E.

The ordering requirements under the WC, RC, and PRC models are shown in Figure 2.11 for the program 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. These protected sections are represented by curly brackets. The data operations on (*A, C, E*) are not protected under any lock, therefore, they are unprotected data operations. The data operations on (*B, D*) are protected under the two locks (*S1*) and (*S2*), hence they are protected data operations. The WC model does not permit the unprotected data operations on (*A, C, E*) to be reordered and overlapped with respect to the protected data operations (*B, D*) which are issued in the two protected sections. It is because of the fact that they are not issued in between the two consecutive synchronization operations. The WC model also does not allow reordering and relaxation among the data and synchronization operations. The outstanding data operations issued by a processor must be completed before the issuance of a synchronization operation and vice versa. The RC model allows the unprotected data operations on (*A, E*) to be pipelined and overlapped with respect to the protected data operations which are issued inside the protected sections *PS1* and *PS2*, respectively. The unprotected data operations on (*C*) can be reordered with respect to the protected data operations which are issued inside both the protected sections *PS1* and *PS2*. The RC model allows the reordering of unprotected data operation on (*A*) with the subsequent acquire operation on lock (*S1*). Note that, the unprotected data operation on (*A*) is bound to complete before the issuance of a release operation on lock (*S1*). The RC model also permits overlapping among the unprotected data operation on (*E*) with the previously issued outstanding release operation on lock (*S2*). However, this is not allowed under the WC model. Recall that, the unprotected data operation on (*E*) is bound to be issued by a processor on the successful completion of the acquire operation on lock (*S2*) under the RC model. The RC model does not allow the unprotected data operations on (*A, C*) to be reordered with the subsequent release and unprotected data operations. The PRC model allows the unprotected data operations on (*A, C*) to be pipelined with the subsequent release and unprotected data operations which is not permitted under the RC model.

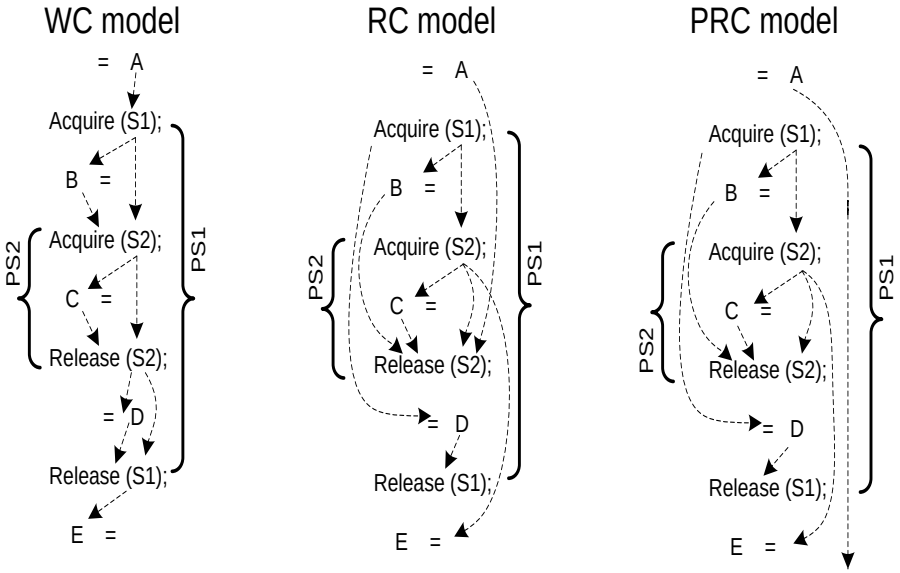



**Figure 2.12.** Nested protected sections, protected data: B, C, D and unprotected data: A, E.

Figure 2.12 demonstrates the ordering constraints under the WC, RC, and PRC models for the program segments using *nested* protected sections. The *PS2* is nested inside the *PS1*. The data operation on (*C*) is now protected data operation under the lock (*S2*), while this is unprotected data operation under the non-overlapped protected case discussed earlier in Figure 2.11. The ordering constraints to be enforced under the WC model are similar to that described under the non-overlapped protected case. This is due to the fact that acquire and release operations are treated as one type of synchronization operation under the WC model. On the other hand, the RC model distinguishes among the different types of synchronization operations (e.g., acquire and release operations) and specifies different ordering requirements for these different kinds of synchronization operations. Therefore, according to the RC model, the unprotected data operations on (*A, E*) can now be reordered and overlapped with respect to the protected data operations issued inside *both* the protected sections *PS1* and *PS2* under *certain conditions*. These are discussed under next paragraph. While under the non-overlapped protected case in Figure 2.11, the unprotected data operations on (*A, E*) are not allowed to be reordered and overlapped with respect to the protected data operations which are issued inside both these protected sections. According to Figure 2.12, under the RC model, the unprotected data operation on (*A*) is not allowed to be reordered and overlapped with the operations issued in that part of the code/program which come *after* the release operation on lock (*S2*). Also, the unprotected data operation on (*E*) is not allowed to be reordered and overlapped with the operations issued in that part of the program which come *before* the acquire operation on lock (*S2*). The PRC model further relaxes the ordering restrictions on the unprotected data operations on (*A*) to be reordered with all the subsequent release and unprotected data operations. However, this is not permitted under the RC model.

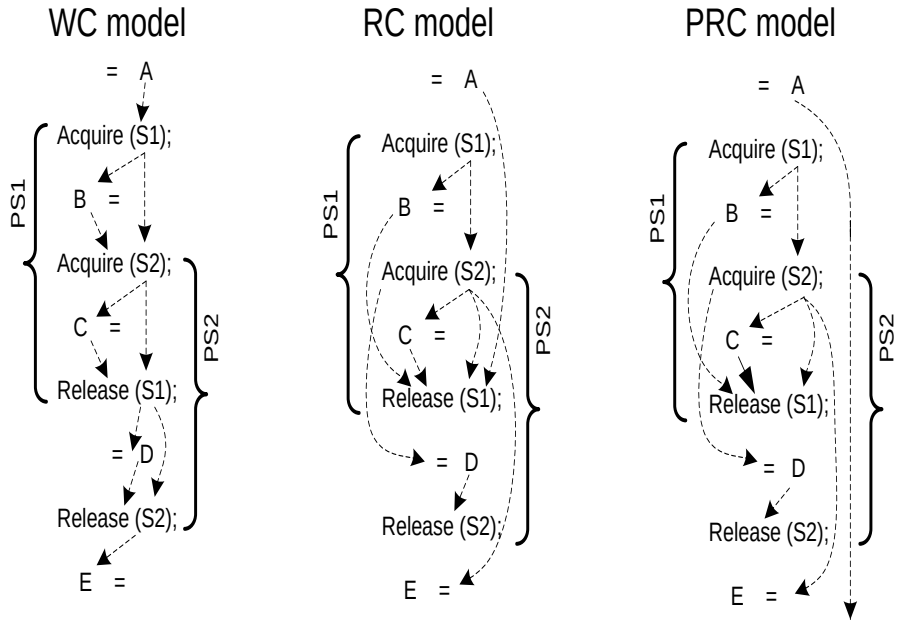



**Figure 2.13.** Partially overlapped protected sections, protected data: B, C, D and unprotected data: A, E.

Figure 2.13 illustrates the ordering requirements under these memory consistency models for the program segments using *partially overlapped* protected sections. A portion of the protected sections *PS1* and *PS2* overlap with respect to each other. The ordering constraints to be enforced on the memory operations under the WC model still remain the same as that in Figure 2.11 (non-overlapped protected section case). The protected data operation on (*D*) becomes now a part of the protected section *PS2*, while under the nested protected case as shown in Figure 2.12, it is a part of the protected section *PS1*. Therefore, both under the RC and PRC models, the protected data operation on (*D*) is now restricted and cannot be reordered and overlapped with the operations which are issued in that part of the program which is before the acquire operation on lock (*S2*). This is because that both the RC and PRC models enforce the global orders in the case of an acquire followed by a protected data operation.

## 2.7 Operations to the Same Memory Location

The operations issued by a processor or different processors to the same memory location must be serialized for the correct behavior of the shared memory systems. We discuss this point from two different perspectives. *First*, we consider the operations issued by different processors to the same location in the memory. The operations which are issued by two different processors to the same memory location result in conflicts (e.g., data races) when at least one of them is a write operation [70]. The *conflicting* or competing accesses to the same location in the memory must be accomplished in a sequential order. Specially, the write operations issued by different processors to the same location in the memory must be serialized in the shared memory systems. This requires the synchronization support among the multiple processors in the system. The WC, RC, and PRC models emphases on the explicit synchronization support among the processors in the system. However, the synchronization support must be provided both at the hardware and software levels in the multi-processor systems to resolve the issues of conflicts and data races. The *second* aspect is to consider the operations which are issued by the same processor to the same location in memory. Further, we assume that only one processor operates on a memory location and no other processor accesses the same memory location at the same time. The *outstanding* data operations issued by a processor to the same location in memory may be reordered and overlapped with respect to each other. As a result, the parallel program correctness could be violated. Thus, the operations issued by a processor to the same location in the memory must be constrained to complete as per program order for the correct behavior of the shared memory systems.

We illustrate by an example, a case of reordering and overlapping of the operations issued by a processor to the same location in the memory which leads to the illegal outcome or execution of the program. We assume that the system supports a relaxed memory consistency model that allows outstanding memory operations in the system. Figure 2.14 demonstrates a piece of code mapped on the processor ($P_1$) in the node ($N_1$) of the network to access the shared data variable (*X*) located in the node ($N_{11}$). The global variable (*X*) is initialized to zero by the processor ($P_1$). Due to the support of a relaxed

memory consistency model like: WC, RC or PRC model, the data (write, read) operations issued by ($P_1$) to the same memory location (*X*) are outstanding in the adaptive network. We assume that the network follows the non-deterministic adaptive routing policy. We only consider the operations issued by the processor ($P_1$) and assume no other processor touches the same memory location (*X*) at the same time. The two outstanding operations issued by the processor ($P_1$) could be reordered in the network. Due to the adaptive routing policy, a write operation on (*X*) may take the longer path (*X-Write*, dashed line arrow) compared to the subsequent read operation path (*X-Read*, solid line arrow) to avoid the congestion or fault in the network. Due to the reordering of the outstanding operations issued to the same location in the memory, the operations issued by the processor ($P_1$) may not be accomplished as per program order. As a result, the inconsistent or old value of (*X*) could be returned in the register (Reg=0) instead of the updated value (Reg=1). This is never expected by the programmer. Hence, the program correctness could be violated.

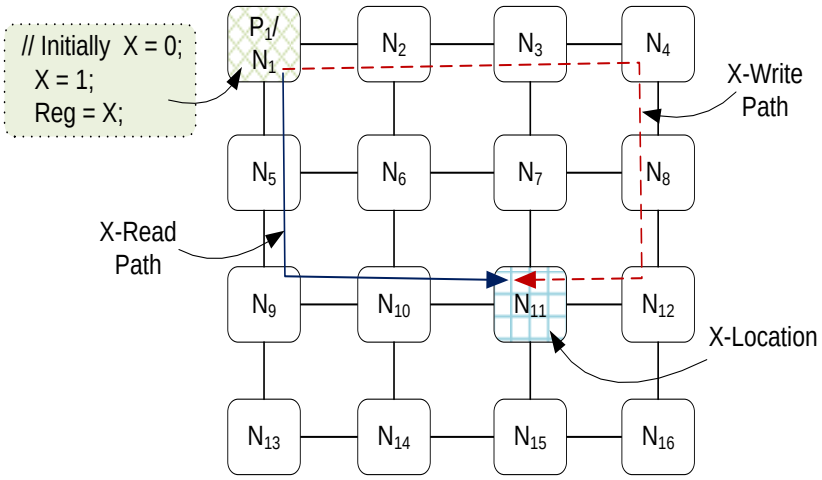



**Figure 2.14.** Reordering of outstanding data operations in the network.

To ensure the parallel program correctness in the adaptive network (NoC)-based systems which support the relaxed memory consistency models, the operations issued to the same location in the memory must be constrained to accomplish as per program order. In the above example, the write operation issued by the processor ($P_1$) must be performed before the issuance of a subsequent read operation to the same memory location (*X*). This would guarantee to obtain the final consistent result (Reg=1). The operations which are issued by a processor to the same location in the memory could be constrained either at the source or destination nodes. Enforcement of the ordering constraints on the memory operations at the destination nodes could be very expensive in terms of cost and performance due to out-of-order reception of the data/memory operations. This is equally applicable for the return/response operations/packets other way round. Alternatively, the issuance of operations by a processor to the same location in the memory could be effectively constrained at the source node at the processor interface. The addresses of the outstanding data operations can be tracked by maintaining a dynamic hardware structure

which can be utilized efficiently at the processor interface. This address tracking mechanism could effectively constrain the operations issued by a processor to the same memory location. The operations issued by a processor to the same location in the memory could be restricted to complete according to the program order for the correctness of the parallel program.

## 2.8 Summarizing the Memory Consistency Models

Tables 2.5 to 2.7 summarize the relaxation offered under six different memory consistency models. A cross "x" denotes that the corresponding relaxation is not allowed, while a dash "-" represents that the corresponding relaxation is allowed. Table 2.5 only considers the relaxation among the read and write operations under the SC, TSO, PSO, WC, RC, and PRC models, while Table 2.6 considers the relaxation among the data (protected, unprotected) operations. The reordering and relaxation among the data and synchronization variables is summarized in Table 2.7. Note that, we consider the synchronization operations under all these memory consistency models in order to make them comparable with respect to each other [67].

**Table 2.5.** Relaxation offered among the read and write operations, R: read, W: write

| Model | R→R | R→W | W→W | W→R | R/W to the same memory location |
|---|---|---|---|---|---|
| SC | x | x | x | x | x |
| TSO | x | x | x | - | x |
| PSO | x | x | - | - | x |
| WC | - | - | - | - | x |
| RC | - | - | - | - | x |
| PRC | - | - | - | - | x |

**Table 2.6.** Relaxation offered under the data operations, PD: protected data, UPD: unprotected data

| Model | UPD→UDP | UPD→PD | PD→PD | PD→UDP |
|---|---|---|---|---|
| SC | x | x | x | x |
| TSO | x | x | x | x |
| PSO | x | x | x | x |
| WC | x | x | x | x |
| RC | x* | - | - | - |
| PRC | - | - | - | - |

**Table 2.7.** Relaxation offered among the data and synchronization operations, Acq: acquire, Rel: release

| Model | Acq→UPD | Acq→PD | UPD→Acq | PD→Acq | UPD→Rel | PD→Rel | Rel→UPD | Rel→PD |
|---|---|---|---|---|---|---|---|---|
| SC | x | x | x | x | x | x | x | x |
| TSO | x | x | x | x | x | x | x | x |
| PSO | x | x | x | x | x | x | x | x |
| WC | x | x | x | x | x | x | x | x |
| RC | x | x | - | -** | x | x | - | -*** |
| PRC | -* | x | - | -** | - | x | - | -*** |

x*). The unprotected data operations are allowed to be reordered with the subsequent unprotected data operations in the program under the RC model. The reordering is only permitted after the issuance of an acquire and before the issuance of a release operation. However, the unprotected data operations under the RC model are not allowed to be reordered before the issuance of an acquire and after the issuance of a release operation which is only allowed under the PRC model.

-*). The subsequent unprotected data operations can be reordered and overlapped with respect to a previously issued acquire operation under the PRC model.

-**). According to both the RC and PRC models, when the acquiring lock is different from the lock which protect the previous protected data operations, then the outstanding data operations which are issued in the previous protected section can be reordered with the subsequent acquire operation.

-***). According to both the RC and PRC models, when the releasing lock is different from the lock which protect the subsequent protected data operations, then the release operation can be reordered with the subsequent protected data operations.

## 2.9 Memory Models at the High level Programming languages

The memory models are also introduced at the high level programming languages. For example, the Java memory model [40] specifies the legal semantics for the multi-threaded Java programs which run correctly on the Java Virtual Machine (JVM) or Java compiler on top of the actual hardware system. The Java memory model ensures the sequentially consistent execution of the Java programs that are free from the data races. It specifies the legal optimizations and transformations for the Java compiler and virtual machine. Java was the first high level programming language where the memory model was specified. Similarly, the memory models are also defined for the other high level programming languages like C, C# and C++. The memory model for the C++ multi-threaded programs is described in [102]. The C++ concurrency memory model is similar to the Java memory model. It also ensures the sequentially consistent execution of the C++ multi-threaded programs which are free from the data races. The Java, C, C# and C++ programming languages use the *volatile* keyword in the declaration of variables which

restricts the compiler optimizations on them. In addition, the C++ also uses *atomic* keyword to assign special properties to the variables. The challenging issue for these memory models which are specified at the high level programming languages is to couple or link these memory models to the relaxed memory models like the RC and PRC supported by the actual hardware in the systems. Figure 2.15 shows the conceptual view of the distinct interfaces at two different levels (e.g., at the programming language level, and hardware interface level) where these memory models could be implemented. The high level language memory model could be specified at the code/program interface level. This memory model must also be integrated or coupled with the hardware memory model which may be implemented underneath at the hardware interface level.

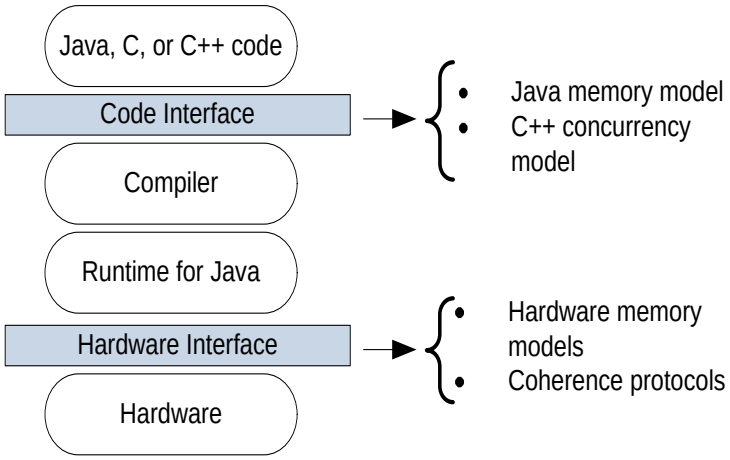



**Figure 2.15.** Interfaces at the programming language and hardware levels.

## 2.10 Influence of the Compiler Optimizations

The compiler can reorder the memory operations to avoid the data dependencies. Within the compiled code, the memory references may be accomplished out of program order. This violates the sequentially consistent execution of a parallel program. As discussed earlier, 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 these variables and the memory operations are accomplished as per program order. Some memory consistency models could provide more space for the compiler to optimize the program. For instance, the relaxed memory models like WC and RC models provide more space for the compiler to optimize the program compared to the stricter memory models. The WC model can allow the compiler to statically reorder the memory operations which are specified in between the two consecutive synchronization points in the program. The RC model compared to the WC model permits the compiler to optimize the program segment in between the two consecutive release points. The PRC model provides even more freedom for the compiler to optimize the program. We refer to this compiler reordering as the static relaxation among the memory operations as it is done statically by the compiler before the execution of a program. The compiler can also allocate the variables (memory operations) to the CPU registers. The variables which are

assigned to the registers are accessed by the processor from the registers instead of memory locations. As a result, the memory access time is reduced due to the reordering of memory operations. The compiler can also eliminate the common sub-expressions by evaluating them to a single value which reduces the number of memory references and the average execution time of a program is decreased. The compiler can also perform some operations on the loops and move the code around to avoid the control dependencies in the code. In brief, the relaxed memory consistency models can utilize the compiler optimizations directly compared to the strict SC model. The PRC and RC models can take maximum advantage of the most common compiler optimizations in the system.

## 2.11 Related Work

### 2.11.1 Memory Consistency in General Multi-processors Systems

A number of memory consistency models are described in the literature [9]-[14]. Adve et al. [10] discussed the memory consistency models from the system optimizations point of view. They described the reordering of shared memory operations due to the system optimizations both in the hardware and software. This work also discusses the impact of the reordering and relaxation among the memory operations on the parallel program correctness. They proposed the counter-based mechanism to realize the WC model; however, they did not discuss the realization of the RC model. The SC model [9] enforces a *total order* on the memory operations. The SC model is an intuitive memory consistency model for the multi-processor systems, but it is very restrictive and cannot take advantage of the performance benefits in the systems. The TSO model [27][73] relaxes the ordering constraints in the case of a *write followed by a read* operation compared to the SC model. The subsequent read operation can bypass the value of the most recent write from the write buffer. The TSO model like the SC model does not provide the relaxation in the cases of a *write is followed by a write* operation, a *read is followed by a read* operation and a *read is followed by a write* operation. The PSO model [73] further provides the relaxation among the memory *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 and SBAR are used in the SPARC architectures [26][73] and MFENCE, SFENCE and LFENCE are used in the x86 architectures [27]. Both the TSO and PSO models exploit the write buffer optimization in the system hardware to reduce the memory access time by hiding the latencies of write operations. The WC model [12] classifies the shared memory operations as *data* and *synchronization* operations. The data (read, write) operations issued by a processor in between the two consecutive synchronization points can be reordered with respect to each other. The RC model [13] is a refinement of the WC model and it further classifies the synchronization operations as *acquire* and *release* operations. The data operations issued by a processor *after* the acquire synchronization operation in the non-critical and critical section can be reordered with respect to each other. Also, the data operations issued by a processor *before* the release synchronization

operation in the non-critical and critical section can be reordered and overlapped with respect to each other.

As discussed earlier, there are some variants of the RC model which claim further reordering and relaxations among the memory operations compared to the RC model. For instance, in the *Eager RC* model [28] the processor delays the propagation of all its modifications to the shared data until the release points in the program. The Eager RC model hides the memory access latency by sending updates belonging to the same destination in a single message at the release points, and reduces the number of messages compared to the RC model. In the *Lazy RC* model [29], propagations of the modifications are further delayed to the acquisition of a lock by another processor. The acquiring processor observes the needed modifications at the time of acquiring a lock. The *Entry RC* [30] further classifies the lock acquire into two different modes (exclusive and non-exclusive). Once a lock is acquired in an exclusive mode, the next non-exclusive acquire of that lock by any other processor is allowed to perform only after the lock owner performs the release operation. The lock can be acquired by more than one processor in the non-exclusive mode when they perform only read operations in their critical sections. Furthermore, the *Scope consistency* model [31] defines a scope which contains all the critical sections guarded by the same lock. Then global ordering is enforced only within a scope. The scope consistency 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 RC model is implemented in the DASH project [32]. The implementation scheme is based on the transaction tracking mechanism by using several counters. The implementation of the RC model is also dependant on the cache coherence protocol. This approach is elegant as it addresses both the memory consistency and cache coherence issues simultaneously; however, there are some confronting issues like: false sharing, massive cache coherence traffic and scalability which need further considerations to be resolved. An orthogonal approach is preferable when these two problems have very different requirements and an implementation scheme for the memory consistency models which is independent of the cache coherence protocols is required. The DASH project implements the *directory*-based cache coherence protocol [16] which maintain the status information of the cache blocks. The directory-based cache coherence protocol is practical compared to the snooping-based coherence protocol [15] in the network-based multi-processor systems. Recent works [33][34][55], on the directory-based coherence protocols mainly focus on the reduction of the directory overheads, energy and power consumption. *Token Coherence* [35] 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. The performance protocol (*Token-B*) is based on the broadcasting of transient requests which is not a scalable approach. The *Location Consistency* [111] is proposed as a new memory model and cache coherence protocol. It does not rely on the assumption of memory coherence for the scalable shared-memory architectures by using the existing coherence protocols. It eliminates the limitations and overheads associated with the existing coherence protocols in the form of directories and buss snooping.

Recently, address translation aware memory consistency models at the physical and virtual address levels (PAMC, VAMC) have been proposed in [36]. The address translation and translation coherence are proposed to enforce a total order on all the memory

operations. They emphasized on the detection of design and runtime faults due to the address translation. In [37], a memory consistency 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 [38] relies on the additional transactional caches and coherence protocols. The transaction size is bounded by the set size of the set associative transactional caches. The software approach [56] has no such restriction and relies on the runtime data structures, but is less efficient. A hybrid approach [39] combines the benefits of both the hardware and software transactional memories. To ensure the consistent behavior of the memory system, aborted transactions due to the conflicts are re-executed. Memory consistency models are also explored at the high-level programming languages [40][102]. For instance, the Java memory consistency model [40] specifies the legal transformations and optimizations for the compiler and virtual machine or hardware.

In [81][82], the authors provide a framework for the Dynamic Verification of Memory Consistency (DVMC) to verify the correct operation and behavior of the memory consistency model. Gharachorloo et al. [83] proposed two general techniques (pre-fetching, speculation) to enhance the performance of the systems which support the memory consistency models. The data pre-fetching could be performed in the time slots when a processor is stalled due to the constraint imposed by the memory consistency models. Speculative execution could allow a processor to proceed when it is stalled for the enforcement of the ordering constraints on the memory operations. The SC model has been implemented aggressively in the recent works [84]-[87]. In [84], InvisiFence approach is proposed for implementing the ordering model that is based on the post-retirement speculation. It avoids some concerns related to the storage requirements in the earlier proposals. In [85], Bulk enforcement of the SC (Bulk-SC) is proposed as a novel way of providing the sequentially consistent execution which can be easily implemented. The instructions are grouped into chunks dynamically which execute atomically in isolation. It enhances the performance of the SC model and makes it comparable to the RC model. The Transactional memory Coherence and Consistency (TCC) is proposed in [86]. The TCC model eliminates the need for the synchronization by using atomic transactions as a unit of coherence and consistency object. In [87], two mechanisms are proposed to enable store-wait–free implementation of any memory consistency model. A scalable store buffer is proposed to eliminate the stalls related to the capacity of store buffer. The atomic sequence ordering is proposed to eliminate the stalls related to the ordering. Gniady et al. [88] introduced SC++ model as an enhanced SC model. The SC++ model requires the proper support of speculation for the comparable performance with the RC model. Ranganathan et al. [89] proposed some mechanisms for the performance improvements of the memory consistency models. They used the non-binding pre-fetching technique to observe the performance improvement in some memory consistency models (e.g., Processor Consistency, RC). The speculative retirement technique is introduced to alleviate some ordering constraints under the SC model to enhance its performance.

Adve et al. [90] used the 3-P criteria (performance, portability, and programmability) for the assessment of memory consistency models and compared the memory consistency

models on the basis of these criteria. A unified framework was established for the reasoning of memory models using 3-P criteria. The SCNF (Sequential Consistency Normal Form) model is specified as the programmer-centric model. It is used as a contract between the programmer and system, where the programmer provides information about the program and the system has to ensure the sequentially consistent execution along with the high performance. The usefulness of the SCNF mechanism is demonstrated by using it on the earlier hardware-centric models. The Processor Consistency (PC) is defined by Goodman [91]. According to the PC model, the write operations issued by the different processors may not be observed by a processor in the order in which they are issued, while the write operations issued by a processor must be observed by the different processors in the same order in which they are issued. Collier et al. [92] also described different memory consistency models that include the IBM-370 model. They defined a memory model for the Shared Memory Multi-Processor (SMMP) and various rules are specified under this model for the coherence and to ensure the execution of operations as per program order. Sindhu et al. [68] presented the formal analysis of the memory consistency models and the TSO model was formally defined for the first time. In [93], a memory model (x86-CC) is proposed for the programs running on the Intel x86 and AMD multi-processor systems. Later on, they improved this model by addressing some issues and introduced the (x86-TSO) model in [94]. They implemented two different memory models (the TSO model, and the axiomatic model), which comply with the features of x86 architectures.

The data-race-free-1 model was proposed in [95] as a unified framework which combines four different consistency models (data-race-free-0, VAX memory model, weak ordering, and release consistency). Under the Data-Race-Free-0 (DRF0) model, all the memory accesses are categorized using properly-labeled-1 model and all the data operations which are involved in the race conditions are distinguished as the synchronization operations. It is based on the assumption that prior knowledge is required about the data races even for the sequentially consistent execution of the parallel program. The data-race-free-1 is based on the assumption that if the proper synchronization support is provided within the system; the SC model would guarantee a high performance by combining the benefits of all these four different memory consistency models. The data-race-free-1 and data-race-free-0 models could guarantee a high performance with the sequentially consistent execution of the specific programs which do not exhibit the data races. In [96], the authors have investigated the techniques and mechanisms for the dynamic detection of data races and for the sequentially consistent execution of the parallel programs under the systems supporting the weaker memory model. The authors argue that the weaker memory models cannot predict about the occurrence of the data races. They have formulated the mechanisms to dynamically detect the data races by preserving the sequential consistency at the data race points in the program. In [97], the authors have presented a class of memory models which is based on the local reordering of the memory operations and on the relaxation in the write atomicity requirements. They have used the fence instructions in a weaker memory model to obtain the SC model. The *diy* tool is used to run the litmus test to identify the errors in the implementation of the memory models on the Power-5 and Power-6 processors. In [98], the formal models were developed for the verification and testing of the parallel program execution on the shared memory multi-processor systems which support the relaxed

memory consistency models. They have used some litmus test programs in order to verify the behavior of parallel programs. The weak ordering was redefined by Adve et al. [99] as a contract between the hardware and software. Within this contract, both the hardware and software obey the constraints related to the ordering of memory operation to get the final execution which is sequentially consistent for the programmer. The re-definition of the weak ordering addresses the issues of data races and coherence which were not focused in the earlier definition. The ARM architectures [100][101] also provide the ordering model for the memory operations. The ordering restrictions are specified based on the attributes of the memory operations. Two explicit memory operations of a processor can be reordered with respect to each other as long as the data dependencies are respected. A memory model for the C++ programming language is described in [102]. This follows the same approach as that of the Java memory model [40]. The C++ concurrency memory model is described for the C or C++ multi-threaded programs.

In [103], the authors have focused on the store and load queues which comprise of the Content-Addressable Memories (CAM) to resolve the dependencies among the memory operations. The working mechanism of the CAM is based on the address matching of the memory transactions. The addresses of load and store operations which are issued by the processor are searched in the CAM at the issuance time. The CAM is used to enforce the ordering constraints on the memory operations at the issuance time. The size of the CAM can be reduced by segmentation, caching, and bloom filtering. The value-based memory ordering allows the load operations to execute again in the program order. They proposed a scalable implementation of the load queue in the FIFO access fashion. In [104], the author has argued that the multi-processor systems should support simple memory consistency models like the SC model, because the relaxed memory consistency models do not justify the performance benefits due to their increasing complexities and difficulties. The IBM latest research [105][106] has investigated two synchronization techniques (locks, queues) in the many-core systems by using the hardware threads. The queues and locks are typically applied in the streaming and non-streaming applications, respectively. The analysis of these two different synchronization mechanisms is studied from several aspects (e.g., implementation, interaction with the operating system, etc.). The analysis is conducted both on the IBM PowerEN$_{TM}$ and Intel X86$_{TM}$ systems. In [107], the authors have developed a mechanism to compare different memory consistency models. They have developed a prototype of a tool for testing the operational and axiomatic specifications of different memory consistency models. Two cases were studied (identification of errors in the TSO specifications, some other variants of the existing memory models are developed by using this tool). The flaws in the Java memory model are identified in [108]. The author argues that ordering constraints imposed under the Java memory model cancels the compiler optimizations and are very hard to implement on the hardware system. It is also reported that the system performance is significantly affected (degraded) by the Java memory model.

### 2.11.2 Memory Consistency in NoC-based Multi-core Systems

Some works [41]-[43] have been done on the memory consistency issue in the context of NoC-based multi-core systems. Petrot et al. [41] explored the reordering of synchronization and data operations due to the routing algorithms, diverse paths and

physical location of the targets in the NoC-based shared memory multi-processor SoC architectures. They proposed that the initiator should wait for the response of the first target before sending the request to the second target in order to avoid the interference (reordering) between the synchronization and data operations. However, the proposed mechanism is very restrictive and allows one outstanding transaction of an initiator at a time in the network. A protocol stack for on-chip interconnects is proposed at the different levels of the SoC design [42]. They briefly outlined a mechanism to implement the RC model at the memory-mapped stack. However, they do not discuss the implementation detail of it. The streaming consistency [43] is based on the software cache coherence protocol. It targets the systems that run streaming applications and share the data through circular buffers (in the shared memory) between the multiple producers and consumers. The streaming consistency is different from the RC model where synchronization sections can overtake each other. However, streaming consistency does not allow the implicit synchronization and every write or read operation to the shared memory must take place inside the synchronization section and not outside of it. Also, 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][17]-[25] to realize the memory consistency models which are independent of the coherence protocols in the McNoC systems. In [19], the SC model is realized by stalling the processor on the issuance of an operation till its completion. A processor issues the next operation in the program on the completion of a previously issued operation.

The TSO and PSO models are realized by using the *Write Transaction Counter* (*WTC)* and *Write Address Stack* (*WA-Stack)*-based approaches [21]. The *WTC* in each node of the network keeps track of the outstanding write operations issued by a processor in the system. The *WA-Stack* constrains the shared memory write operations which are issued by a processor to the same location in the memory. The write operations are constrained to accomplish as per program order for the purpose of correctness. The global orders under the TSO and PSO models are enforced by stalling the processor and by using a *WTC* and a *WA-Stack* in each node of the network. The WC model is realized in [17]-[19] by using a *TC*-based approach in the McNoC systems. The *TC* in each node of the network keeps track of the outstanding data (read, write) operations which are issued by a processor in the system. The required global orders under the WC model are enforced by stalling the processor and by using a *TC* in each node of the network. The RC model is realized in [1][18] by using two *TCs*-based approaches in the McNoC systems. The *TC1* and *TC2* are used in each node of the network to keep track of the outstanding shared data operations issued in the non-critical and critical sections, respectively. However, *TC2* is unnecessarily checked at the acquire points to be zero in [18]; which is already checked at the previous release points in the program. The global orders under the RC model are enforced by stalling the processor and by using two *TCs* in each node of the network. In [20], 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 [21], the realization scheme of the RC model [20] is further enhanced to ensure the parallel program correctness by using an additional hardware structure *A-Stack* (*Address Stack*) in each node of the network. The *A-Stack* constrains the shared memory operations which are issued by a processor to the same memory location in order to accomplish as per program

order. The global orders under the PRC and RC models are enforced by stalling the processor and by using a *TC* and an *A-Stack* in each node of the network.

In [50], the complexities of the dynamic routing are categorized and emphasized that consideration should be given to the transactions ordering requirements in the adaptive routing NoC-based systems. The AXI [44] and OCP [45] protocols enforce the *ordering models* by using transactions IDs and thread IDs, respectively. In [44], transactions of the same master with *different* IDs can be reordered with respect to each other, but transactions with the *same* ID are not allowed to be reordered. In [45], *tagged* transactions of the same master using thread IDs are allowed to be reordered with respect to each other, but *non-tagged* transactions are strictly ordered. Likewise, the *A-Stack* and *WA-Stack* are used in the realization schemes of the memory consistency models [21]-[23] to constrain the memory operations issued by a processor with the *same* address, but the memory operations with the *different* addresses are allowed to be reordered.

To sum up, the main focus of our study is on the memory consistency issue in the customized McNoC systems. The memory consistency models are realized in the McNoC systems by using novel approaches which are independent of the cache coherence protocols (chapter 3). In general purpose parallel computing systems, fence instructions are mostly used to implement the memory consistency models. Our implementation of memory consistency models in the McNoC systems has 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. We have presented a comprehensive and better analysis of the ordering constraints under different memory consistency models. The scalability analysis of these memory consistency models is performed by developing and mapping the synthetic and application workloads on the increasing size of the network (chapter 4). The scalability study is performed in the McNoC systems with 1 to 64-cores. Our study underlines some interesting and key factors which affect the performance gain under the relaxed memory consistency models over the stricter memory consistency models.

## 2.12 Summary

This chapter has discussed the background and motivation for the memory consistency problem by using an example. The cache coherence and memory consistency protocols are briefly described. Some well known memory consistency models like: SC, TSO, PSO, WC, RC, and PRC are discussed and the ordering constraints under these memory consistency models are analyzed by using different program segments. A new memory consistency model (PRC model) is proposed as an extension of the RC model which provides further reordering and relaxation among the shared memory operations. The ordering constraints under these memory consistency models are analyzed and compared with respect to each other. A comprehensive analysis of the ordering constraints under the WC, RC, and PRC models is also presented with different program segments using non-overlapped, nested, and partially overlapped protected sections. The ordering constraints on the operations to the same memory location under the relaxed memory consistency models are also described. The ordering constraints and relaxation

offered under these memory consistency models are also summarized. The memory models which are specified at the high level programming languages are also discussed. The influence of compiler optimizations on the memory consistency models is described. The final part of the chapter reviews the state of the art research on the memory consistency issue. The related work on the memory consistency is reviewed comprehensively both in the general purpose multi-processors and customized NoC-based multi-core systems.

# Chapter 3

# Architecture Support for the Memory Consistency Models

This chapter introduces the McNoC systems which support different memory consistency models. We discuss the functionality of each module (component) in these systems. The main focus of this chapter is on the implementation of different memory consistency models in the McNoC systems. The realization schemes of six different memory consistency models: SC, TSO, PSO, WC, RC, and PRC are discussed. These memory consistency models are realized by enforcing the required global orders on the memory operations. These global orders are enforced by *stalling* the processor and by using the *Transaction Counter* and *Address Stack*-based novel approaches.

## 3.1 Synchronization Handler-based McNoC Platform

A homogenous McNoC platform is given in Figure 3.1(a). All the nodes in platform are composed of the same modules and components. A packet-switched on-chip network is used to interconnect all these nodes with each other in the system. Each node is a typical Processor-Memory (PM) node which consists of a processor, transaction controller (TCTRL), Synchronization Handler (SH), Network Interface (NI) and the local memory. The platform uses SH for the memory synchronization, therefore; we refer to it as SH-based McNoC platform. The structure of a PM node is shown in Figure 3.1(b). In the following sub-sections, we briefly illustrate the features and properties of each component in the platform:

### 3.1.1 On-chip Network

The platform uses a packet-switched *Nostrum* on-chip communication network [46]. The Nostrum NoC integrates all the PM nodes in the McNoC platform. It uses two dimensional (2-D) regular mesh topology, as it is easy to analyze. The Nostrum NoC also uses an adaptive routing policy. The deflection routing algorithm is used to avoid the faults and congestions in the network. The packets may follow the alternative paths in the network to reach their destinations. The adaptive or non-deterministic nature of the routing policy means that the two consecutive packets (transactions) which are issued by a processor from the *same* source node to the *same* destination node in the network can be reordered and overlapped with respect to each other. The two packets of the same source

node may follow different paths on the network to reach the same destination. The on-chip network is a buffer-less network which minimizes the hardware implementation overhead, and the area cost of the system is considerably reduced. The buffers are only used at the NIs which stores the packets before injecting them into the network. These buffers also hold the ejected packets from the network in the other direction. The *network protocols* are implemented at the NIs for the communication among the PM nodes over the network. The NI deals with the transactions from the processor via the TCTRL for the remote memory accesses and performs packetization, queuing, arbitration and communication over the network. It handles both the data and synchronization transactions which are issued by a processor within the node. The NI also receives the packets from the network and after de-packetization hands them over to the processor or memory system. The response packets are diverted towards the processor, while the requesting packets of the other nodes are diverted to the local memory or SH for accessing the shared memory locations or locks, respectively.

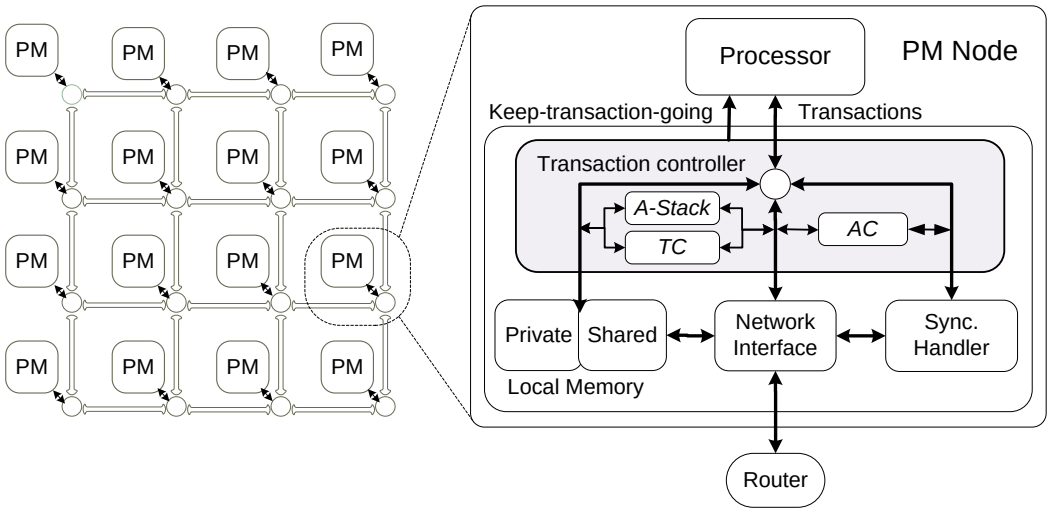



**Figure 3.1.** SH-based platform: a) Homogeneous McNoC; b) PM node. PM: processor memory, A-Stack: address stack, TC: transaction counter, AC: acquire counter.

The *packet format* is shown in Figure 3.2. A packet has a total width of 97-bits. This includes 37-bits long header and 60-bits payload (user data). The header provides the control information about the delivery of the payload in the network. A packet has a total of 7 fields (source relative address, destination relative address, valid packet, hop-count, type of the packet, address and data). Starting from the left side, the first two fields are used for the purpose of routing the packet over the network and facilitate the communication between the source and destination nodes. The valid bit (V) indicates a valid or an invalid packet. As discussed earlier, the network uses deflection routing policy, therefore, it should handle the problem of *livelock*. The livelock may happen when a packet traverses through the network and never reaches to the destination. The hop-count field is used to avoid the problem of livelock. The hop-count is incremented at

every hop in the network which indicates the age of a packet in the network. Thus, older packets are assigned higher priorities in the routing process. The Packet Type field is used to differentiate among the various types of packets in the system. Different types are assigned to both the requesting and response packets. The last two fields indicate the address and data of a memory transaction/operation.


**Figure 3.2.** Packet format.

### 3.1.2 Distributed Shared Memory

The distributed shared memory (DSM) in a single global address space is preferred over the centralized shared memory organization. Within DSM systems, the memories are physically distributed, but logically they are connected in a single shared address space. The DSM system has good potential of scalability and can accommodate the larger problems when the network size is scaled up. The DSM architectures can also help the 3-D (three dimensional) integration process in the chip designs, because the logic die and DRAM require different processing technology. One or more DRAM dies can be stacked on top of the logic die which decreases the physical distance between the logic and DRAM. The expected gains of the 3-D integration process could be in terms of memory bandwidth, latency and power consumption [74]. The 3-D integration process can also reduce the individual die size, improves the chip yield, ease packaging and minimize the power consumption of the main memory [75]. These 3-D memory stacks are critically dependent on the TSV (Through-Silicon-Via) diameters and its overhead. Our platform supports the DSM organization and the shared memory is distributed across the network. The shared part in the local memory is visible to every other node in the system. All the shared parts in the local memories constitute the DSM in a single global address space. Each processor can operate on the DSM in a single global address space. When the number of processors grows in the network, the entire global address space of the DSM is also scaled accordingly. As shown in Figure 3.1, the local memory is connected to the local processor and NI, respectively. For the shared memory access two addressing schemes are used and a Virtual-To-Physical (VTP) address translation is required. The memory (read, write) operations issued by a processor to the local shared memory are accomplished within the node. For the remote shared memory accesses, message passing is carried out to the remote nodes over the network. The requesting messages are routed through the network to their destinations and the remote shared memory is accessed via the NIs in the remote nodes. A response message is sent from the remote node to the local node which completes a remote memory transaction.

### 3.1.3 Memory Synchronization

The platform also uses *distributed* locks scheme, and it is very similar to the DSM scheme. The distributed locks scheme is preferred over the single centralized locks scheme. It can also offer better scalability compared to the centralized locks scheme as the network size is scaled up. When the number of processors grows in the network, the number of distributed locks in the global address space is also increased accordingly. Looking from another perspective, different nodes within a network can synchronize over the locks which are maintained in the *different* nodes (segments) of the network. This can significantly reduce the synchronization overhead and improves the system performance compared to the centralized lock scheme. The SH in our platform controls *k* locks which are maintained in the global address space. These locks are mapped in the shared address space (e.g., memory mapped). Every lock is accessed in a sequential order by the multiple processors in the system. Once a lock is acquired by a processor, it cannot be acquired by any other processor in the system. Other processors have to wait for the release of the lock by the previous processor which has already acquired it. The lock status transition diagram is given in Figure 3.3. A lock can either be in the *locked* or *unlocked* status. In the locked state, the lock is acquired by a processor and it is not available to be acquired by any other processor in the system. In the *unlocked* state, the lock is not acquired by any processor in the system and it is available to be gained by any processor in the system. As shown in Figure 3.1, the synchronization (acquire, release) requests to the SH either arrive from a local processor or from a remote processor over the network. Local acquire and release operations issued by a processor are accomplished within the same node. For the remote lock accesses, message passing is carried out to the remote node over the network. If the requested lock's status is *unlocked*, then it is available and the acquire request changes its status to *locked* and a positive acknowledgement is sent back to the acquiring node. If the requested lock's status is *locked*, then the lock is not available and a negative acknowledgement is sent back to the originating node. The source node sends again the same request until the lock is obtained. If the acquire and release requests arrive at the same time, the lock remains in the *locked* status and the lock ownership/possession is transferred from the releasing node to the acquiring node. A release request changes the lock's status to *unlocked* and the lock is made available to all the processors in the system.

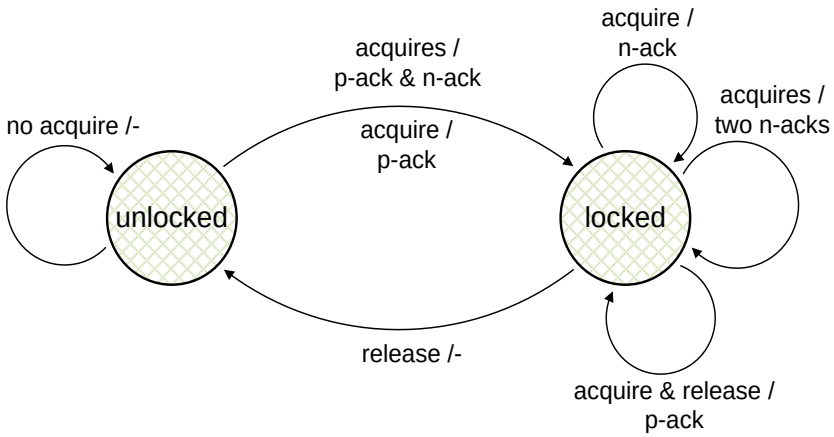



**Figure 3.3.** Lock status transition diagram. p-ack: positive acknowledgement, n-ack: negative acknowledgement.

### 3.1.4 Customized Processor Interface

The processor interface can play an essential role in order to provide the architectural support for the memory consistency models in the shared memory multi-processor systems. The processor interface can facilitate the communication between a processor and the rest of the system. It can also specify the required protocols for such communication at the processor level. Apart from the integration of a processor with the rest of the system, the processor interface can also deal with some other critical issues in the system. For instance, the memory operations which are issued by a processor can be *classified* which help to implement the *relaxed* memory consistency models in the system. The address translation and memory mapping of different operations can be implemented at the processor interface. The flow of transactions from the processor can also be controlled/regulated to avoid the buffer overflows and congestions in the system. More flexible, configurable and sophisticated interfaces can also implement the strategies for the cache coherence protocols and some other complex issues in the systems. There are some *standard* and well-defined on-chip interfaces which are available to integrate the IP-cores with each other in the system. For example, AXI [44], OCP [45], VCI [64], CoreConnect [65] and DTL [66] are the commonly available interfaces. These interfaces enforce the *ordering models* by assigning IDs to the memory transactions which are issued by the same processor. Based on these IDs, the memory transactions are either allowed or not allowed to be reordered with respect to each other. Apart from these standard interfaces, *customized* interfaces can also be developed depending on the needs, situations and requirements. A customized interface should be equipped with the key features and functionalities which may be required under any standard interface such as AXI [44] and OCP [45].

As demonstrated in Figure 3.1, the transaction controller (TCTRL) in our platform is a customized interface which is developed to integrate the processor with the rest of the system. It implements some important and key functions. 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: *Transaction Counter (TC), Acquire Counter (AC),* and *Address Stack (A-Stack*) to realize various memory consistency models in the McNoC systems. The architecture support for these different memory consistency models is discussed in the later part of this chapter with more details. The platform also uses a LEON3 processor [47] in each node of the network. The data cache system is disabled from the base processor for the independent implementation of the memory consistency models. The cache coherence scheme can be implemented on top of this, but the issuance and completion of the data transactions should be redefined to be tracked by the *TC* and *A-Stack* in the TCTRL. However, as discussed earlier, that the scope of this thesis is limited to the memory consistency issue in the customized McNoC systems. Our study targets the hard real time applications, NoC-based customize systems, application specific systems and some other applications which may require the implementation of the

memory consistency protocols that is independent of the cache coherence protocols. The TCTRL is developed specifically for the LEON3 IP-core. The main goal is to highlight the hierarchy and level where the independent memory consistency protocols can be implemented in the McNoC systems. The TCTRL receives different types of transactions (read, write, acquire, release). It handles the transactions issued by a processor with the word granularity. Transactions with half-word and byte granularities could also be tracked by the *TC* and *A-Stack*.

## 3.2 Data Management Engine-based McNoC Platform

We have also used another homogenous McNoC platform for the implementation of various memory consistency models [1][19]. It is given in Figure 3.4(a). This platform also uses the same *Nostrum* interconnection network and DSM organization. The structure of a node is given in Figure 3.4(b). The main difference of this platform from the previous platform is the data management engine and a different type of support for the memory synchronization. In the following sub-sections, both these main differences are briefly discussed.

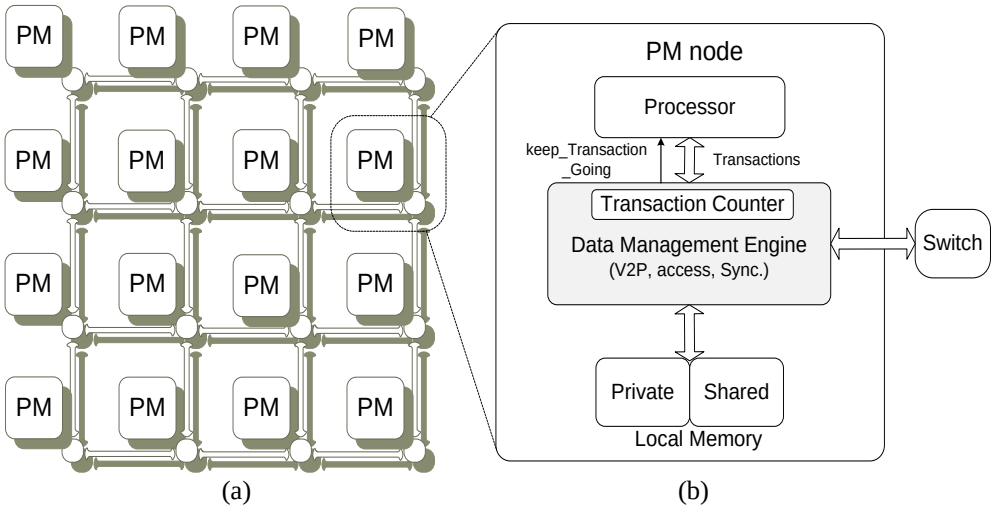



**Figure 3.4.** DME-based platform: a) Homogeneous McNoC; b) PM node. PM: processor memory, V2P: virtual to physical address translation, Sync: synchronization.

### 3.2.1 Data Management Engine

The Data Management Engine (DME) is a hardware accelerator used in each node of the network which offers greater flexibility to implement a variety of read, write and synchronization commands/primitives with different granularities. The DME architecture is shown in Figure 3.5. It is connected to the CPU core, the local memory and the network.

It contains Core Interface Control Unit (CICU), Network Interface Control Unit (NICU), control store, mini-processor-A, mini-processor-B, synchronization supporter and a *Transaction Counter* (*TC*). The CICU and NICU provide the hardware interfaces to the CPU core and network, respectively. Two mini-processors are used as a central processing engine. The micro-program is initially stored in the local memory and it is dynamically uploaded into the control store during the program execution. The synchronization supporter coordinates the two mini-processors to avoid simultaneous accesses to the same synchronization variable (lock) and guarantees the atomic read-modify-write operations over a lock. Both the local memory and control store are dual ported, which are connected to the mini-processor-A and B, respectively. The *TC* is used in the DME hardware to realize the WC and RC models [1][19] in the McNoC systems. The SC model is also realized by stalling the processor on the issuance of an operation till its completion. The memory (read, write) and synchronization (acquire, release) commands are implemented in the DME micro-codes. As illustrated in Figure 3.5, local shared memory and synchronization operations are handled by the mini-processor-A and are completed within the node. Likewise, for the remote shared memory and synchronization operations, messages are sent to the remote nodes by the mini-processor-A over the network. The mini-processor-B in the remote node accesses the shared memory and provides the response messages. Remote shared memory operations are completed either by the data return or write acknowledgments which are issued by the mini-processor-B in the remote node.

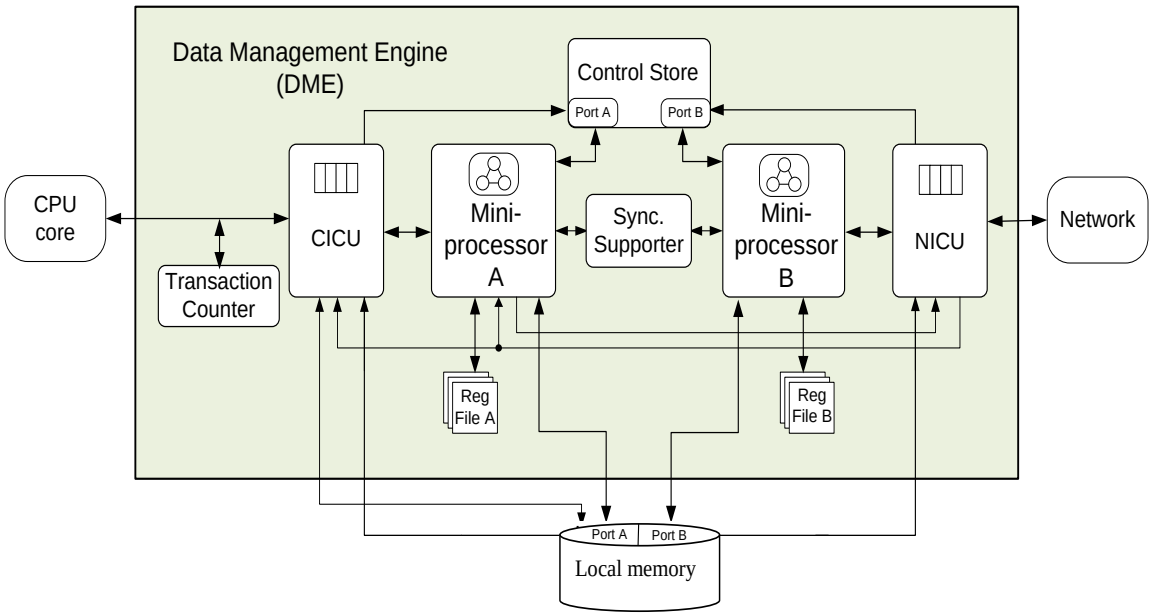



**Figure 3.5.** Structure of the DME.

### 3.2.2 Support for the Memory Synchronization

The DME provides a support for the memory synchronization both at the hardware and software levels. The synchronization supporter is a hardware structure in the DME

architecture which is used in between the two mini-processors. It provides the underlying hardware support for the memory synchronization. As demonstrated in Figure 3.5, the synchronization supporter can receive and respond simultaneously to the synchronization requests from the local CPU-core via the mini-processor-A and the remote synchronization requests via the mini-processor-B. Two special micro-operations (*ll*) and (*sl*) are used in the micro-code together with the synchronization supporter to ensure the atomic read-modify-write operations over the lock accesses. Different synchronization primitives (spin lock, queue locks) like *test-and-set* are implemented in the DME micro-code using these special (*ll)* and (*sl)* micro-operations. Note that, the synchronization supporter does not maintain the locks. It only records the lock addresses for the time in which a lock is being accessed by either mini-processor (A or B) in the memory. The *ll* micro-operation checks the lock address in the synchronization supporter, if the address is not present at the synchronization supporter, then it means that no mini-processor (A or B) is currently accessing the same lock. Either mini-processor (A or B) can then accesses a lock, and the lock address is recorded over the synchronization supporter for the entire period of lock access (acquire or release operation). When the lock address is present on the synchronization supporter, it means that a mini-processor either (A or B) is currently accessing the intended lock at the same time. In such conditions, the mini-processor that observes the lock address on the synchronization supporter is stalled until the other mini-processor completes the acquire or release operation. The *sl* micro-operation in the acquire or release synchronization primitives removes the lock address from the synchronization supporter on the completion of an acquire or release operation. The pending mini-processor can then access the lock in the shared memory. This ensures the exclusive access to a lock by a processor in the system. In contrast to the SH-based platform, the locks are not mapped in the separate reserved segment in the global address space. Any shared memory location can be used as a lock in the DME-based platform. More details on the DME architecture are given in [76].

## 3.3 Architecture Support for the Memory Consistency Models

### 3.3.1 Overview

This section summarizes our main contributions of the research work on the novel realization schemes of different memory consistency models in the McNoC systems. Most of the materials in this section are also reported and presented at the various international forums. We have provided the architecture support for six different memory consistency models (e.g., SC, TSO, PSO, WC, RC, and PRC) in our McNoC systems. The *Transaction Counter (TC)* and *Address Stack (A-Stack)*-based novel approaches are adopted in [1][17]-[25] to realize these various memory consistency models in the McNoC systems. The realization schemes of these memory consistency models are independent of the cache coherence protocols, since we do not consider the data caches in our McNoC systems. The SC model is realized in [19] by stalling the processor on the issuance of an operation till its completion. The TSO and PSO models are realized by using the *Write Transaction Counter* (*WTC)* and *Write Address Stack* (*WA-Stack)*-based approaches [21]. The *WTC* and *WA-Stack* are used to enforce the required global orders under the TSO and PSO models. The WC model is realized in [1][17]-[19] by using a *TC*

in each node of the network. The *TC* keeps track of the outstanding data (read, write) operations which are issued by a processor in each node of the network. In [1][18], two *TCs*-based approaches are used to realize the RC model in the McNoC systems. In [18], the *TC1* and *TC2* are used in each node of the network to keep track of the outstanding data operations which are issued by a processor in the non-critical and critical sections, respectively. However, *TC2* is unnecessarily checked at the acquire points to be zero in [18]; which is already checked at the previous release points in the program. This shortcoming is rectified in the realization scheme of the RC model in [1]. In [20], a single *TC*-based approach is adopted to realize the RC and PRC models in the McNoC systems. The PRC model is proposed as a refinement of the RC model. In [21], the realization scheme of the RC model presented in [20] is further enhanced to ensure the parallel program correctness. An additional hardware structure *A-Stack* is used in each node of the network for this purpose. The *A-Stack* constrains the memory operations which are issued by a processor to the *same* memory location in order to accomplish as per program order.

### 3.3.2 Sequential Consistency Model

The Sequential Consistency (SC) is a *strong ordering* model and is one of the intuitive memory consistency models proposed by the Lamport [9] for the multi-processor systems. The SC model enforces the *program order* on the memory operations issued by an individual processor in the system. It also enforces a *sequential order* among the operations of multiple processors on the shared memory or critical resource in the system. As a result, the enforcement of these global orders (program order, sequential order) on the memory operations under the SC model ensures the *total* order on the memory operations which are issued in the multi-processor system. The SC model was proposed very earlier, therefore, it is based on a simple representation of the shared memory system to the programmer or to the end user. It does not allow reordering and overlapping among the memory operations which are issued by a processor in the systems. Some reordering and pipelining among the memory operations could violate the sequentially consistent execution, however, this may not lead to the inconsistent behavior of the multi-processor system. Thus, the SC model imposes very stringent ordering restrictions on the memory operations issued by a processor in the multi-processor systems. In the following sub-section, we summarize the realization scheme of the SC model which is proposed in [19].

### 3.3.3 Realization Scheme of the SC Model

As discussed earlier, the SC model [9] enforces the *program order* among the operations of an individual processor. The memory operations which are issued by a processor must be completed in the order specified by the program. The SC model enforces the *sequential order* among the operations which are issued by different processors to the critical/shared memory locations. It enforces a *total order* on all the memory operations which are issued in the system. The global orders to be enforced on the memory operations under the SC model are given in the following:

- *Program Order*: among the operations of an individual processor
    - *Read* → *Read*: a read followed by a read operation
    - *Read* → *Write:* a read followed by a write operation
    - *Write* → *Read:* a write followed by a read operation
    - *Write* → *Write:* a write followed by a write operation

- *Sequential Order:* among the processors on the shared (critical) memory references in the system.

We discuss these global orders from the realization perspective of the SC model in the McNoC system. The SC model can be realized by enforcing these global orders on the memory operations. In fact, according to the definition of the SC model, reordering can be allowed, since the results are the same as the strict program order expects. However, it would cause a significant hardware implementation overhead due to the global monitoring of each reordering of the memory operations and constraining these operations. In the following, we discuss an effective realization scheme of the SC model that exactly follows the program order.

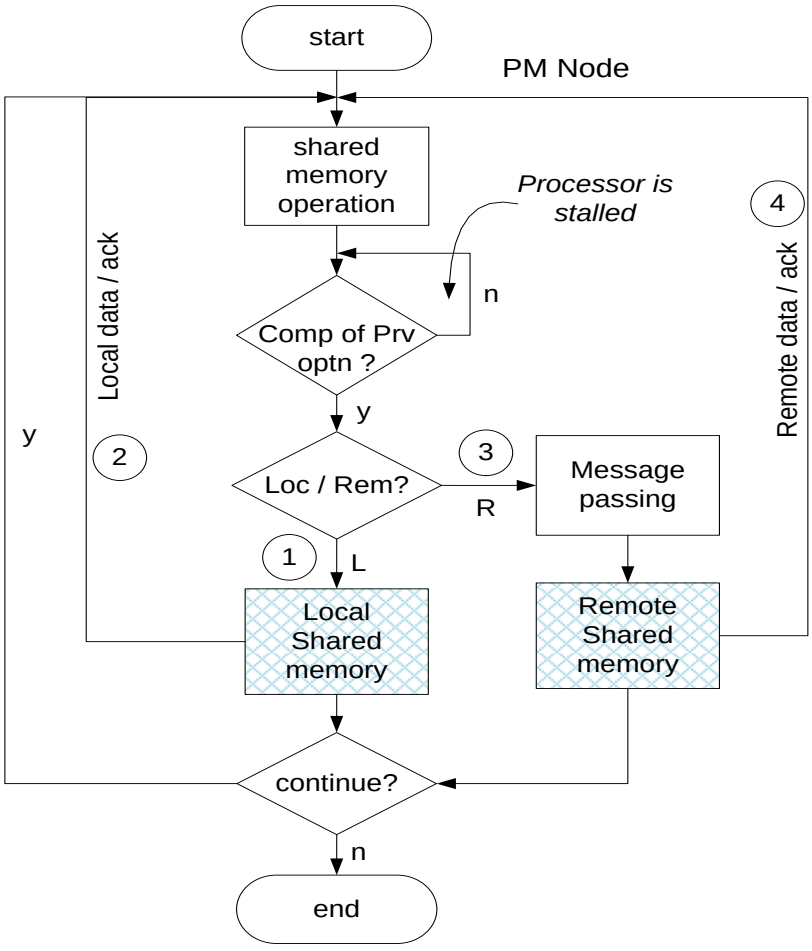



**Figure 3.6.** Implementation scheme of the SC model. PM: processor memory, Comp: completion, Prv: previous, optn: operation, Loc: local, Rem: remote, ack: acknowledgment.

The SC model is realized in [19] by stalling the processor on the issuance of a memory operation till its completion. The processor issues the next operation in the program on the completion of a previously issued operation. The completion of a previously issued operation is indicated by the return data or acknowledgment. This enforces the program order on the memory operations issued by an individual processor in the system, because they are issued and completed in the order specified by the program (program order). This is a straightforward implementation of the program order under the SC model. It would avoid a significant hardware implementation overhead that may otherwise have occurred due to the global monitoring and constraining of memory operations.

The *sequential order* is maintained by sequentially accessing the locks among the multiple processors in the system. It guarantees the sequential accesses to the critical memory locations among the multiple processors in the system. Figure 3.6 demonstrates the realization scheme of the SC model in the McNoC system. The local memory (read or write) operations are issued (1) to the shared memory within the local node. These local memory operations are completed (2) either by return data or write acknowledgements. For the remote memory accesses, message passing (3) is carried out to the remote nodes over the on-chip network. These memory operations are completed by the response messages (4) from the remote nodes. At the issuance time of a memory operation either to the local (1) or remote shared memory (3), the processor is stalled till the completion of a previously issued operation in the program. When the previously issued operation is completed, then the next operation is issued in the program. Overall, the memory operations are issued and completed in the order specified in the program.

### Relaxed Memory Consistency Models

The SC model is a strict memory consistency model and enforces more ordering constraints on the memory operations. It does not allow even those reordering and relaxation among the memory operations which would not lead to the unexpected behavior of the multi-processor systems. The reordering and overlapping among the memory operations may occur due to the system optimizations. These optimizations potentially enhance the system performance at the reasonable cost. The SC model cannot exploit these system optimizations both in the hardware and software due to its strict nature. Thus, several *relaxed or weaker* memory consistency models have been introduced to rectify the shortcomings of the SC model. The relax memory consistency models reduce the ordering constraints on the memory operations compared to the SC model. These memory models allow reordering and relaxation among the memory operations which are issued by a processor in the system. Under the relaxed memory consistency models, the memory operations which are issued by a processor can be executed out of the program order. The memory operations may not complete in the order in which they are issued in the program. However, to ensure the parallel program correctness the relaxed memory models enforce the ordering constraints on the memory operations which are the sub-sets of the ordering constraints that are imposed under the SC model. There are several relaxed memory consistency models which correspond to the different relaxations they offer among the memory operations. We present the realization schemes of some relaxed memory consistency models (e.g., TSO, PSO, WC, RC, and PRC) in the next sub-sections. We

analyze the ordering constraints which are enforced on the memory operations under these memory consistency models from the realization point of view of these memory models in the McNoC systems.

### 3.3.4 Total Store Ordering Model

The Total Store Ordering (TSO) is a relaxed model compared to the SC model which is implemented in the SPARC and x86 architectures [27][68][73]. In contrast to the SC model, the TSO model allows reordering and relaxation in the memory operations in the case of a *write followed by a read* operation. The subsequent read operation can be reordered and overlapped with the outstanding write operation issued earlier in the program. The TSO model exploits the write buffer optimization in the system architecture. A read operation can fetch the data of a previously issued outstanding write operation from the write buffer instead of referring to the memory. Therefore, the memory access latency is reduced under the TSO model compared to the SC model by pipelining the write operation latency with the subsequent read operation latency. The reordering due to the write buffer optimization can sometimes produce inconsistent results under the execution of a parallel program. In chapter 2 (Table 2.3), we have shown by an example of the Dekker's algorithm that the reordering of operations as a result of a write buffer may lead to a situation where more than one processor can enter into the critical section simultaneously. Therefore, we have argued that the conflicting or critical references over the memory must be protected by using proper synchronization support among the multiple processors in the system. The synchronization support must be provided both at the software and hardware levels. Under the TSO model, the ordering constraints on the memory operations are enforced by using the *fence* instructions. These fence instructions are used with the read and write operations which are inserted either by a programmer or by a compiler to control the ordering of memory operations. A fence instruction enforces the program order between the memory operations which are before and after it. The next sub-section reviews the realization scheme of the TSO model in the McNoC system that is proposed in [21].

### 3.3.5 Realization Scheme of the TSO Model

The TSO model allows reordering and relaxation in the case of *a write followed by a read* operation compared to the SC model. Both the TSO and SC models enforce the ordering constraints in the cases of *a read followed by a write operation, a write followed by a write operation* and *a read followed by a read* operation. Also, the ordering constraints with respect to the synchronization operations must also be enforced in order to avoid the inconsistent behavior of the multi-processor systems. The global orders to be enforced on the memory operations under the TSO model are given in the following:

- *Read → Read:* a read followed by a read operation
- *Read → Write:* a read followed by a write operation
- *Write → Write:* a write followed by a write operation
- Ordering constraints with respect to the synchronization operations

We discuss a straightforward realization scheme that exactly follows the semantic of the TSO model. The memory operations are mostly constrained in the hardware at the processor interface instead of using explicit fence instructions among the memory operations. We revisit these global orders under the TSO model from the realization point of view. The TSO model is realized in [21] by enforcing the required global orders on the memory operations. The flow of operations under the realization scheme of the TSO model is illustrated in Figure 3.7. The read (1) and write (3) operations issued to the shared memory are completed by the return data (2) and write acknowledgment (4), respectively. The synchronization operations issued (5) to the Synchronization Handler (SH) are completed by the synchronization acknowledgments (6).

- ***Read → Read:*** The TSO model does not allow reordering and relaxation in the case of *a read followed by a read* operation. To enforce this global order under the TSO model, the processor is *stalled* on the issuance of a read operation (1) to the shared memory till its completion by the data return (2). The issuance of a subsequent *read* operation is delayed till the completion of a previously issued read operation. The processor is *stalled* by issuing an active low *keep-transaction-going* signal by the TCTRL (refer to Figure 3.1).

- ***Read → Write:*** The TSO model does not allow reordering and relaxation in the case of *a read followed by a write* operation. This global order is also enforced under the TSO model by stalling the processor on the issuance of a read operation till its completion. The issuance of a subsequent *write* operation is delayed till the completion of a previously issued read operation.

- ***Write → Write:*** The TSO model also does not allow reordering and relaxation among the memory *write* operations which are issued by a processor. To enforce this global order on the memory write operations, a *Write Transaction Counter* (*WTC)* is used in each node of the network. The *WTC* keeps track of the outstanding write operations which are issued by a processor in the system. The *WTC* is initialized to zero. The *WTC* is incremented by the issuance of a write operation (3). It is decremented by the completion of a write operation (4). The *WTC* is not affected by the read and synchronization operations. It is checked at the issuance of each write operation and the issuance of a write operation is delayed by stalling the processor till the completion of a previously issued outstanding write operation, which is indicated by the zero value of *WTC*. The ordering constraint in the case of *a write followed by a write* operation is enforced by using the *WTC* and stalling the processor.

The TSO model uses *WA-Stack* (*Write Address Stack)* in each node of the network to constrain the outstanding write operations which are issued by a processor to the same memory location. This is necessary to ensure the parallel program correctness.

The outstanding operations which are issued to the *same* memory location under the TSO model are constrained in the case of *a write followed by a read* operation. The *WA-Stack* keeps track of the *addresses* to be accessed by the previously issued outstanding *write* operations. It stores the addresses of outstanding write operations. On the issuance of a write operation (3), the address to be accessed by the write operation is stored on the *WA-Stack*. On the completion of a write operation (4), the address is removed from the *WA-Stack*.

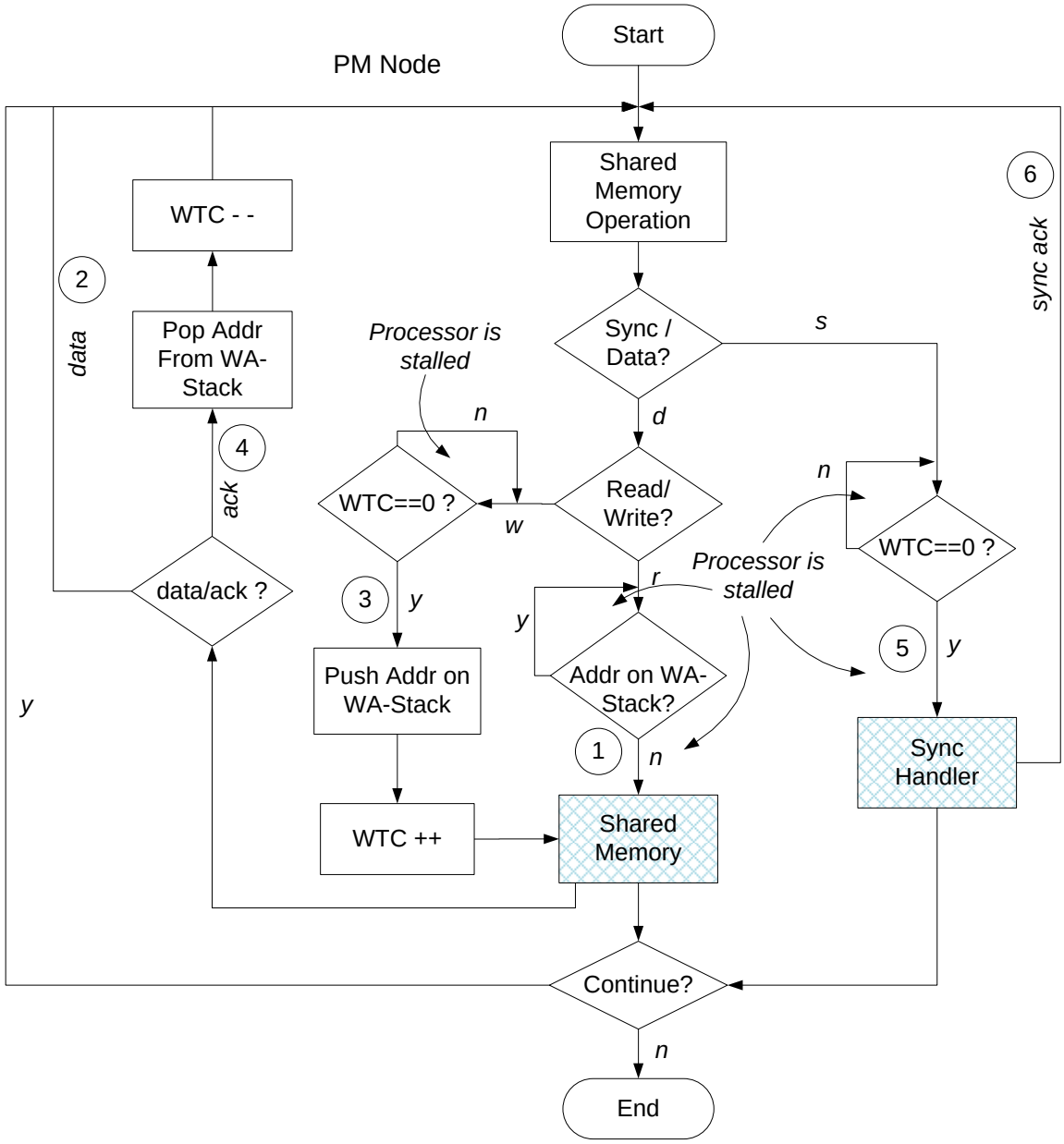



**Figure 3.7.** Realization scheme of the TSO model. PM: processor memory, Sync: synchronization, Addr: address, WTC: write transaction counter, WA-Stack: write address stack.

The ordering constraints with respect to the synchronization operations are also enforced under the TSO model. The shared memory operations (read, write) are completed before the issuance of a synchronization operation. Also, a synchronization operation is completed before the issuance of a shared memory operation. At the issuance of a synchronization operation (5), there is no outstanding read operation, because the processor is stalled on the issuance of a read operation till its completion. However, to ensure the completion of outstanding write operations, *WTC* is checked at the issuance of each synchronization operation. The issuance of a synchronization operation is delayed till the completion of previously issued outstanding write operations (e.g., *WTC=0*). On the issuance of a synchronization operation (5), the subsequent memory operations are delayed by stalling the processor till the successful completion of a synchronization operation (6). Overall, the global orders required under the TSO model are enforced by stalling the processor and by using a *WTC* and a *WA-Stack* hardware structures in each node of the network.

### 3.3.6 Partial Store Ordering Model

The Partial Store Ordering (PSO) model is another relaxed memory consistency model which allows additional reordering and relaxation among the memory *write* operations compared to the TSO model. The PSO model is also proposed in the SPARC architectures [26][73]. It also exploits the write buffer optimizations in the system architecture. It further reduces the memory access time by overlapping the write operations with respect to each other. In contrast to the SC model, the PSO model allows reordering and relaxation among the memory operations in the cases of *a write followed by a read* operation and *a write followed by a write* operation. The ordering constraints under the PSO model are also enforced by using different kind of fence instructions in the commercial systems [25][72]. The following sub-section summarizes the realization scheme of the PSO model in the McNoC system which is proposed in [21].

### 3.3.7 Realization Scheme of the PSO Model

The PSO model allows reordering and relaxation among the memory write operations compared to the TSO model. The PSO model enforces the ordering constraints in the cases of *a read followed by a write operation* and *a read followed by a read* operation. Also, the ordering constraints with respect to the synchronization operations must be enforced in order to avoid the inconsistent behavior of the multi-processor systems. The global orders to be enforced on the memory operations under the PSO model are listed in the following:

- *Read → Read:* a read followed by a read operation
- *Read → Write:* a read followed by a write operation
- Ordering constraints with respect to the synchronization operations

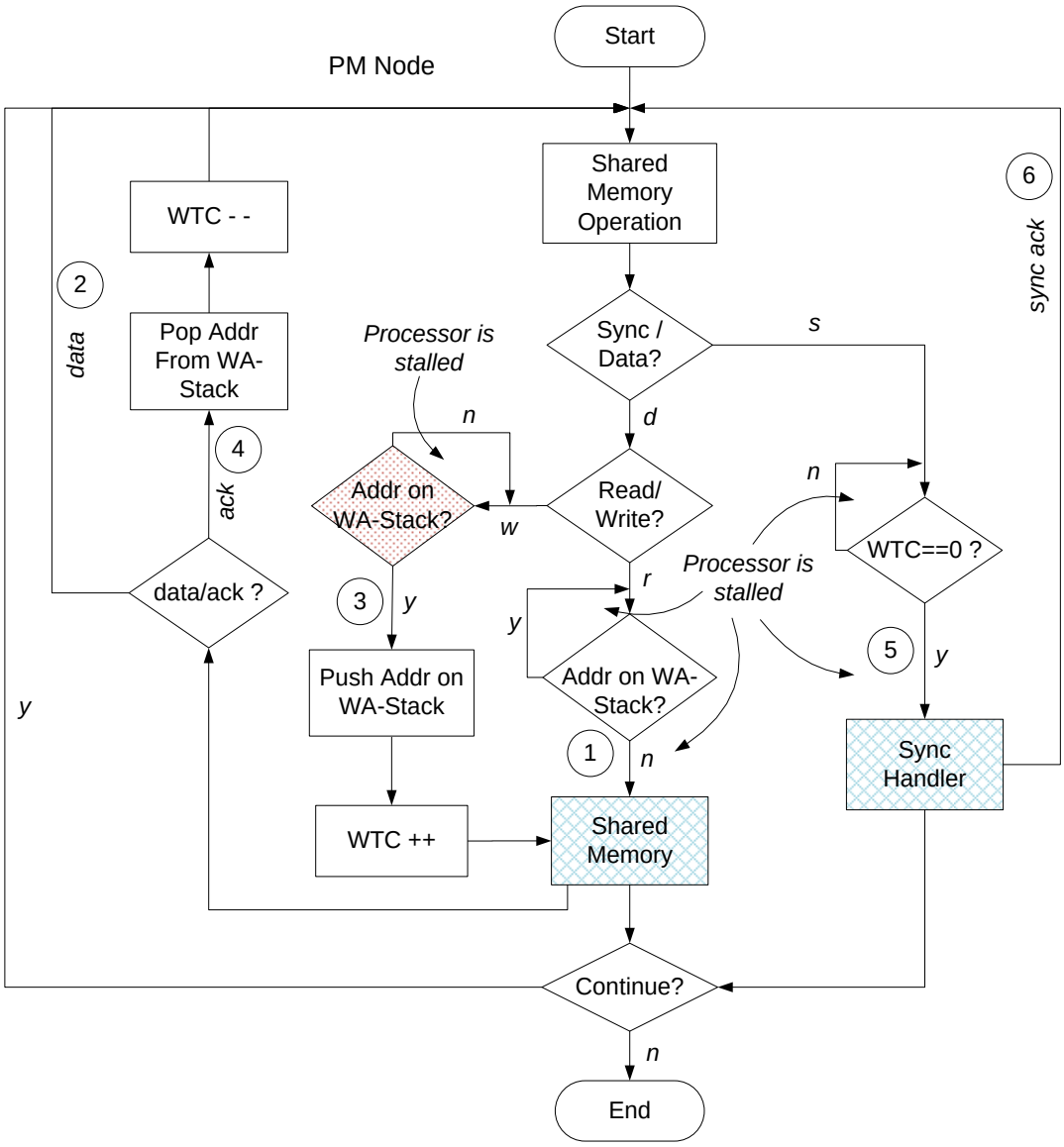



**Figure 3.8.** Realization scheme of the PSO model. PM: processor memory, Sync: synchronization, Addr: address, WTC: write transaction counter, WA-Stack: write address stack.

We reconsider these global orders from the realization standpoint of the PSO model. The PSO model is realized in [21] by enforcing these global orders on the memory operations.

First, we briefly describe the difference among the realization schemes of the PSO and TSO models (Figures 3.8 and 3.7). According to the realization scheme of the PSO model as shown in Figure 3.8, the issuance of a memory write operation (3) from a processor is not delayed when there is an outstanding write operation that is issued by the same processor to a different location in the memory. The issuance of a write operation under

the PSO model does not wait until the completion of a previously issued outstanding write operation, which is indicated by the zero value of the *WTC*. As a result, multiple outstanding memory write operations are allowed in the system, *while this is not allowed under the TSO model (Figure 3. 7)*. The independent memory *write* operations can be reordered and overlapped with respect to each other and the system performance is improved compared to the TSO model. The PSO model also uses *WA-Stack* in order to constrain the operations which are issued by a processor to the same location in the memory for the correct behavior of the system. The *WA-Stack* works in the similar way as described under the TSO model. The *WA-Stack* keeps track of the addresses of outstanding write operations which are issued by a processor in the system. However, there are additional checks on the *WA-Stack* under the PSO model as it allows further outstanding operations compared to the TSO model. The *WA-Stack* is also checked on the issuance of each shared memory write operation (3). If the address to be accessed by the write operation is on the *WA-Stack*, then there is an outstanding write operation issued by a processor to the same memory location. The issuance of a write operation is then delayed until the same address is removed from the *WA-Stack* on the completion of a previously issued write operation to the same memory location. The PSO model enforces the following ordering constraints on the memory operations:

- ***Read → Read:*** The PSO model like TSO model does not allow reordering and relaxation in the case of *a read operation followed by a read* operation. To enforce this global order under the PSO model, the processor is *stalled* on the issuance of a read operation (1) to the shared memory till its completion by the returned data (2). The issuance of a subsequent *read* operation is delayed till the completion of a previously issued read operation.

- ***Read → Write:*** Same as TSO model, the PSO model also does not allow the reordering and relaxation in the case of *a read operation followed by a write* operation. This global order is also enforced under the PSO model by stalling the processor on the issuance of a read operation till its completion. The issuance of a subsequent *write* operation is delayed till the completion of a previously issued read operation.

The ordering constraints with respect to the synchronization operations are also enforced under the PSO model. The shared memory operations are completed before the issuance of a synchronization operation and vice versa. At the issuance time of a synchronization operation (5) there is no outstanding read operation, because the processor is stalled on the issuance of a read operation till its completion. But, in order to ensure the completion of outstanding write operations, *WTC* is checked at the issuance of each synchronization operation. The issuance of a synchronization operation (5) is delayed till the completion of previously issued outstanding write operations (e.g., *WTC=0*). After the issuance of a synchronization operation (5), the subsequent memory operations are delayed by stalling the processor till the successful completion of a synchronization operation (6). On the whole, the global orders required under the PSO model are also enforced by *stalling* the processor and by using the *WTC* and *WA-Stack* hardware structures in each node of the network.

In the following discussion, we consider other relaxed memory consistency models which allow additional reordering and pipelining among the memory operations compared to the SC, TSO and PSO models. These memory consistency models further reduce the memory access latency due to the additional relaxation among the memory operations. The system performance is further improved under these relaxed memory consistency models. We discuss the realization schemes of the two well known relaxed memory consistency models: WC and RC. We also propose the PRC model as an extension of the RC model, which allows further reordering and relaxation in the memory operations.

### 3.3.8 Weak Consistency Model

The Weak Consistency (WC) model (or *Weak Ordering)* is proposed by Dubois et al. [12] which classify the memory operations as *synchronization* and *data* operations. The synchronization operations are related to the special synchronization variables which are maintained in the global address space. The multi-processors in the system synchronize over the critical memory references by using these synchronization variables. The data (read, write) operations are related to the ordinary global variables. The WC model enforces the ordering constraints at the synchronization points in the program. It offers more reordering and relaxation among the memory operations compared to the SC, TSO and PSO models. The WC model compared to the PSO model further allows reordering and overlapping among the *read* operations and also in the case of a *read followed by a write* operation. According to the WC model, independent data operations which are issued by a processor in between the two consecutive synchronization points can be reordered and pipelined with respect to each other. Therefore, the WC model allows all possible reordering and relaxation among the independent data operations (e.g., a read followed by a read relaxation, a read followed by a write relaxation, a write followed by a write relaxation, a write followed by a read relaxation). Hence, the memory access latency can be significantly reduced compared to the stricter memory consistency models like: SC, TSO and PSO. The WC model does not allow reordering among the data and synchronization operations in order to ensure the parallel program correctness. The issuance of a synchronization operation must wait for the completion of previously issued outstanding data operations. Also, the issuance of data operations must be delayed for the successful completion of a previously issued synchronization operation. Under the WC model, the data operations issued on both sides of a synchronization point cannot be reordered and overlapped with respect to each other.

We summarize the different realization schemes of the WC model in the McNoC systems which are proposed in [1] [17]-[19]. The proposed realization scheme of the WC model in [17] uses a *Transaction Counter* (*TC*)-based novel approach. However, the performance of the WC is not compared with any other memory consistency model. The realization schemes of the WC and RC models are described in [18] and their performances are compared with respect to each other. The realization schemes of the WC and SC models are discussed in [19] and their performances are compared in the McNoC systems. In [1], the realization schemes and performance comparison of the SC, WC and RC models are discussed in the DME-based McNoC systems. However, in the earlier realization schemes of the WC model the outstanding operations of a processor to

the same memory location are not constrained. Later on, the realization scheme of the WC model is enhanced in [23] to constrain the data operations which are issued by a processor to the same memory location for the purpose of correctness. The WC model is realized in [23] by using a *TC* and an Address Stack (*A-Stack*)-based novel approach. In the following sub-section, we discuss the latest realization scheme of the WC model in the McNoC systems.

### 3.3.9 Realization Scheme of the WC Model

The WC model enforces the ordering constraints on the data and synchronization operations. All the previously issued outstanding data operations must be completed before the issuance of a synchronization operation. Similarly, all the previously issued outstanding synchronization operations must be completed before the issuance of a data operation. The WC model introduces the notion of synchronization in the parallel programs by explicitly categorizing the memory operations as *data* and *synchronization* operations. Therefore, the ordering constraints among the synchronization operations are also enforced under the WC model. For instance, the previously issued synchronization operation must be completed before the issuance of a next synchronization operation in the program. The global orders to be enforced under the WC model are given as follows:

- *Data → Synchronization*
- *Synchronization → Data*
- *Synchronization → Synchronization*

The WC model is realized by enforcing the above global orders on the memory operations. The realization scheme of the WC model is illustrated in Figure 3.9. A data (read, write) operation (1) to the shared memory is either completed by the return data or write acknowledgment (2). A *synchronization* operation (3) to the SH is completed by the synchronization acknowledgment (4).

- ***Data → Synchronization:*** According to the WC model, outstanding data (read, write) operations issued by a processor must be completed before the issuance of a synchronization operation. To enforce this global order, a *TC* is used in each node of the network. The *TC* keeps track of the outstanding *data* operations which are issued by a processor before a synchronization operation. The *TC* is initialized to zero. The *TC* is incremented by the issuance of a data operation (1). It is decremented by the completion of a data operation (2). The *TC* is not affected by the synchronization operations, i.e., it is neither incremented nor decremented by the issuance and completion of the synchronization operations. The *TC* is checked at the issuance of a synchronization operation (3) and the issuance of a synchronization operation is delayed by stalling the processor till the completion of previously issued outstanding data operations, which is indicated by the zero value of the *TC*.

- ***Synchronization → Data:*** According to the WC model, the previously issued outstanding synchronization operation must be completed before the issuance of a data operation. To enforce this global order, the processor is *stalled* at the issuance of a synchronization operation (3) till its successful completion (4). The subsequent data operations are delayed till the successful completion of previously issued synchronization operation.

- ***Synchronization → Synchronization:*** This global order is enforced by sequentially accessing a lock by the multiple processors in the system which is also discussed in the previous section. A synchronization operation on a lock must be successfully completed by a processor before the issuance of a next synchronization operation on it.

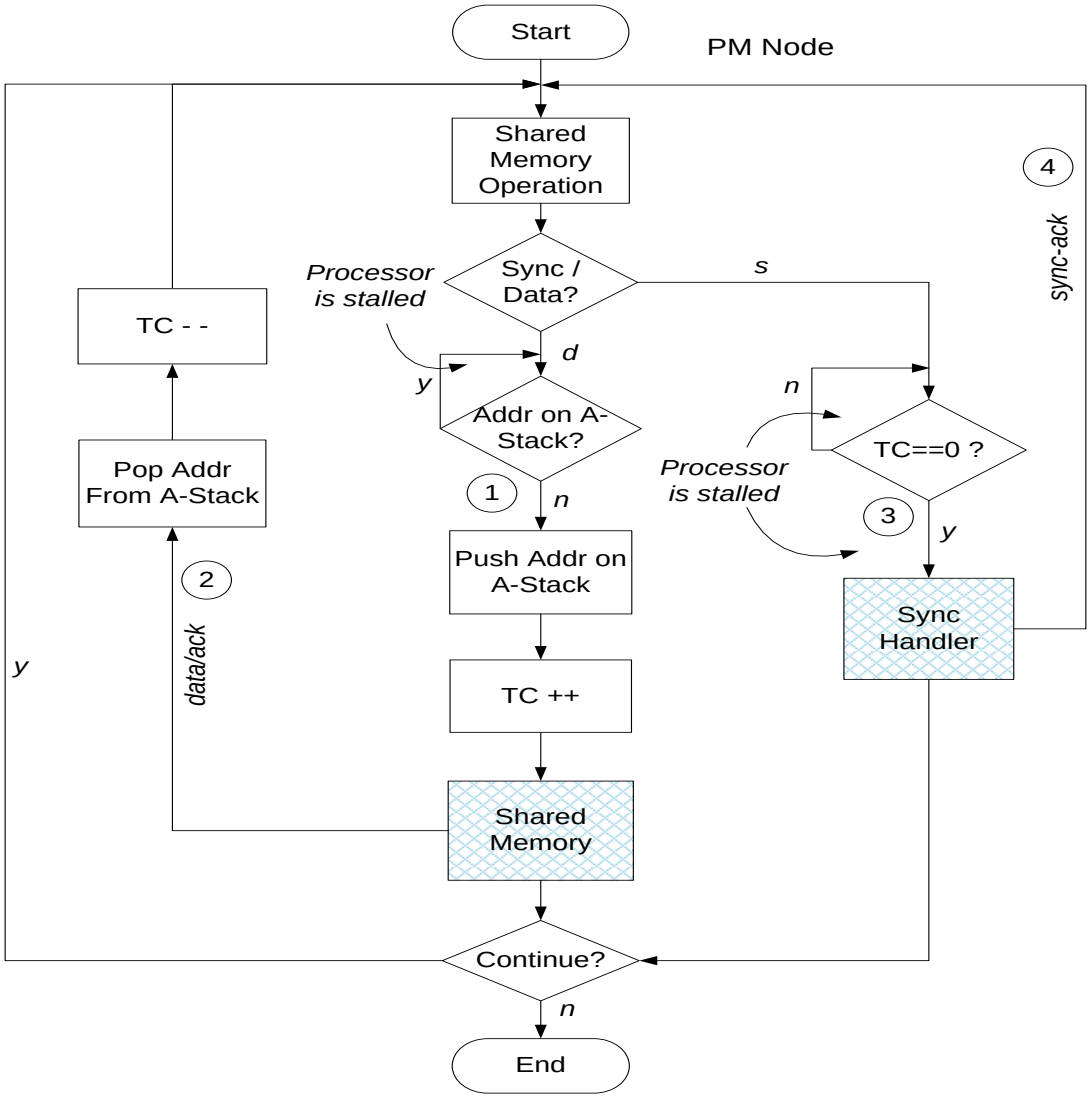



**Figure 3.9.** Realization Scheme of the WC model. PM: processor memory, Sync: synchronization, Addr: address, TC: transaction counter, A-Stack: address stack.

The WC model uses *Address Stack* (*A-Stack*) in each node of the network to constrain the *data* (read, write) operations which are issued by a processor to the *same* location in the memory. The outstanding memory operations issued by a processor to the same memory location must be completed as per *program order*. This is mandatory to ensure the parallel program correctness. The *A-Stack* keeps track of the *addresses* to be accessed in the shared memory by the previously issued outstanding *data* operations. On the issuance of a data operation (1), the address to be accessed in the shared memory by a data operation is stored on the *A-Stack*. On the completion (2) of a data operation, the address is removed from the *A-Stack*. The entries are removed from the *A-Stack* in a random access fashion. The address removing order does not necessarily follow the writing order, since the transactions may complete out-of-order. The *A-Stack* is checked at the issuance of each data operation. If the address is already on the *A-Stack*, it means that there is an outstanding data operation issued by a processor to the same location in the memory. The issuance of a data operation is then delayed until that address is removed from the *A-Stack.* The address is removed from the *A-Stack* on the completion of a previously issued data operation to the same address in the memory.

In summary, the *TC* and *A-Stack* are used in each node of the network to enforce the required global orders under the WC model. The *TC* is incremented with the issuance of the local and remote data operations (1). It is decremented by the completion of previously issued local and remote data operations (2). It is not affected by the issuance (3) and completion (4) of the local and remote synchronization operations. The *TC* is checked at the issuance of a synchronization operation (3). The issuance of a synchronization operation is delayed till *TC*=0, i.e., till the completion of all the previously issued outstanding data operations. The *A-Stack* keeps track of the *addresses* of the outstanding *data* operations. At the issuance time, the address to be accessed by a local and remote data operation (1) is stored on the *A-Stack*. The address is removed from the *A-Stack* on the completion of a local and remote data operation (2). The addresses of the synchronization operations are not tracked (3, 4). The *A-Stack* is checked at the issuance of the data operations (1) and the data operations issued to the same memory location are constrained to accomplish as per program order for the correctness of the parallel program.

### 3.3.10 Release Consistency Model

The Release Consistency (RC) model [13][20][21][69] extends the idea of WC model and further classifies the synchronization operations as *acquire* and *release* operations. The RC model distinguishes the ordering requirements for different types of synchronization operations. An acquire operation delays the subsequent data operations until the lock is gained. It is due to the fact that the lock must be obtained before entering to the critical section. The acquire operation does not wait for the completion of previously issued data operations of a processor in the program. It is because, an acquire operation does not pass information to any other processor about the completion of previously issued data operations in the program. This allows further relaxation among the data operations and the acquire operation. A release operation notifies the completion of previously issued outstanding data operations to other processors in the system. It does not delay the subsequent data operations, because they are independent of the release operation. The release operation does not pass any information about the issuance and

completion of the *future* (subsequent) data operations. This also enables additional reordering and relaxation among the release and subsequent data operations. The classification of synchronization operation under the RC model further enables reordering and relaxation among the data and synchronization operations compared to the WC model.

We recapitulate the different realization schemes of the RC model in the McNoC systems which are proposed in [1][18][20][21]. The realization schemes of the RC model in [1][18] use two *TCs*-based approaches. In [18], the *TC1* and *TC2* are used in each node of the network to keep track of the outstanding data operations which are issued by a processor in the non-critical and critical sections, respectively. The *TC2* is checked at the issuance of each acquire operation and the issuance of an acquire operation is delayed by stalling the processor till the completion of previously issued outstanding data operations in the critical section. The *TC1* and *TC2* both are checked at the issuance of each release operation and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations both in the non-critical and critical sections, which is indicated by a zero value of the *TC1* and *TC2*. This enforces the required global order on the memory operations in the case of a data operation followed by a release operation. The processor is stalled on the issuance of an acquire operation till its successful completion. This enforces the required global orders under the RC model in the cases of an acquire operation followed by a data operation and an acquire operation followed by a release operation. The global order required under the RC model in the case of a release operation followed by an acquire operation is enforced by sequentially accessing the synchronization variables among the multi-processors in the system. In contrast to [18], the realization scheme of the RC model in [1] does not check the *TC2* at the issuance time of an acquire operation. This is due to the fact that the completion of previously issued outstanding data operations is already ensured at the previous release points. The issuance of a release operation checks both the *TC1* and *TC2* to be zero. In [20], a single *TC*-based approach is adapted to realize the RC model in the McNoC systems. Additionally, the PRC model is proposed as an extension of the RC model in the McNoC systems. In [21], the realization scheme of the RC model that is already proposed in [20] is further enhanced by using the hardware structure *A-Stack* in each node of the network to ensure the parallel program correctness. In the following, we discuss the latest realization scheme of the RC model which uses a single *TC* and an *A-Stack*-based approach.

### 3.3.11 Realization Scheme of the RC Model

The RC model enforces the global orders among the data, acquire and release operations. The outstanding data operations issued by a processor must be completed before the issuance of a release operation. Similarly, the lock must be acquired before entering to the critical section. The RC model further classifies the synchronization operations as acquire and release operations, therefore, the ordering constraints among the acquire and release operations are also enforced. The global orders to be enforced under the RC models are given in the following:

- *Data* → *Release*
- *Acquire* → *Data*

- *Acquire → Release*
- *Release → Acquire*

We analyze these global orders from the realization perspective of the RC model in the McNoC systems. The RC model is realized in [21] by enforcing the required global orders on the memory operations. The realization scheme of the RC model is shown in Figure 3.10. A data operation (1) to the shared memory is completed either by the write acknowledgment or data return (2). The *acquire* operation (3) to the SH is completed by the acquire acknowledgment (4). The *release* operation (5) is completed by the release acknowledgment (6).

- **Data → Release:** To enforce this global order under the RC model, a *TC* is used in each node of the network to keep track of the outstanding data operations which are issued before a release operation. Initially, the *TC* is zero. The *TC* is only affected by the data operations. The *TC* is incremented by the issuance (1) of a data operation. It is decremented by the completion (2) of a data operation. The *TC* is not affected by the acquire and release operations. It is checked at the release point and the issuance of a release operation (5) is delayed by stalling the processor until *TC* becomes zero, which indicates the completion of previously issued outstanding data operations. This enforces the global order in the case of data operations followed by a release operation. Note that, the *TC* under the WC model keeps track of the outstanding data operations which are issued before a synchronization (acquire or release) operation. While the *TC* under the RC model keeps track of the outstanding data operations which are issued before a release operation. In other words, the *TC* under the WC model keeps track of the outstanding data operations which are issued in between the two consecutive synchronization operations. While the *TC* under the RC model keeps track of the outstanding data operations issued in between two consecutive release operations.

- **Acquire → Data:** This global order is enforced under the RC model by *stalling* the processor on the issuance of an acquire operation (3) till the successful acquisition of a lock (4). The subsequent data operations in the critical section are delayed for the successful acquisition of a lock. The RC model enforces this global order due to the fact that the lock must be gained by a processor before entering to the critical section.

- **Acquire → Release:** This global order is also enforced under the RC model by *stalling* the processor on the issuance of an acquire operation till the successful acquisition of a lock. The subsequent release operation over the same lock is delayed for the lock acquisition. The RC model enforces this global order due to the fact that the lock must be gained by a processor before trying to release it.

- **Release → Acquire:** This global order is enforced by a sequential order on a lock among the multiple processors in the system. More details are given in the earlier

part of this chapter (previous section). The release operation on a lock must be completed before the next acquire operation on it. The lock must be released by a processor before attempting to acquire it.

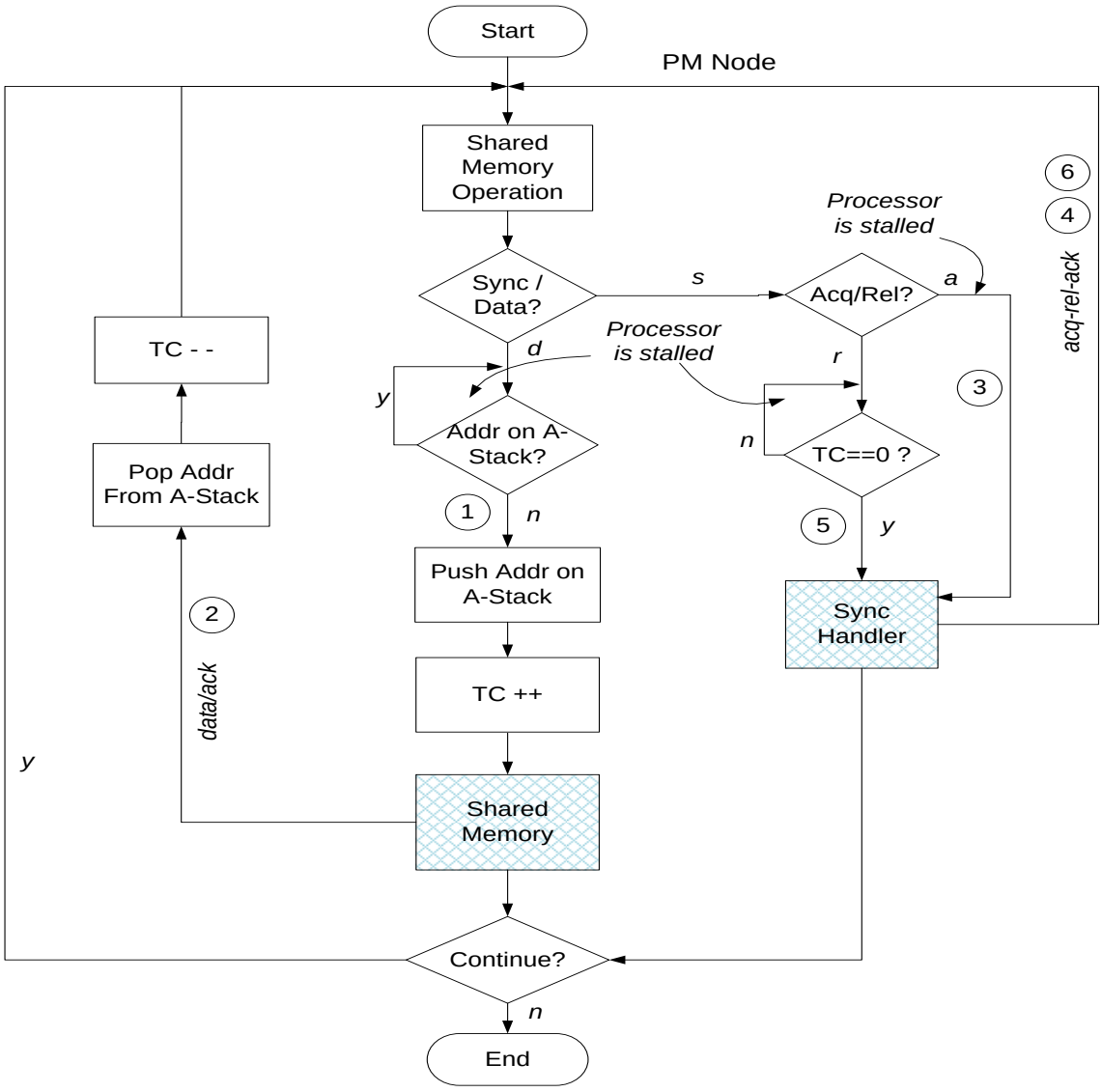



**Figure 3.10.** Realization scheme of the RC model. PM: processor memory, Sync: synchronization, Addr: address, TC: transaction counter, A-Stack: address stack.

The RC model also uses *A-Stack* in each node of the network to constrain the *data* operations which are issued by a processor to the *same* location in the memory. This is mandatory to ensure the parallel program correctness. The *A-Stack* works similar to that described under the WC model.

In brief, according to the realization scheme of the RC model, the *TC* is incremented with the issuance (1) of a local or remote data operation. It is decremented by the completion (2) of a previously issued local or remote data operation. It is not affected by the local and remote acquire/release operations (3, 5). The *TC* is checked at the issuance of a release operation (5). The issuance of a release operation is delayed till *TC*=0. The *A-Stack* keeps track of the addresses of outstanding data operations. The address to be accessed by a local or remote data operation (1) is stored on the *A-Stack* at the issuance time. The address is removed from the *A-Stack* on the completion of a local or remote data operation (2). The addresses of the synchronization operations (3, 5) are not tracked. The *A-Stack* is checked at the issuance of data operations (1) and the data operations to the same location in the memory are constrained to accomplish as per program order. Overall, the required global orders under the RC model are enforced by *stalling* the processor and by using a *TC* and an *A-Stack* in each node of the network.

### 3.3.12 Protected Release Consistency Model

The Protected Release Consistency (PRC) model [20] is a refinement of the RC model. It further classifies the data operations as *unprotected* and *protected* operations. The unprotected data operations are not protected under the acquire-release operations on any lock. The protected data operations are protected under acquire-release operations on a lock. According to the PRC model, the issuance of a release operation is *not* delayed till the completion of previously issued unprotected data operations. This allows additional reordering and relaxation among the unprotected data operations with the subsequent release and unprotected data operations under the PRC, while this is not allowed under the RC model. The issuance of a release operation is delayed under the PRC model till the completion of previously issued *protected* data operations. The release operation *only* notifies the completion of previously issued protected data operations to the other processors in the system. While under the RC model, the release operation unnecessarily notifies the completion of previously issued unprotected data operations. This additional classification of the data operations under the PRC model permits more overlapping and pipelining among the memory operations compared to the RC model. The PRC model distinguishes the ordering requirements for different types of data operations compared to the RC model. In the following sub-section, we summarize the realization scheme of the PRC model in the McNoC system which is proposed in [20].

### 3.3.13 Realization scheme of the PRC Model

The PRC model enforces the global orders among the protected data, acquire and release operations. The outstanding data operations issued by a processor in the protected/ critical section must be completed before the issuance of a release operation. Likewise, the lock must be acquired before entering to the protected/critical section. Similar to the RC model, the PRC model also enforces the ordering constraints among the acquire and release operations. The global orders to be enforced under the PRC models are given in the following:

- *Protected Data → Release*
- *Acquire → Protected Data*
- *Acquire → Release*
- *Release → Acquire*

The PRC model is realized in [20] by enforcing the required global orders on the memory operations. The realization scheme of the PRC model is given in Figure 3.11. A protected data operation (1) to the shared memory is completed either by the write acknowledgment or data return (2). Similarly, an unprotected data operation (3) to the shared memory is also completed by the write acknowledgment or data return (4). The *acquire* operation (5) to the SH is completed by the acquire acknowledgment (6). The *release* operation (7) is completed by the release acknowledgment (8).

- **Protected Data → Release:** This global order is enforced under the PRC model by using a *Transaction Counter for the protected data operations* ($TC_{PD}$). The $TC_{PD}$ is used in each node of the network. It keeps track of the outstanding *protected* data operations which are issued by a processor in the system. The $TC_{PD}$ is initialized to zero. It is only affected by the protected data operations. The $TC_{PD}$ is incremented by the issuance (1) of a protected data operation. It is decremented by the completion (2) of a protected data operation. The $TC_{PD}$ is not affected by the issuance (3) and completion (4) of the *unprotected* data operations. It is also not affected by the issuance and completion of the acquire operations (5, 6) and release operations (7, 8). The $TC_{PD}$ is checked at the release point and the issuance of a release operation (7) is delayed by stalling the processor until $TC_{PD}$ to be zero, i.e., till the completion of previously issued outstanding *protected* data operations. This enforces the required global order under the PRC model in the case of protected data operations followed by a release operation. Note that, in contrast to the RC model, the release operation under the PRC model only notifies the completion of protected data operations. While under the realization scheme of the RC model (Figure 3.10), the release operation notifies the completion of data operations which also include the unprotected data operations.

- **Acquire → Protected Data:** This global order is enforced under the PRC model by stalling the processor upon the issuance of a lock acquire operation (5) until the successful acquisition of the lock. The subsequent protected data operations in the protected (critical) section are delayed for the lock acquisition. The lock must be gained by a processor before entering to the critical/protected section. In contrast to the realization scheme of the RC model, the subsequent unprotected data operations under the PRC model are allowed to be reordered and overlapped with respect to the prior acquire operation. While the RC model restricts the reordering and relaxation of the acquire operation with the subsequent data operations which also includes the unprotected data operations.

- **Acquire → Release:** This global order is enforced by stalling the processor at the issuance of a lock acquire operation until the successful acquisition of a lock. The

subsequent release operation is delayed for the lock acquisition. The lock must be gained by a processor before trying to release it.

- **Release → Acquire:** This global order is enforced by sequentially accessing the locks among the multiple processors in the system. The release operation on a lock must be completed before the next acquire operation on it. The lock must be released before attempting to acquire it by a processor in the system.

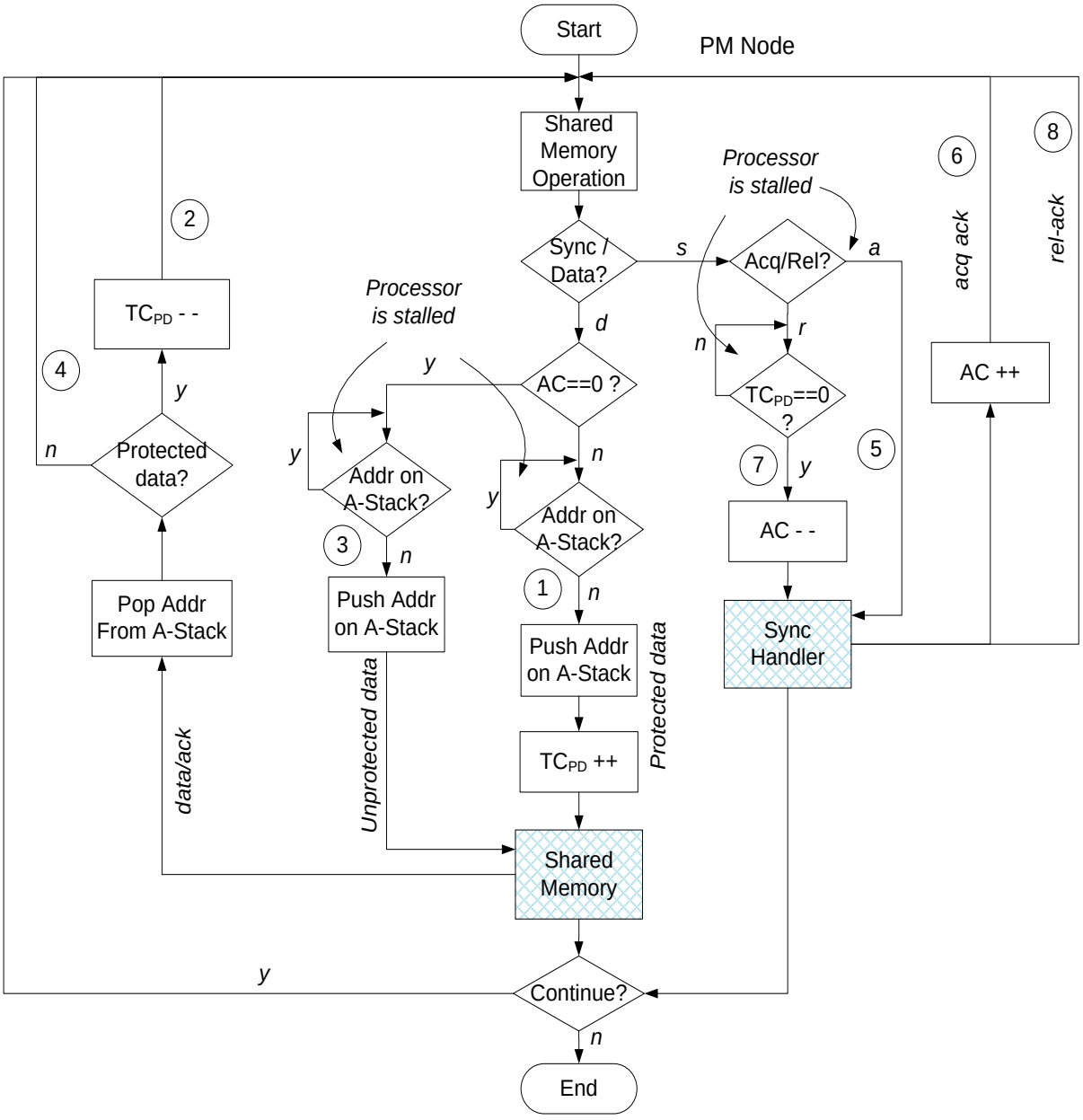



**Figure 3.11.** Realization scheme of the PRC model. PM: processor memory, Sync: synchronization, Addr: address, $TC_{PD}$: transaction counter for protected data operations, A-Stack: address stack, AC: acquire counter.

As demonstrated in Figure 3.11, the PRC model also uses *A-Stack* in each node of the network for the purpose of parallel program correctness. The *A-Stack* is used to constrain the *data* operations which are issued by a processor to the *same* location in the memory. It works similar to that as described under the WC and RC models.

To sum up, according to the PRC model, the $TC_{PD}$ is incremented with the issuance (1) of a local or remote protected data operation. It is decremented by the completion (2) of a previously issued local or remote protected data operation. It is not affected by the local and remote unprotected data operations (3, 4). It is also not affected by the local and remote acquire/release synchronization operations (5, 6, 7, 8). The $TC_{PD}$ is checked at the issuance of a release operation (7). The issuance of a release operation is delayed till $TC_{PD}$=0. The *A-Stack* keeps track of the addresses of the outstanding data operations. At the issuance time, the address to be accessed by a protected (1) or unprotected (3) data operation is stored on the *A-Stack*. The address is removed from the *A-Stack* on the completion of a protected (2) or unprotected (4) data operation. The entries are removed from the *A-Stack* randomly. The address removing order does not necessarily follow the writing order, since the transactions may complete in the order opposite to that in which they are issued in the program. The addresses of acquire (5, 6) and release (7, 8) synchronization operations are not stacked. The *A-Stack* is checked at the issuance of both the protected (1) and unprotected (3) data operations. The required global orders under the PRC model are enforced by stalling the processor and by using a $TC_{PD}$ and an *A-Stack* in each node of the network.

#### 3.3.13.1 Classification of Memory Operations under the PRC Model

The relaxed memory consistency models require the *classification* of memory operations. This classification helps to specify different ordering requirements for the different type of memory operations. This allows more reordering and relaxation among the memory operations under the relaxed memory consistency models. We discuss the categorization of memory operations under the PRC model as shown in Figure 3.12. The memory operations are classified as *private* and *shared* operations based on a boundary address configured in each node of the network. The private memory operations are irrelevant in this context therefore, we do not discuss them. The shared memory operations are categorized into *local* and *remote* operations by virtual to physical address translation. The virtual to physical address translation is carried out at the processor interface (TCTRL). Further, both the local and remote shared memory operations are differentiated as *data* and *synchronization* operations on the basis of different memory mapping in a single global address space. At this point, the classification of memory operations is identical to that under the WC model. The synchronization operations are further distinguished as *acquire* and *release* operations by using different commands/APIs. The classification of memory operations up to this point is similar to that described under the RC model. The PRC model even further distinguishes the memory operations compared to the RC model. It classifies the data operations as *unprotected* and *protected* data operations. An *Acquire Counter* (*AC*) is used in each node of the network for this purpose. As demonstrated in Figure 3.11, under the realization scheme of the PRC model, the *AC* is

incremented by the acquisition of a lock (6), i.e., on the successful completion of a lock acquire operation. It is decremented by releasing a lock (7). The *AC* keeps track of the number of locks which are acquired by a processor in each node of the network. The *AC* is checked at the issuance of data operations (1, 3). When *AC* is zero, it means that the data operation is *unprotected* (3), because either no lock is acquired yet or all the prior acquired locks have been released. When the *AC* is non-zero, the data operation is *protected* (1), because either one or more lock(s) acquire exist already. The *packet type* field is used in the network protocol to differentiate among these different types of memory operations/packets in the system (Figure 3.2).

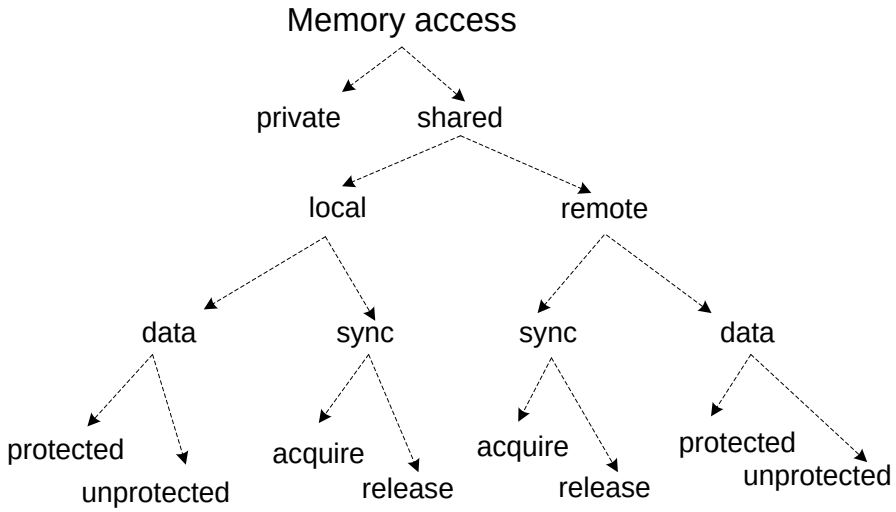



**Figure 3.12.** Classification of memory operations under the PRC model.

## 3.4 Operations to the Same Memory Location

In order to ensure the parallel program correctness, the shared memory operations under the relaxed consistency models to the same memory location must be constrained for their accomplishment as per program order. The TSO model allows outstanding memory operations in the case of a *write followed by a read* operation. Therefore, the realization scheme of the TSO model in [21] uses a *WA-Stack* in each node of the network to constrain the memory operations which are issued to the same memory location. The *WA-Stack* keeps track of the addresses to be accessed by the previously issued outstanding memory write operations. On the issuance of a write operation, the address to be accessed by a write operation is stored on the *WA-Stack*. On the completion of a write operation, the address is removed from the *WA-Stack*. The issuance of a read operation checks the *WA-Stack*. If the address is on the *WA-Stack*, then there is an outstanding write operation that is issued to the same memory location. The issuance of an operation is then delayed until the same address is removed from the *WA-Stack*. The address is removed from the *WA-Stack* on the completion of a previously issued write operation to the same location in the memory. The PSO model in contrast to the TSO model allows further reordering among

the memory operations in the case of a *write followed by a write* operation. Hence, there is an additional check on the *WA-Stack* under the PSO model. The *WA-Stack* is also checked at the issuance of each memory write operation. If the address to be accessed by a write operation is on the *WA-Stack*, then the issuance of a write operation is delayed until the same address is removed from the *WA-Stack*. Similarly, the WC, RC, and PRC models in [21]-[23] use *A-Stack* in each node of the network to constrain the outstanding *data* operations which are issued by a processor to the same location in the memory. The *A-Stack* keeps track of the addresses to be accessed in the memory by the previously issued outstanding data operations. At the issuance time of a data operation, the address to be accessed in the memory by the data operation is stored on the *A-Stack*. On the completion of a data operation, the address is removed from the *A-Stack*. On the issuance of each data operation, the *A-Stack* is checked. If the address is already on the *A-Stack*, then there is an outstanding data operation that is issued to the same memory location. The issuance of a data operation is then delayed until that address is removed from the *A-Stack* on the completion of a previously issued data operation to the same address in the memory.

## 3.5 NoC features and the performance of memory models

The performance of memory consistency models can be affected by the communication patterns. Different traffic patterns allow different overlapping and pipelining among the memory operations under the relaxed memory models like WC, RC, and PRC. Thus, the execution time of different memory models is dependent on the physical distance between the initiator and target nodes in the network. Based on our previous study [21], the execution time is reduced more under the relaxed memory models over SC model for the bit complement traffic pattern. The reduction in execution time is least under the local shared traffic pattern. Irregular random traffic pattern increases the network congestion and latency which increases the overall execution time under the memory consistency models.

Different routing policies also allow for different reordering and overlapping of the memory operations under the relaxed memory models in the network. For instance, the X-Y deterministic routing schemes do not allow reordering among the transactions of the same source to the same destination in the network. However, the operations issued by a processor to the memory locations in different nodes can be reordered with respect to each other. In contrast, the adaptive, non-deterministic nature of the routing policy allows two consecutive packets (transactions) from the same source to the same destination to be reordered on the way. By using adaptive routing in our platform, we incorporate worst reordering situations. Thus, if an implementation of a consistency model works well under an adaptive routing, it also works well under the deterministic routing. But note that, the benefits of relaxed consistency models can be higher under the adaptive routing, because more optimization potential can be exploited by the increasing reordering and relaxation among the memory operations in the network.

The traffic is closed-loop under the implemented memory consistency models in the McNoC systems; hence, the injection rate cannot be arbitrary. It is under the control of the responses. Even with outstanding transactions, there are synchronization points which will slow down the overall injection. However, increasing the injection rate will

increase the number of outstanding transactions under the relaxed memory consistency models like PRC and RC. As a result, the performance gain of the PRC and RC models could be further increased over the SC model. Under the SC model, increasing the injection rate will not have any impact on the performance improvement, because the next operation is issued on the completion of previously issued operation in the program. The injection of a memory operation is controlled by the completion of a previously issued memory operation. Alternatively, the injection of memory operations under the WC model is controlled at the synchronization points in the program, and the outstanding data operations are issued in between the two consecutive synchronization operations. Under the PRC and RC models, the injection of memory operations is constrained at the release points in the program.

The network congestion can also affect the performance of different memory consistency 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 be increased 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 to the SC model, the pipelining and overlapping among the memory operations under relaxed memory models like PRC and RC models could be significantly increased and the memory access latency on average could be decreased.

## 3.6 Caches, pre-fetching and Transactional Memories

Our independent solution of the memory consistency issue does not mean to reject the caches and cache coherence protocols at all. In the cache-based systems, the average memory access time could be reduced. The cache coherence protocols could still be accommodated along with these independent memory consistency models to get the benefits of data caches. As a future work, the directory-based cache coherence protocols can be implemented on top of the memory consistency models in the McNoC systems. 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 the non-blocking caches (which deal with the multiple requests simultaneously) over the blocking caches (which deal one request at a time) [116]. In the systems using blocking caches, the processor is stalled on the cache miss till the completion of the request. On the other hand, the non-blocking or lock up-free caches improve the system performance by servicing/allowing multiple cache miss requests. Several outstanding cache misses could be pipelined with respect to each other and the memory latency is considerably reduced. The relaxed memory models like WC, RC, and PRC models allow outstanding data operations which can get more benefits of the architectures using non-blocking caches compared to the blocking caches. The PRC model allows the maximum relaxation among the memory operations and could provide

more space compared to the stricter models to utilize the systems using non-blocking caches.

The ordering constraints on the memory operations are enforced by stalling the processor. The system efficiency could be increased by doing some useful work in the time slots in which the processor is stalled. The DASH multi-processor system [32] uses the non-binding pre-fetching where the data is brought close to the processor vicinity somewhere in the data cache in advance which is later on read by the processor. The fetched data are ensured to be coherent by using the directory-based coherence protocols. The system performance can be enhanced by utilizing the processors in the time slot when they are waiting/stalled due to the constraints imposed by the memory consistency models.

The memory consistency models and transactional memories are two different approaches to achieve the consistent view of the shared memory systems. The memory consistency models are mainly related to the enforcement of the ordering constraints on the shared memory operations for the expected behavior of the shared memory systems. The transactional memories main focus is to simplify the parallel programming by atomic execution of a set of memory operations which are treated as one transaction. Along with the atomicity, some other requirements like consistency and isolation are also ensured on the transactions. Transactional memory achieves the consistency at the memory transaction level. This is very similar to the cache coherence protocols which achieve the consistency at the cache line/block 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 would be increased and the bandwidth would not be utilized properly. This will also limit the system scalability. Furthermore, as a future work, the transactional memories can be implemented along with the independent memory consistency models in the McNoC systems. The required 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 model.

Each memory consistency model maintains both the write atomicity and write causality. The write atomicity over critical memory references is guaranteed by serial execution of the code segments in the critical/protected sections. This is ensured by the sequential accesses among the multiple processors to the common locks maintained in the system. The writes are casually related as the read operations return the most recently written values to the memory locations.

## 3.7 Summary

In this chapter, we have described the architecture support for six different memory consistency models in the customized McNoC systems. Different McNoC platforms are developed which support these various memory consistency models, e.g., SC, TSO, PSO, WC, RC, and PRC. The composition and functionality of each sub-system in the platforms is discussed. Each platform uses *Nostrum* NoC as a communication backbone with 2-D mesh topology and deflection routing policy. Each platform supports the DSM, memory synchronization and customized processor interface. The processor interfaces are

developed to provide the support for address translation, classification of memory operations, flow control and facilitating the communication between a processor and the rest of the system. These different memory consistency models are realized in the McNoC systems by enforcing the required global orders on the memory operations by *stalling* the processor and by using a *Transaction Counter* and an *Address Stack*-based *novel* approaches. The SC model is implemented by stalling a processor on the issuance of an operation till its completion. The hardware structures *WTC* and *WA-Stack* are used for the enforcement of the ordering constraints on the memory operations under the TSO and PSO models. The *TC* and *A-Stack* are used to enforce the global orders on the memory operations under the WC and RC models. The realization scheme of the PRC model also uses *AC* at the processor interface in addition to the *TC* and *A-Stack*. The *AC* is used to further classify the data operations as *unprotected* and *protected* operations under the PRC model. The global orders related to the synchronization operations are also enforced under these memory consistency models by using the sequential accesses among the multiple processors over the locks maintained in a single global address space. At the end of the chapter, the enforcement of the ordering constraints on the memory operations to the same memory location is described under these different memory consistency models. The impact of the traffic patterns, routing algorithms, and injection rate and network congestion, non-blocking caches, pre-fetching and transactional memories on the performance of the memory models is also discussed.

# Chapter 4

# Scalability Analysis of Memory Models

This chapter focuses on the scalability analysis of memory consistency models which are realized in the McNoC systems. The experimental setups and platform configuration parameters are described. The workloads are developed for the performance evaluation of these different memory consistency models in the McNoC systems. The performance metrics for the scalability analysis of these memory consistency models are defined. The scalability of different memory consistency models is analyzed under these different application workloads, which are mapped on the various sized networks. At the end of the chapter, the scalability analysis of six different memory consistency models is summarized.

## 4.1 Experimental Framework

The experiments are conducted on the McNoC platforms which support different memory consistency models. Each platform is a cycle true simulation platform which is constructed in VHDL. The size of each platform is configurable. We have performed the experiments for the scalability analysis in the network using 1 to 64-nodes (cores). As demonstrated in Figure 4.1 (which is also shown in Figure 3.1, chapter 3), each Processor-Memory (PM) node uses LEON3 processor [47] which is a synthesizable VHDL model of a 32-bit processor compliant with the SPARC V8 architecture. Each platform uses *Nostrum* NoC [46] as the communication backbone, which integrates the IP-cores with each other in the system. The Nostrum NoC is a buffer-less packet-switched network. It has 2-D regular mesh topology. The deflection routing algorithm is used to route the packets to proper destinations. The Network Interfaces (NIs) connect the IP-cores to the Nostrum NoC. The NI uses buffers to hold the packets before forwarding them into the network or IP-cores. The buffering capacity at the NI is 64 packets. There is also a flow control mechanism to avoid the overflows of buffers. The packet formation at the NI uses 7 fields (97 bits). Each platform uses a Distributed Shared Memories (DSM) which are maintained in a single global address space and is accessible to all the processors in the system. The size of shared memory in every node is 16 MB. Each node has a private memory in the local memory for the private accesses of the processor. The platform support memory synchronization both at the hardware and software levels. The *distributed* locks are maintained in the Synchronization Handlers (SHs) in the system. The SH in each node of the network maintains 256 locks in a shared address space. In the DME-based platform as given in Figure 3.4, every shared memory location can be used as a lock. The customized processor interface (Transaction Controller) uses a *Transaction*

*Counter (TC)* and an *Address Stack (A-Stack)* to realize these different *memory consistency models* in the NoC-based platforms. An *Acquire Counter* (*AC*) is used in each node of the network to classify the data operations as unprotected and protected data operations under the platform which support the PRC model. The *Write Transaction Counter (WTC)* and *Write Address Stack (WA-Stack)* are used in the platforms which support the TSO and PSO models. The *TC*, *WTC* and *AC* each are 32 bits. The *A-Stack* and *WA-Stack* can stack up to 64 addresses each with 24 bits. The size of the *A-Stack* and *WA-Stack* is kept small and are efficiently utilized. The addresses are removed from the stack continuously on the completion of memory operations in a pipelined manner. The caches are disabled from the LEON3 processors in the experiments, as they are neutral for the evaluation of memory consistency models. The realization schemes of memory consistency models in the McNoC platforms are independent of the cache coherence protocols. The configuration parameters of a McNoC platform are given in Table 4.1.

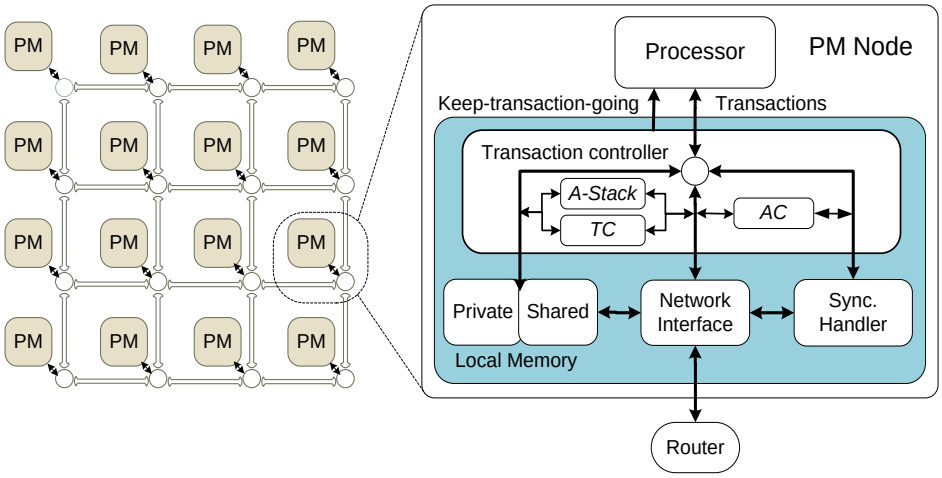



**Figure 4.1.** SH-based platform: a) Homogeneous McNoC; b) PM node. PM: processor memory, A-Stack: address stack, TC: transaction counter, AC: acquire counter.

**Table 4.1.** Configuration parameters of the McNoC platform

| **Sub-system** | **Descriptions/Parameters** |
|---|---|
| *Core processor* | LEON3, synthesizable VHDL model, 32-bit processor 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, WA-Stack/A-Stack stacking capacity 64 virtual addresses each 24 bits |

### 4.1.1 Workloads/Benchmarks

Benchmarks must be chosen so as to accurately represent the system under test for getting some meaningful conclusions from the experiments. The SPLASH-2 benchmark suite [112] has multi-threaded programs which target the high performance computing and graphics domains. The SPEC benchmark suite [113] has the programs which target the engineering and scientific applications. All these programs in SPLASH-2 and SPEC suits use less parallelization models (e.g., parallelization of the workloads to exploit the shared memory multi-processor systems) compared to the PARSEC benchmark suite. The PARSEC benchmark suite [114][115] was publicly released in 2008 which has 13 parallelized workloads chosen from broad range application domains. The PARSEC benchmark suite overcomes the shortcomings of existing SPEC and SPLASH-2 benchmark suites for the performance evaluation of future CMPs. The PARSEC programs are written in C/C++ which targets the desktop and server domains. These programs support pthreads, OpenMP and the Intel Threading Building Blocks (TBB). All these benchmarks suits (SPLASH-2, SPEC and PARSEC) are using the compute-intensive programs/workloads to assess and compare the performance of high-performance computing systems.

On the other hand, the NoC-based systems require the communication centric benchmarks to evaluate the communication aspect of the network. The SPEC, SPLASH-2 and PARSEC benchmark suites have computation intensive programs which are 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 these relaxed memory models. Thus, we have developed both the synthetic and application workloads to evaluate the scalability/performance of different memory consistency models in the McNoC systems. The synthetic workloads are used to test a particular aspect of the system [49]. A set of well defined synthetic workloads could give some meaningful information about the system behavior. However, it cannot replace the real applications. In contrast to the synthetic workloads, the application workloads provide accurate and deeper evaluation and understanding of the system [48]. We have developed the application workloads which are small sized, specific to NoC and embedded architectures and particularly evaluate the communication aspect of the network. These workloads are mapped on the increasing size of the network. The input problem size and the memory instruction density can be varied. Different traffic patterns are generated for the memory operations. The applications are selected from different domains. The data intensive applications (e.g., bit count, angle conversion, matrix multiplication and pattern search) do not use any synchronization primitives and perform

simple scientific computations like bit manipulation, mathematical calculations and pattern recognition. The synchronization intensive applications like wavefront computations are widely applied in the scientific computing, dynamic programming algorithms and particle physics [110]. We have mapped some representative programs for the scalability analysis of memory models which help us to derive some meaningful conclusions from the experiments.

### 4.1.2 Performance Metrics for the Scalability Analysis

To analyze the scalability of different memory consistency models which are realized in the McNoC systems, we have developed and mapped some application workloads on the different sized networks. The scalability analysis is performed in the McNoC systems with 1 to 64-cores. The developed applications are manually mapped on the increasing size of the network. The performance metrics like *execution time*, *performance*, *speedup*, *overhead* and *efficiency* are evaluated and compared under different memory consistency models as the network size is scaled up.

- *Execution Time* (*ET*)*:* The *execution time* of a workload is defined as the time from the start of execution on first processor to the end of execution on last processor in the system.

- *Performance:* The *performance* is the reciprocal of execution time. It is measured in kilo-operations per second.

- *Speedup* (*SP*)*:* The *speedup* is defined as the ratio of the execution time (*Ts*) of the single core and the execution time of the multi-core (*Tm*).

- *Communication Overhead* (*OH*)*:* We define the *communication overhead* of the multi-core system as: *Nc***Tm*-*Ts*, where *Nc* is the number of cores in the system.

- *Efficiency* (*EF*)*:* The *efficiency* of multi-core system is defined as the ratio of *SP* and *Nc*.

In the experiments, the effects of the network size on the *execution time*, *performance*, *speedup*, *communication overhead* and *efficiency* are investigated under different memory consistency models in the McNoC systems.

### 4.1.3 Scalability Analysis

From the perspective of one application, the execution time of an application mainly comprises of the computation and communication costs. When an application is mapped on a single core (one node), it operates on the local data that is available within the node. The communication cost is least, because all the memory accesses/references

are accomplished within the same one node. When the same application is mapped in the network then it also involves a significant communication cost. When the number of nodes is doubled, the computation cost is perfectly divided into half due to the identical processors in the system (Figure 4.1). But, the communication overhead is increased as a result of the increasing *physical distances* between the nodes. The communication overhead is mainly due to the message passing and data transfer in the system. A processor can access the shared memory locations anywhere in the network, because the system supports the DSM architecture. The communication overhead must be handled efficiently in the larger networks in order to increase the speedup and efficiency of the systems. The system supporting the SC model does not handle the communication overhead effectively, because it allows one outstanding transaction at a time in the network. Alternatively, the relaxed memory consistency models like PRC and RC significantly reduce/mitigate the communication overhead, by allowing outstanding transactions in the network which can be reordered, overlapped and pipelined with respect to each other. It is observed under the experimental results which are discussed in the next section.

From the perspective of different applications, the scaling behavior of the different memory consistency models under different application workloads depends on the type of application and the ratio of computation and communication costs. For instance, one application is communication intensive and has less computation-to-communication ratio compared to another application. Under the former application, the relaxed memory consistency models like PRC and RC significantly reduce the communication overhead by pipelining more outstanding transactions in the network. The PRC and RC models show even better and scalable performance compared to the SC model.

The workloads which use the synchronization primitives (e.g., locks, semaphores) show different behavior than the data intensive applications (which are not using the synchronization primitives). The system may show poor scaling behavior under the synchronization intensive applications workloads. The synchronization overhead among the cores is mainly due to the waiting time for the lock acquire and the network congestion. The network congestion also increases due to re-sending of the acquire requests on failure till the acquisition of a lock. The synchronization overhead should be handled effectively to improve the speedup and efficiency of the system. The system supporting the distributed locks scheme compared to the centralized scheme can significantly reduce the synchronization overhead. The different set of nodes (segments) within the network can synchronize in parallel over the distributed locks in the system. Within DSM systems, each processor in the system (Figure 4.1) can access the shared memory locations which are distributed across the network. Due to the non-uniform memory access time, outstanding transactions under the relaxed memory consistency models like PRC and RC are reordered and overlapped with respect to each other in the network. Thus, high speedup and efficiency are achieved compared to the strict SC model.

## 4.2 Scalability Analysis of the RC and SC models

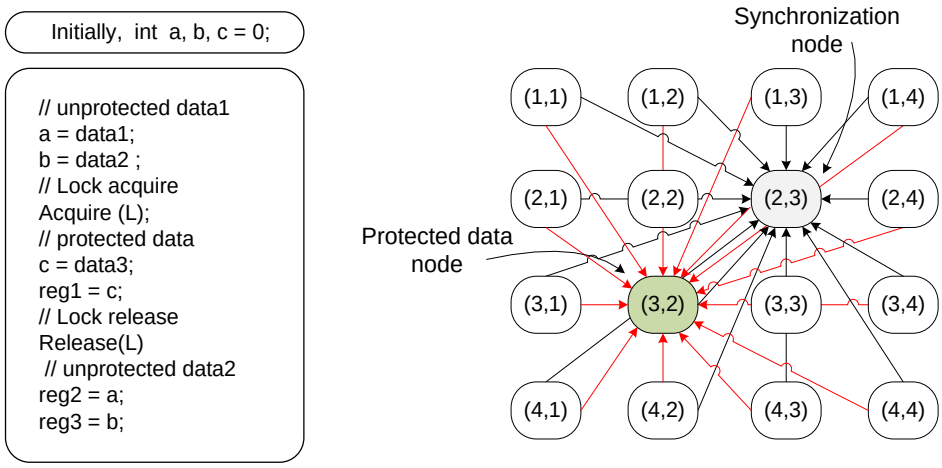
This section summarizes the scalability analysis of the RC and SC models which is proposed in [22]. The scalability of the RC and SC models is analyzed by mapping bit count and pattern search applications on the different sized networks using different problem sizes. The experiments are also conducted with the synthetic workloads to evaluate the performance of the RC and SC models using different distributed locks in the McNoC systems.

### 4.2.1 Experiments with Synthetic Workloads

As discussed earlier, our platforms support the distributed locks scheme. In order to illustrate the benefits of the distributed locks, the performance of the RC and SC models are evaluated with the synthetic workloads. These synthetic workloads are manually mapped on the LEON3 processors in the McNoC system. A processor in each node of the network produces the same sequence of transactions under both these memory consistency models. The synthetic workload(s) mapped on the processors are given in Figure 4.2(a). These synthetic workloads have both data and synchronization operations. The synchronization primitives are used to execute the portion of the code in the critical (or protected) section in a sequential order among the multiple processors in the system. For the lock and protected (critical section) data operations, hotspot traffic pattern is generated as shown in Figure 4.2(b). For the unprotected data operations, the uniform random traffic pattern is generated. We have performed the experiments in the 8x8 network (64-cores system). For the *N*-locks, the network is divided into *N* equal segments. For the 8x8 network, the number of segments is varied from 1 to 32. The whole 8x8 network is treated as one segment. For the two segments in the 8x8 network, each segment includes 32-nodes. All these 32-nodes mutually agrees on a common lock that is maintained in one of the 32-nodes of the same segment. Similarly, this trend goes up to the 32-segments in the 8x8 network. Under 32-segments in the 8x8 network, each segment comprises of two nodes. For the 32-segments, the two nodes synchronize over a common lock maintained in one of the two nodes in the segment. In other words, all these nodes within a segment synchronize over a common lock which is maintained in a node that belongs to the same segment in the network.

The performance of the RC and SC models are evaluated using different number of segments/locks in the 8x8 system. As discussed earlier, the number of segments (e.g., locks) is varied from 1 to 32 in the 8x8 network. The performance results under both the RC and SC models are given in Figure 4.3. As the number of segments or locks increases in the network, the performance also quickly increases under both the RC and SC models. It is due to the fact that different segments synchronize over different distributed locks in the network. When the number of locks increases, the waiting time for the lock acquire on average is also reduced, because the network congestion and traffic is decreased. The RC model performs better compared to the SC model due to the reordering and relaxation of the shared memory operations. For the 32 locks or segments, two nodes in each segment of the network synchronize over a common lock and the synchronization wait

time and the network congestion is considerably reduced. As a result, the performance of the RC model over SC model is further increased when the number of segments is increased in the network. On the average, performance under the RC model for 1 to 32-segments is increased by 17.6% to 54.6% over the SC model.




(a) (b)

**Figure 4.2.** a) Sequences of transactions generated. b) Traffic Patterns.

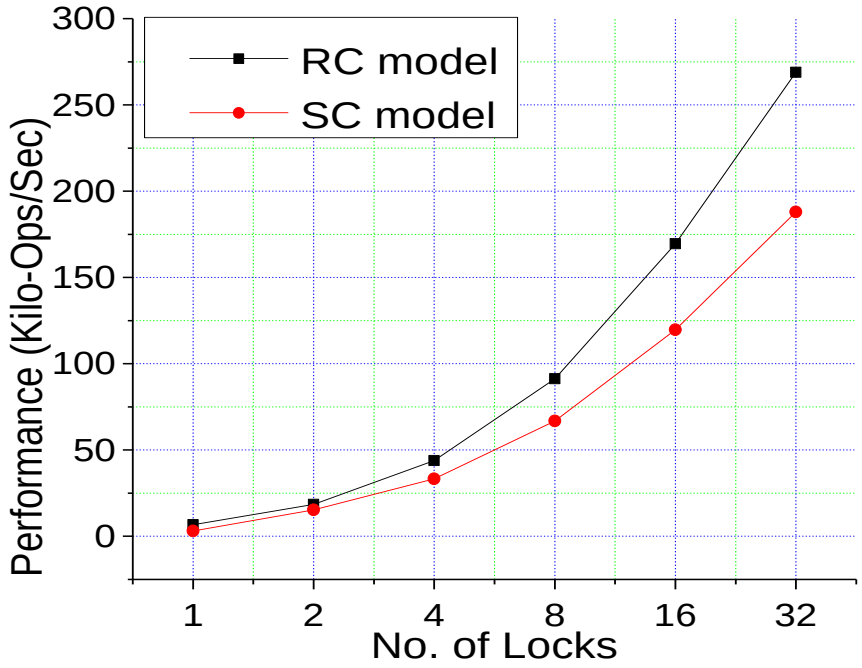



**Figure 4.3.** Performance of the RC and SC models.

The execution time of the 64-cores system normalized to a single core system (Normalized-ET1C) is shown in Figure 4.4. The Normalized-ET1C in the ideal case is (1) by neglecting the communication overhead in the 8x8 system, which means that the

execution time of the 64-cores system is identical to that of the single core system. It is due to the fact that the same sequence of transactions is generated in each node of the network. Note that, identical processors are used in each node of the network. The deviation of actual Normalized-ET1C under both the RC and SC models from the ideal case decreases when the number of segments or locks is increased in the 8x8 system.

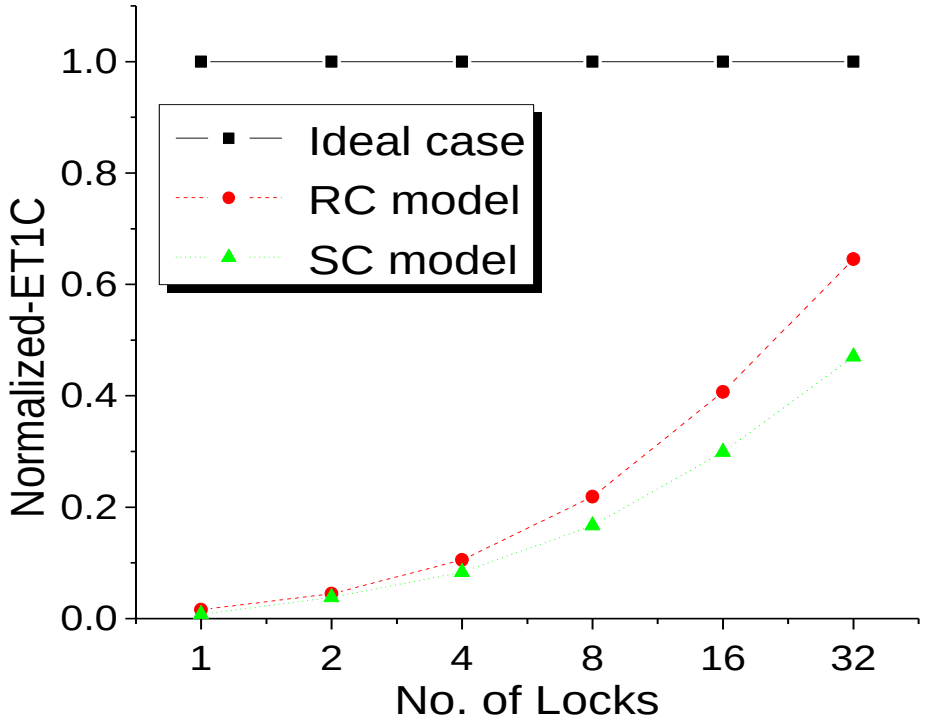



**Figure 4.4.** Normalized-ET of 64-cores to a single-core system.

### 4.2.2 Experiments with Applications Workloads

#### 4.2.2.1 Bit Count

The bit count application analyzes a data vector and calculates the number of set bits (1) in each input integer data item. After initialization these input data items are read, analyzed and the bit counts are calculated. The output values are stored in the distributed shared memory in the system. For the experiments, different sizes of input vectors are used with (16, 32, 64, 128, 256, 512 and 1024) problem sizes. The network size is increased from 1 to 64-nodes (e.g., 8x8 system). When a data vector of 16 elements is mapped on the 4x8 and 8x8 systems, only 16-nodes are involved in the computation process. Similarly, when a data vector of 32 elements is mapped on the 8x8 system, only 32-nodes perform the computations. For the rest of the data vectors, all the nodes in the

8x8 system are involved in the computation process. Each node operates on the data items in the *randomly* selected node and also writes the output results within the same node.

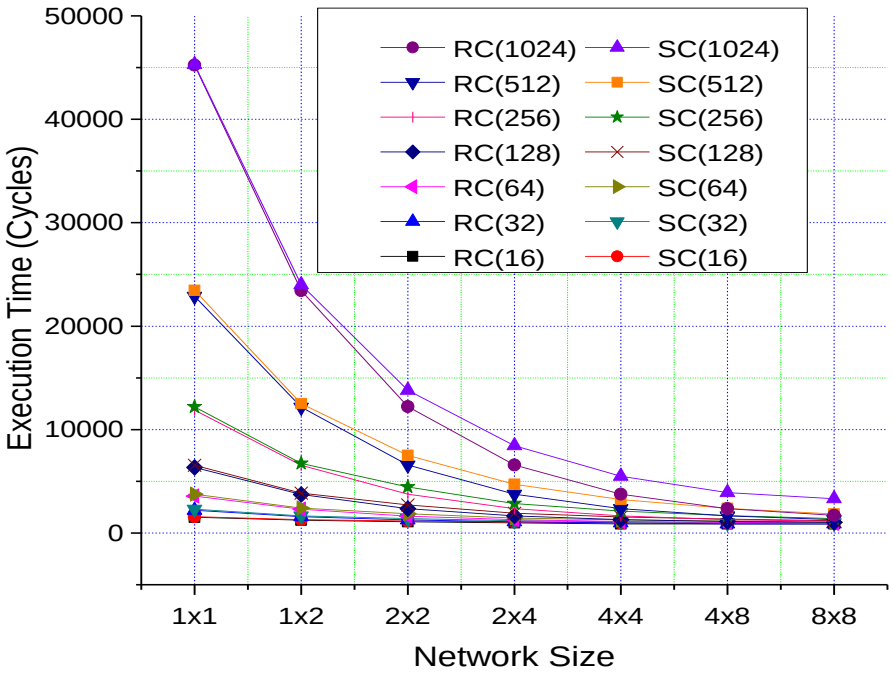



**Figure 4.5.** Execution time under bit count application.

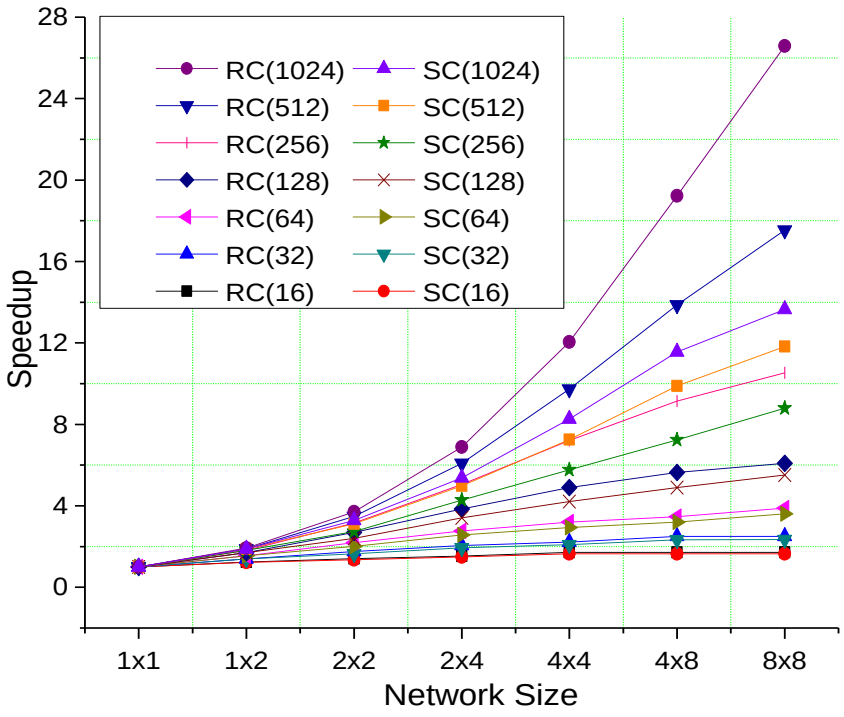



**Figure 4.6.** Speedup under bit count application.

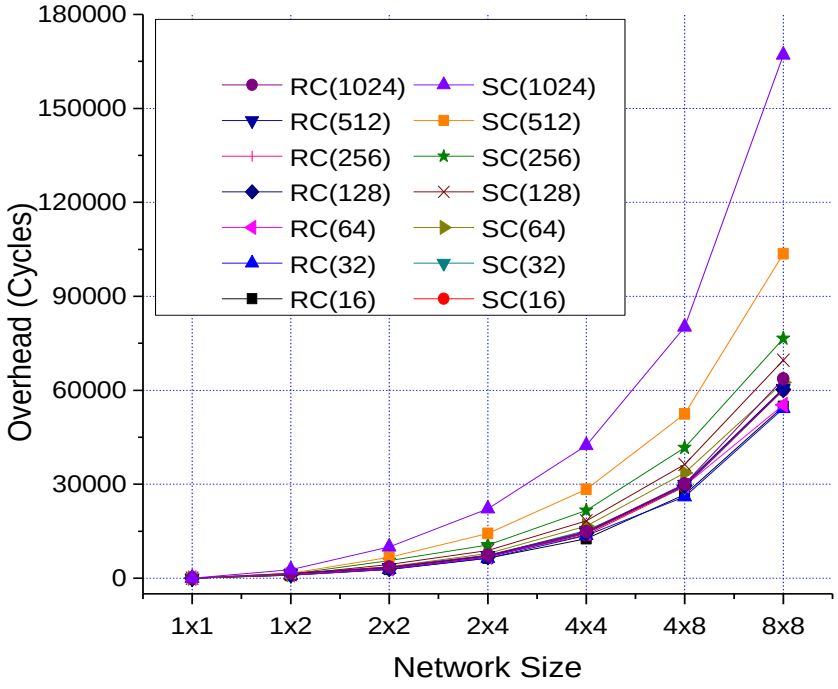



**Figure 4.7.** Communication overhead under bit count application.

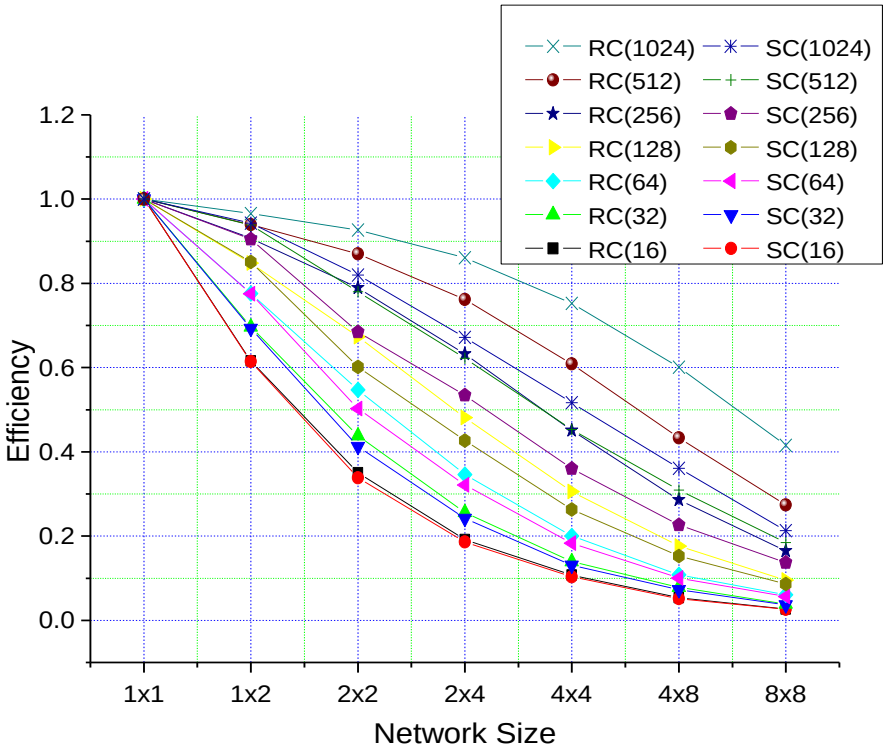



**Figure 4.8.** Efficiency under bit count application.

For the bit count application, the execution time, speedup, communication overhead and efficiency of the RC and SC models under different sizes of the network are demonstrated in Figures 4.5 to 4.8. We have also used different problem sizes under the bit count application. As illustrated in Figure 4.5, the Application workload Execution Time (AET) is decreased when the system size is increased from 1 to 64-cores. This is due to the division of computation cost in the network. The computation cost is significantly reduced due to the parallel processing in the larger networks. The RC model further decreases the AET compared to the SC model by allowing reordering and relaxation among the shared memory operations in the network. The problem size also affects the AET reduction. When the network size is scaled up, the AET reduction is higher under the larger problem. It is due to the parallelization of a significant amount of computation cost in the system. For the 1024-problem, the AETs in the single core system are 26.6 and 13.6 times of that in the 64-cores systems under the RC and SC models, respectively. Similarly, the speedup as shown in Figure 4.6 grows faster under the RC model compared to the SC model when the system size is scaled up. The RC model maintains even higher speedup under the larger problems. This is due to the efficient handling of the communication overhead under the RC model as given in Figure 4.7. The RC model allows more outstanding operations over the network which are pipelined and overlapped with respect to each other. For the larger problem like 1024, the communication overheads in the 64-cores systems are 39.6 and 60.9 times of the overheads in the two-core systems under the RC and SC models, respectively. The RC model effectively controls the communication overhead compared to the SC model by pipelining the memory transactions over the network. When the network size grows up, the RC model compared to the SC model maintains high efficiency under these different problem sizes as shown in Figure 4.8. Overall, the RC model illustrates better scalability and efficiently utilizes the system resources in the larger networks under different problem sizes compared to the SC model. The execution time and communication overhead are kept lower and the speedup and efficiency are maintained higher under the relaxed RC model compared to the strict SC model.

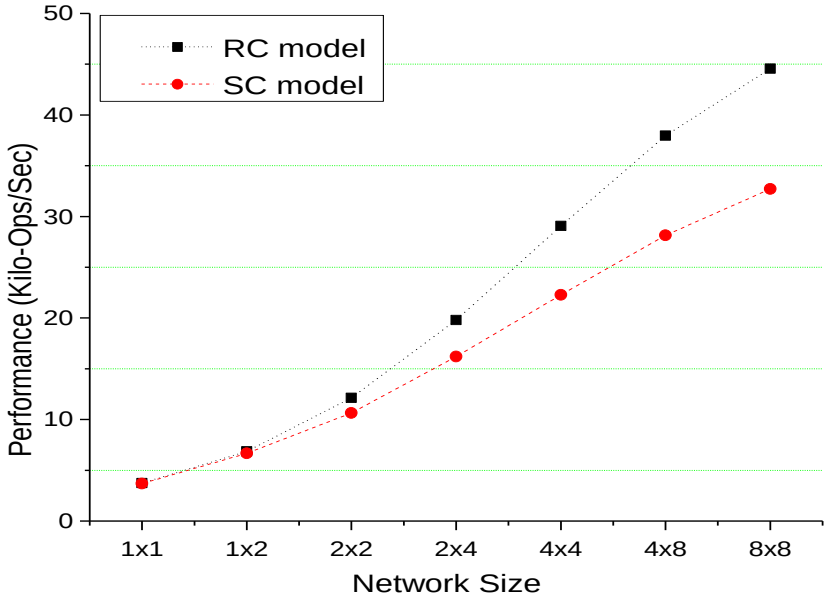



**Figure 4.9.** Average performance under bit count application.

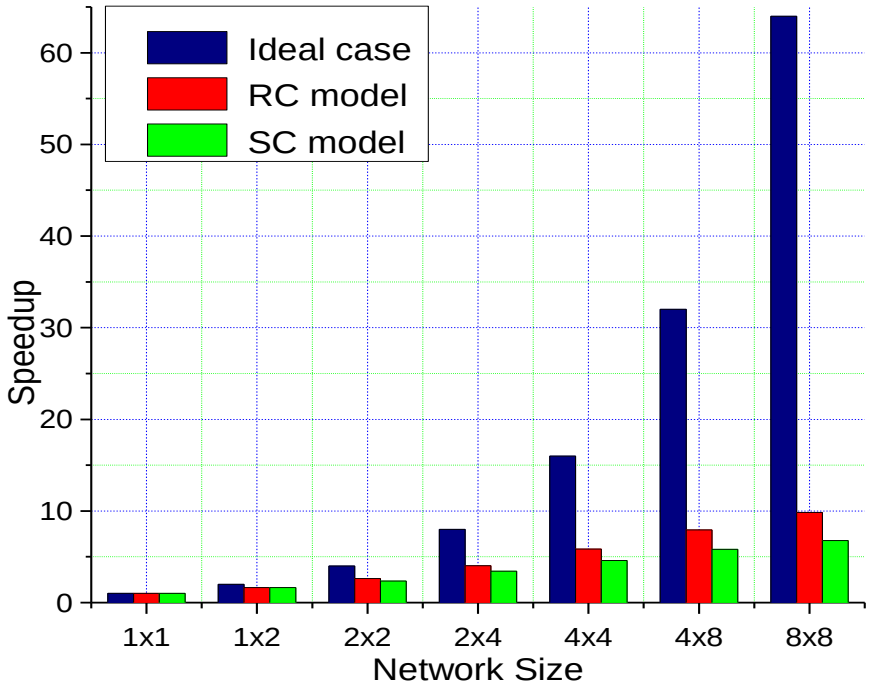



**Figure 4.10.** Average speedup under bit count application.

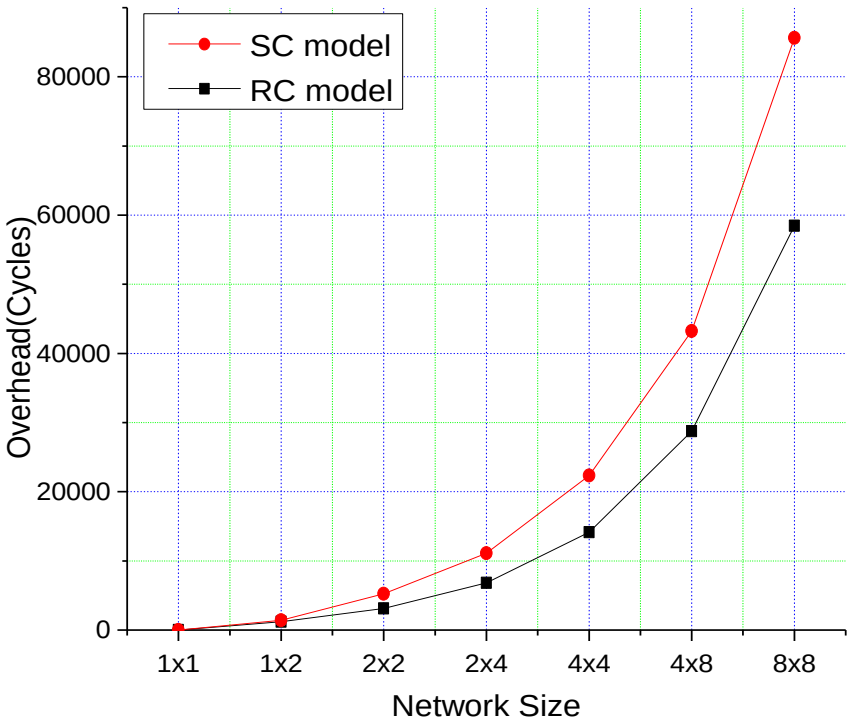



**Figure 4.11.** Average communication overhead under bit count application.

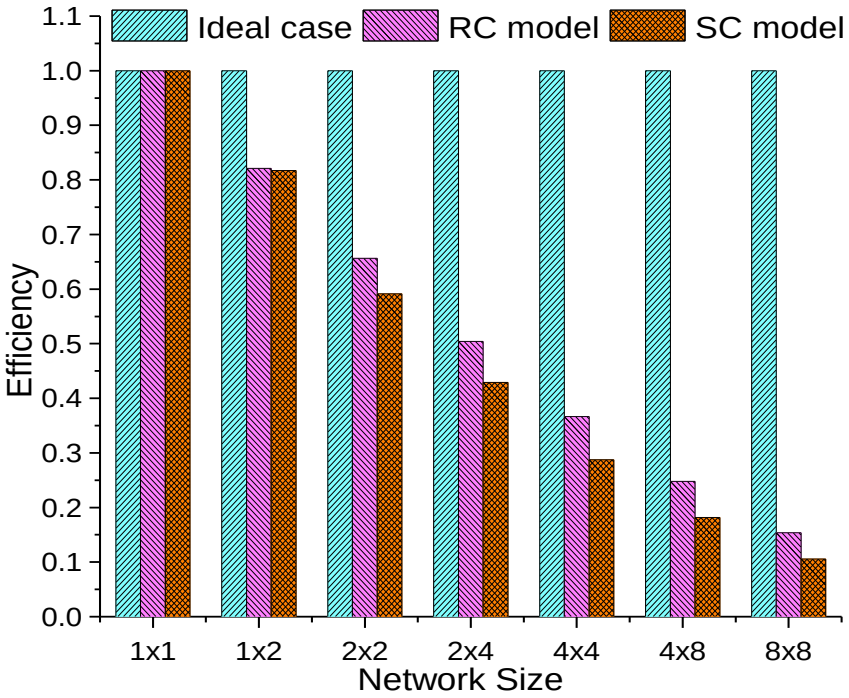



**Figure 4.12.** Average efficiency under bit count application.

The *average* performance, speedup, overhead and efficiency under bit count application for the RC and SC models are given in Figures 4.9 to 4.12. As shown in Figure 4.9, the average performance under the RC and SC models in the 64-cores systems compared to the single core systems are 12 and 8.8 times higher, respectively. It is due to the reordering and relaxation among the memory operations which is allowed/offered under the RC model compared to the SC model. The average speedup as illustrated in Figure 4.10 deviates from the ideal speedup when the network size is scaled from 1 to 64-cores. The ideal speedup assumes zero communication overhead in the network, therefore, it grows linearly along with the increasing size of the network. Likewise, the deviation is more under the SC model compared to the RC model in the 64-cores system, because the SC model does not allow pipelining and overlapping among the memory operations. As demonstrated in Figure 4.11, the average communication overheads under the RC and SC models in the 64-cores systems compared to the two-core systems are 48.8 and 61.2 times, respectively. The average communication is decreased by 31.8% under the RC model compared to the SC model in the 64-cores system. Similarly, the average efficiency as given in Figure 4.12 is maintained higher under the RC model compared to the SC model when the network size grows up. The average efficiency under the RC model is increased by 31.2% compared to the SC model in the 64-cores system.

#### 4.2.2.2 Pattern Search

The application searches the data patterns (*P)* against the data elements (*D)*. These data patterns and elements are initialized in the shared memory across the network. After initialization, these patterns and elements are read and analyzed. The outputs are the number of times that the patterns appear in the data elements, which are written to the shared memory in the local node. We have experimented with four different cases which use 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 computation process. *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. Only, 32-nodes are involved in the computation process. *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 these 64-nodes are involved in the computation process. *P64-D64:* one pattern and data element each is mapped on each node in the 8x8 network and every node perform the computation.

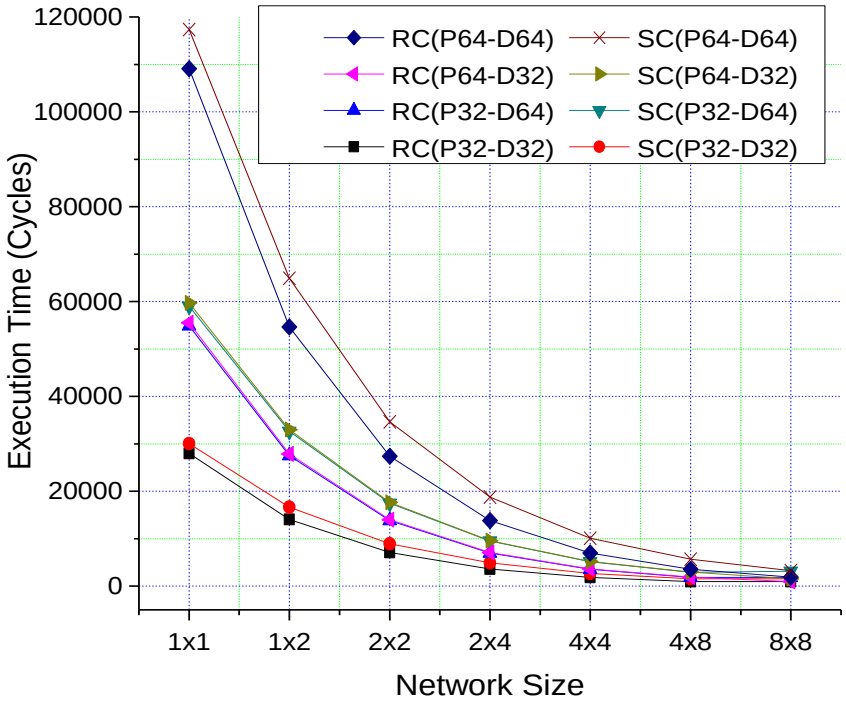



**Figure 4.13.** Execution time under pattern search application.

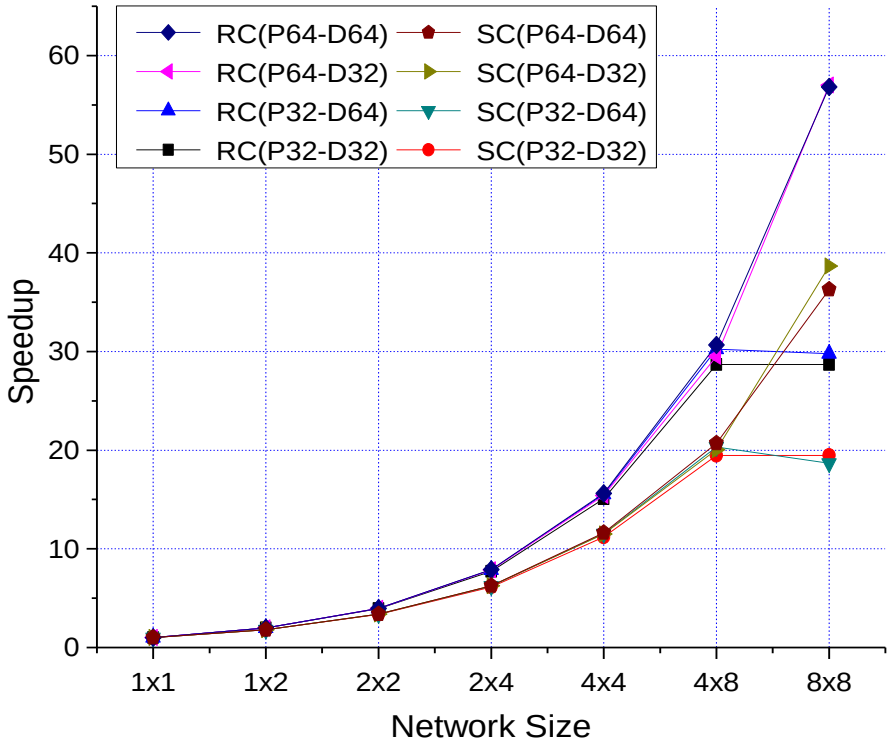



**Figure 4.14.** Speedup under pattern search application.

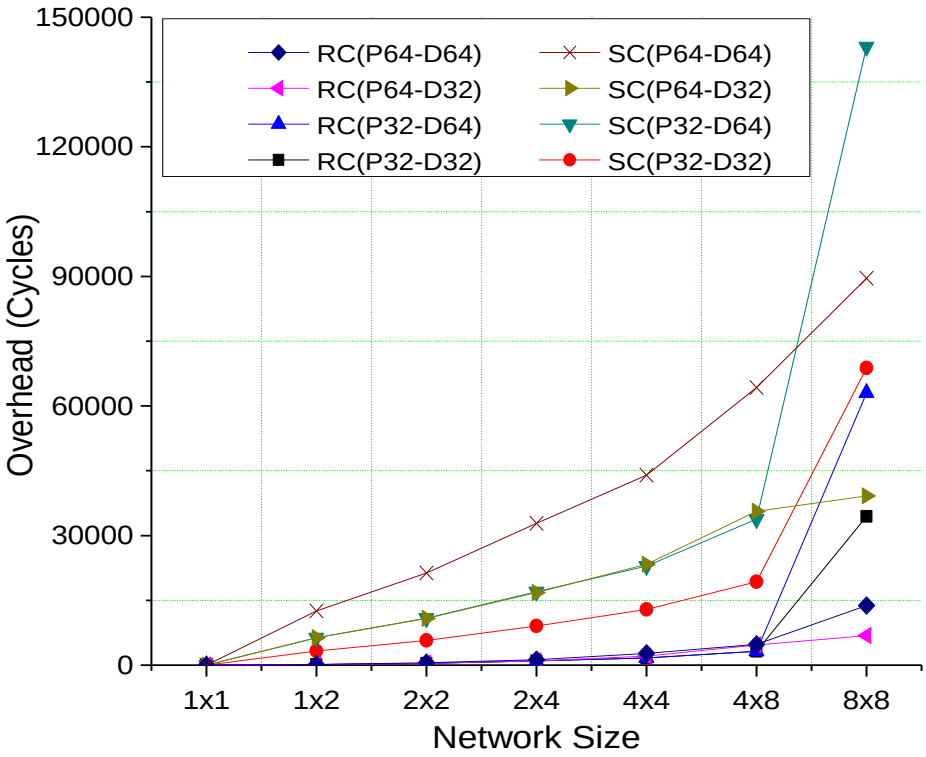



**Figure 4.15.** Communication overhead under pattern search application.

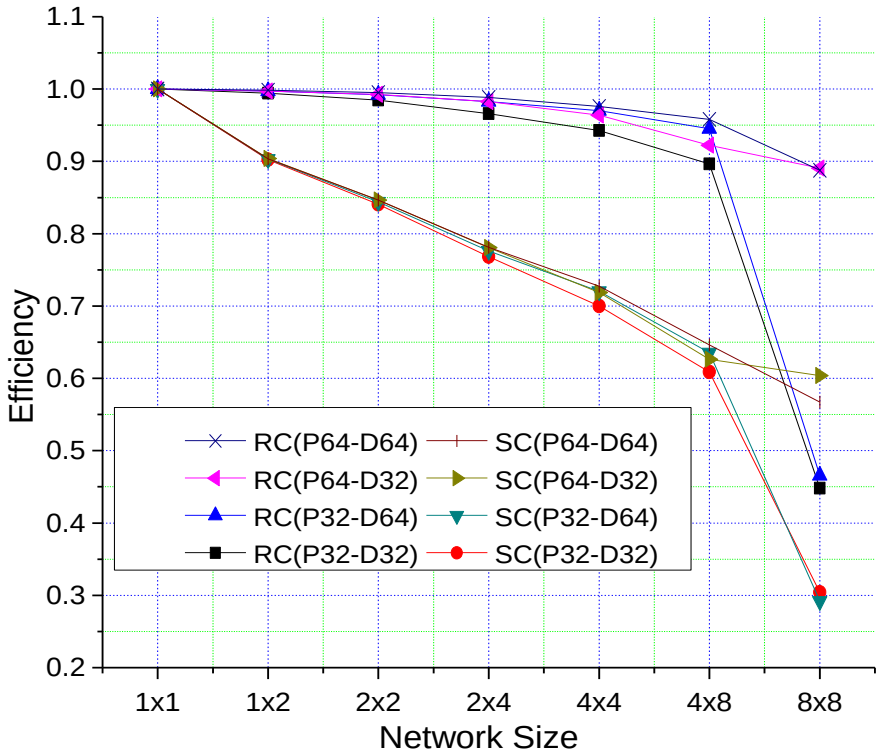



**Figure 4.16.** Efficiency under pattern search application.

The execution time, speedup, communication overhead and efficiency of the RC and SC models for different sized networks and different problem sizes under the pattern search application are illustrated in Figures 4.13 to 4.16. As shown in Figure 4.13, when the system grows up from 1 to 64-cores, the reduction in the execution time is higher under both the memory consistency models for the (P64-Dx) problem sizes. It is because, these problem sizes fit well into the increasing size of the network compared to the (P32-Dx) problem sizes. For example, better scaling behavior is observed under (P64-D32) problem, where the problem size fits well into the 8x8 network and each node is involved in the computation process. A significant amount of computation time is parallelized among the nodes in the larger networks. The AETs are reduced considerably when the system size is scaled up from 1 to 64-cores. For the RC and SC models, the AETs in the single core systems are 57 and 38.6 times of that in the 64-cores systems, respectively. The reduction in AET under the RC model over the SC model is higher due to the pipelining and overlapping of the shared memory operations in the network. The pipelining among the memory operations under the RC model in the network significantly reduces the memory access time and the system performance is enhanced. Likewise, the speedup grows quickly under the RC model compared to the SC model as shown in Figure 4.14. The speedup levels off after 32-nodes up to 64-nodes under the (P32-Dx) problem sizes, since the same amount of computation is performed in the 4x8 and 8x8 networks. Therefore, the execution time almost remains the same both in the 4x8 and 8x8 networks. The communication overhead as given in Figure 4.15 under both these memory consistency models is significantly increased for the (Px-D64) configurations as the network size is increased. It is because, for each pattern, all these 64 data elements are

searched which are distributed across the network. As observed in Figure 4.16, the efficiency is maintained higher under the RC model compared to the SC model when the network size is increased. On the whole, the RC model again maintains lower execution time and higher speedup compared to the SC model. The communication overhead is effectively controlled under the RC model over SC model and also high efficiency is achieved under the RC model in comparison to the SC model.

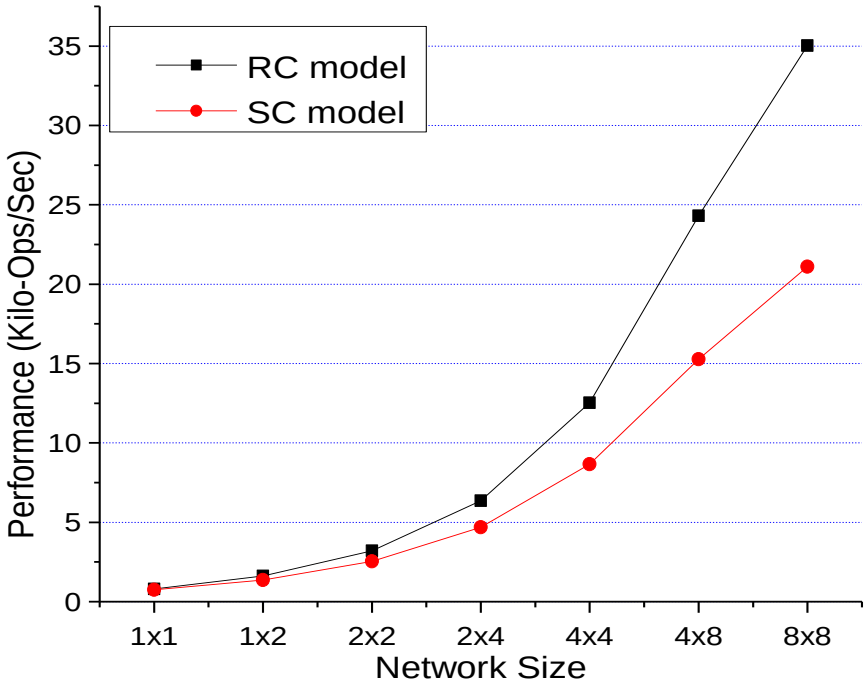



**Figure 4.17.** Average performance under pattern search application.

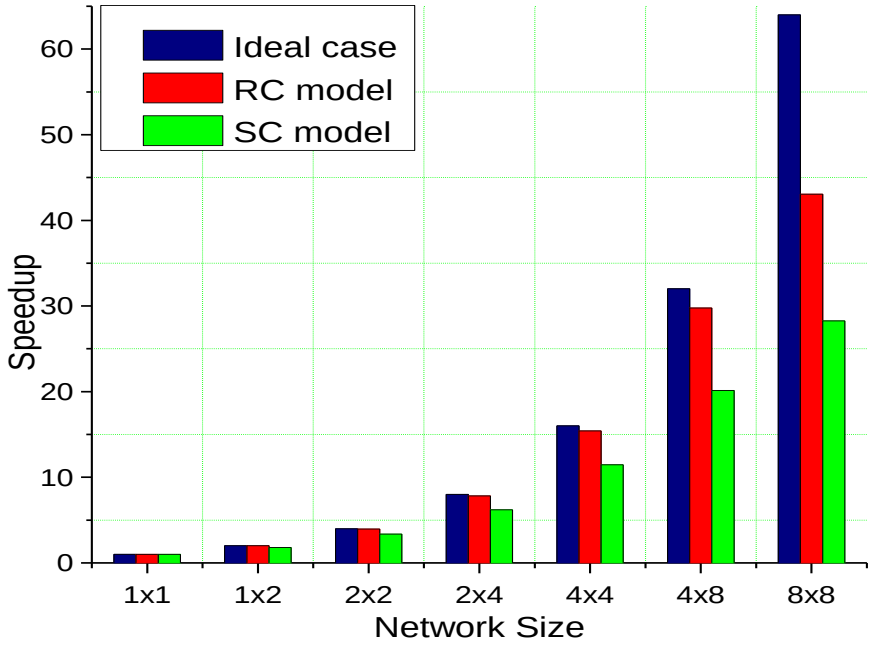



**Figure 4.18.** Average speedup under pattern search application.

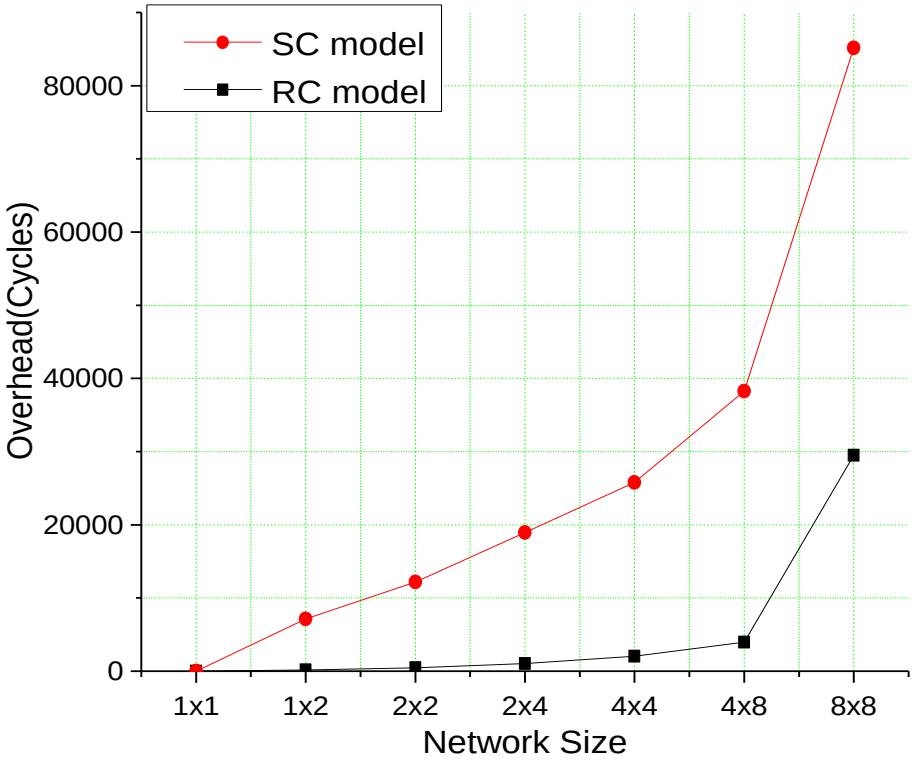



**Figure 4.19.** Average communication overhead under pattern search application.

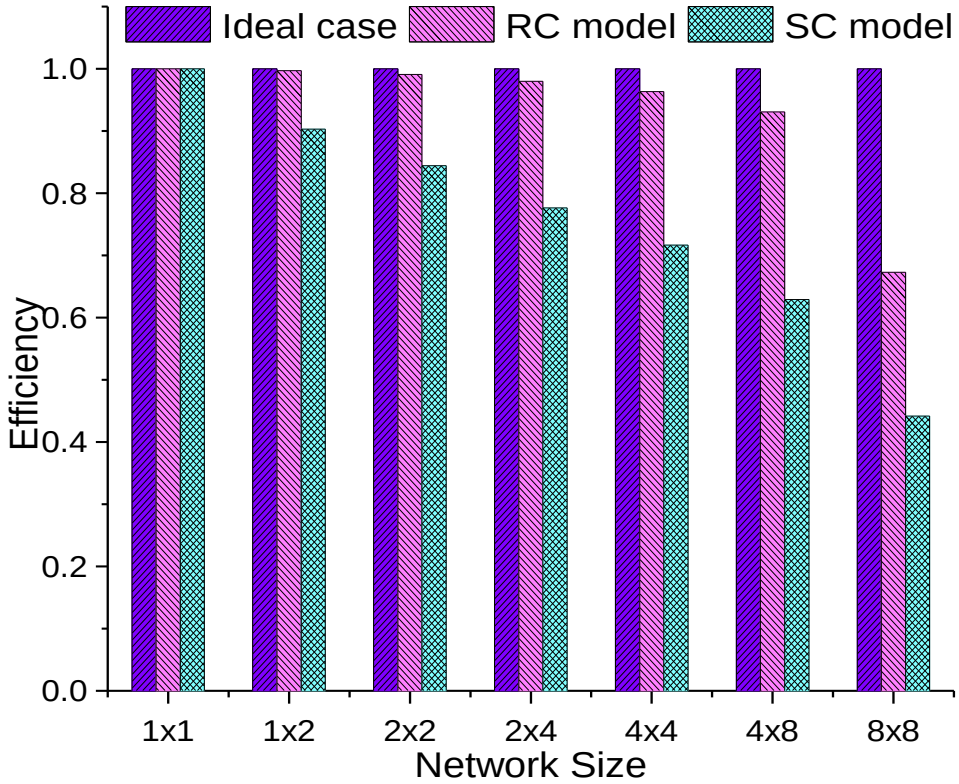



**Figure 4.20.** Average efficiency under pattern search application.

The average performance, speedup, communication overhead and efficiency under the pattern search application for the RC and SC models are shown in Figures 4.17 to 4.20. In contrast to the bit count application, the performance (Figure 4.17) and speedup (Figure 4.18) are higher under both the memory consistency models in the 64-cores systems over the single core systems. This is due to the *low* computation-to-communication ratio under the pattern search application compared to the bit count application. The computation time per input data is more (21 cycles) under the bit count application compared to that under the pattern search application (9 cycles). Additionally, 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. As shown in Figure 4.17, the average performance of the RC and SC models under the pattern search application is 31.4% and 19.2% higher than that under the bit count application (Figure 4.9). The average speedup as shown in Figure 4.18 for the RC model is 43.1 that is almost close to the ideal case of 64, 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 respect to each other. As demonstrated in Figure 4.19, the average communication overhead is controlled efficiently under the RC model with the increasing size of the network. The communication overhead is considerably reduced under the RC model due to the pipelining of outstanding data operations over the network. The reduction in communication overhead under the RC model over SC model is quite high under the pattern search application compared to that under the bit count application (Figure 4.11). As long as the system size increases, the average efficiency as shown in Figure 4.20 under both these memory models is maintained high (close to the ideal case 1) compared to that given under the bit count application (Figure 4.12). On average, the efficiencies in the 64-cores systems for the RC and SC models are 0.67 and 0.44, respectively.

### 4.2.3 Summary of the Scalability Analysis of RC and SC Models

To summarize the scalability analysis of the RC and SC models, the execution time of RC model has been between 50% to 100% of the SC model. The specific numbers are highly sensitive to the application's type and the problem size that how well it fit to the architectures. However, the observed trends in the experimental results imply that the RC model scales inherently better with the network size compared to the SC model. As given in Figure 4.21, the execution time is very similar for the small networks under the bit count application. As the network size scales up, the execution time under the RC model decreases relative to the SC model. The RC model effectively utilizes the system resources in the larger networks by allowing reordering and relaxation among the memory operations. At some point going along the increasing size of the network, the relative execution time under the RC model decreases and flattens off. It depends logically on the application and its match to the platform, when exactly this leveling off occurs. As long as the speedup increases with the increasing size of the network, the benefits of the RC model increase over the SC model. However, some problem sizes

which do not fit well into the architectures cannot exploit the additional parallelism and the benefits of the RC model over the SC model do not increase further. However, problems that scale better, like the 1024-bit count problem, continue to obtain more benefits from the higher level of parallelism that the RC model offers compared to the SC model. We expect the same trend as shown in Figure 4.21 to continue in the larger networks, which means that the performance benefits of the RC model continue to increase for the well matched problems as long as the network size grows. The same trend can also be observed in Figure 4.22. Hence, we summarize that the performance increase of the RC model over SC model can be significantly higher than 50% as observed in our experiments.

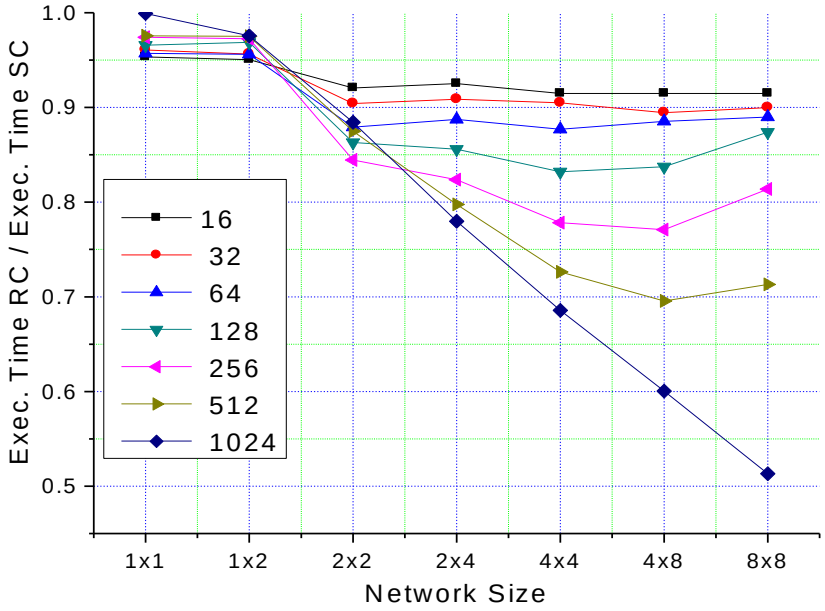



**Figure 4.21.** Bit count: ratio of AETs (RC/SC).

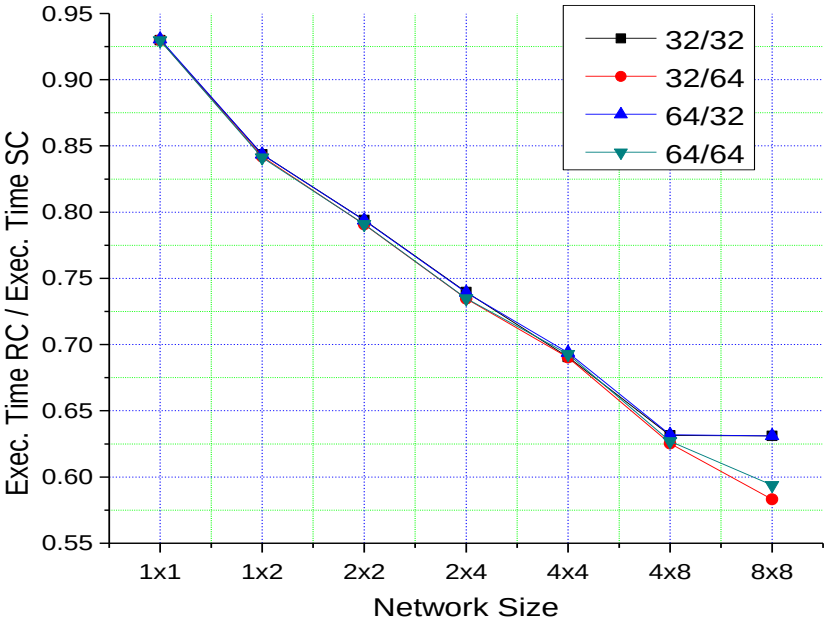



**Figure 4.22.** Pattern search: ratio of AETs (RC/SC).

## 4.3 Scalability Analysis of Six Memory Consistency Models

This section summarizes the scalability analysis of six different memory consistency models (e.g., SC, TSO, PSO, WC, RC, and PRC) in the McNoC systems. Most of the material under this section is already reported in [23]. The scalability analysis of these different memory consistency models is carried out in the McNoC systems with 1 to 64-cores by developing and mapping the application workloads on the increasing size of the network using different problem sizes. The experiments are conducted with both the synthetic and application workloads. The application workloads are chosen from different domains. The scalability study underlines some interesting and key factors which affect the performance gain of the relaxed memory models over the stricter memory models. For instance, these factors include: the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio and the problem size.

### 4.3.1 Experiments with the Synthetic Workloads

The performance of six different memory consistency models is evaluated using different synthetic workloads (SWL1 to SWL5) as given in Figure 4.23. These synthetic workloads are manually mapped on the processors in the system. The same sequence of transactions is generated by the LEON3 processor in each node of the network. Therefore, when the number of cores is doubled, the total amount of work is also doubled. The SWL1 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. The SWL2 contains a write followed by a write, a write followed by a read and a read followed by a read sequence. The SWL3 has also the case of a read followed by a write operation and uses two *non-overlapped* protected or critical sections. The SWL4 uses *nested* protected sections which are protected under two different locks. The SWL5 uses *partially overlapped* protected sections that are protected under two different locks. For the lock and protected (critical section) data operations, hotspot traffic pattern is generated. For the unprotected data operations, the uniform random traffic pattern is generated.

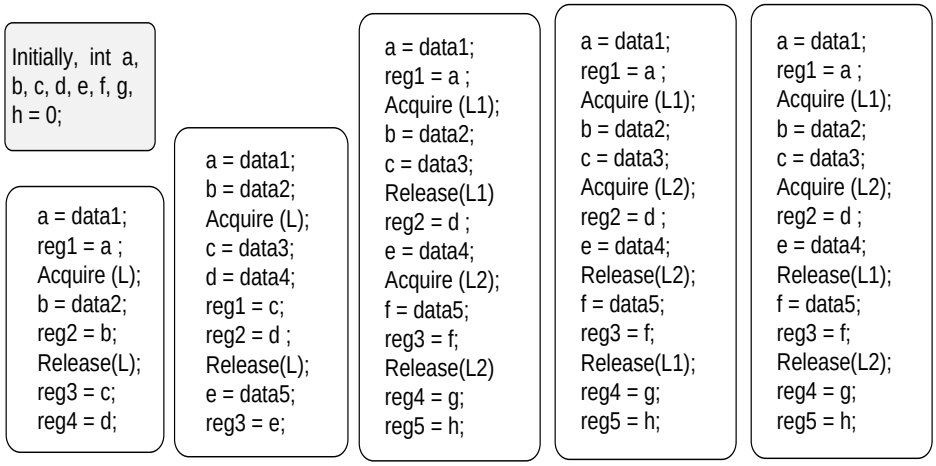



**Figure 4.23.** Transactions sequences: a) SWL1; b) SWL2; c) SWL3; d) SWL4; e) SWL5.

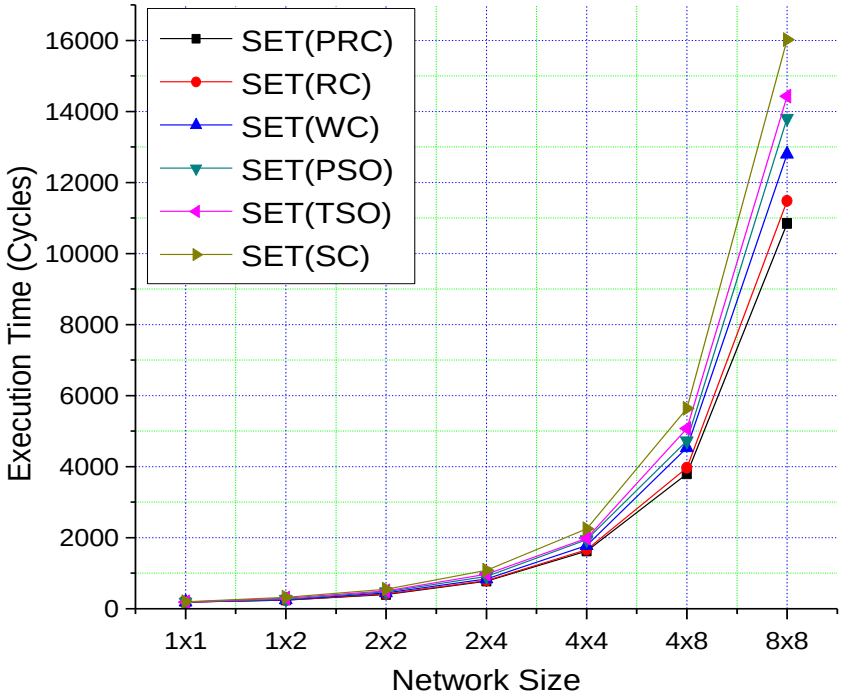



**Figure 4.24.** Average SETs for SWL1-SWL5.

The Synthetic workload Execution Times (SETs) are compared for the six different memory consistency models (SC, TSO, PSO, WC, RC, and PRC) in Figure 4.24. The SC model is used as the baseline model. The performance of relaxed memory consistency models is compared to the SC model, because it constitutes the worst case. When the system size is scaled up, the SETs quickly increase under all the memory consistency models due to the increasing traffic, network congestion and waiting time for a lock acquire. The relaxed memory consistency models compared to the strict SC model maintain lower SETs due to the reordering and relaxation in the shared memory operations. The average SETs under the PRC, RC, WC, PSO and TSO models in the 8x8 network are decreased by 32.3%, 28.3%, 20.1%, 13.8% and 9.9% over the SC model, respectively. The SETs under the relaxed memory consistency models over the SC model are reduced more under the SWL2 due to the issuance of more data operations before a release operation.

### 4.3.2 Experiments with Application Workloads

#### 4.3.2.1 Angle Conversion

The angle conversion application converts the input data vector of degrees into the corresponding output data vector of radians. The input data vector of 128 elements is used. The input data vector is initialized in the distributed shared memory across the network. A uniform random traffic pattern is generated to read these input data. The calculated output vector is stored in the shared memory of the local node.

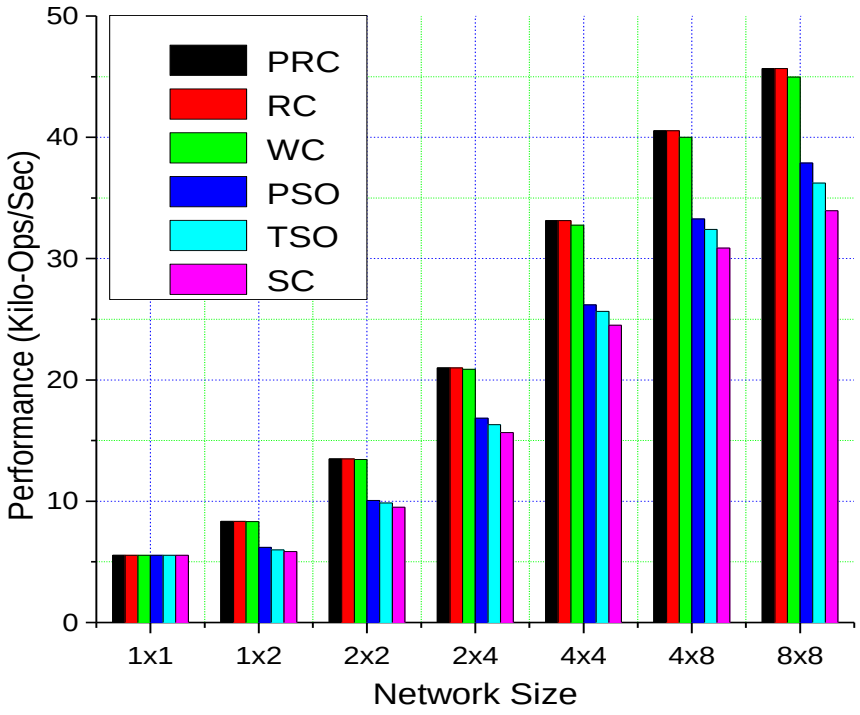



**Figure 4.25.** Performance under angle conversion application.

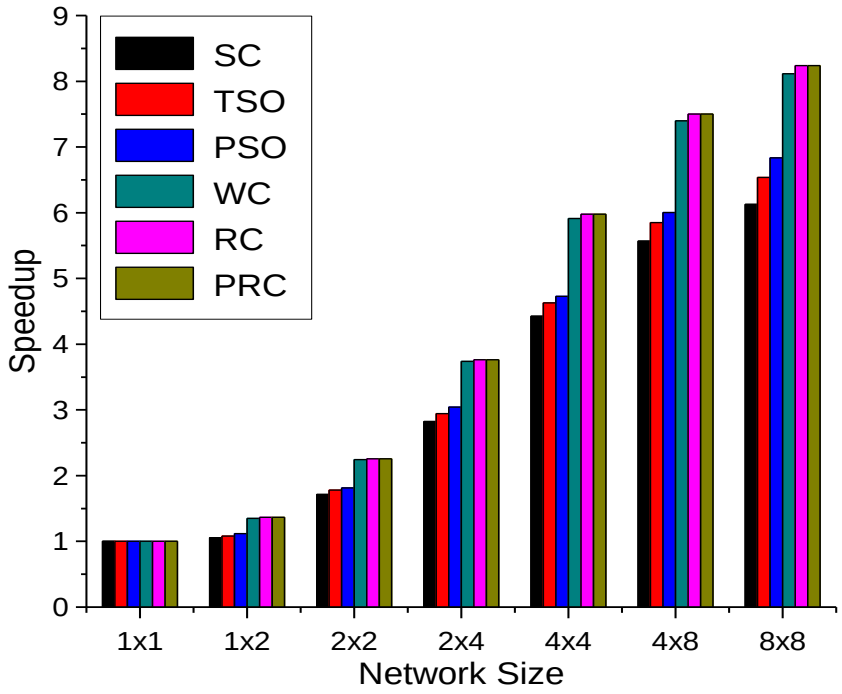



**Figure 4.26.** Speedup under angle conversion application.

The performance comparison of six different memory consistency models under the angle conversion application is illustrated in Figure 4.25. The performance of all these memory consistency models increases when the system size is scaled up from 1 to 64-cores. This is due to the parallelization of computation cost in the larger networks. The computation cost is perfectly divided among the identical processors in the larger systems. Among the relaxed memory consistency models, the PRC and RC models further improve the performance compared to the stricter memory consistency models by allowing additional reordering and overlapping in the shared memory operations. The performance under the PRC, RC, WC, PSO, TSO and SC models in the 64-cores systems are 8.3, 8.3, 8.1, 6.8, 6.5 and 6.1 times higher than that in the single core systems, respectively. Similarly, the speedup as shown in Figure 4.26 grows faster under the PRC and RC models compared to the other memory consistency models.

#### 4.3.2.2 Bit Count

The bit count application is the same as discussed earlier under the scalability analysis of the RC and SC models in the previous section. It analyzes the input data vector and calculates the number of set bits in each integer data item. The input data vector is initialized, read, analyzed and the resultant output vector is stored in the distributed shared memory. The input data vector has 512 elements. Each node operates on the data items in the *randomly* selected node and the output results are stored in the shared memory of the same node.

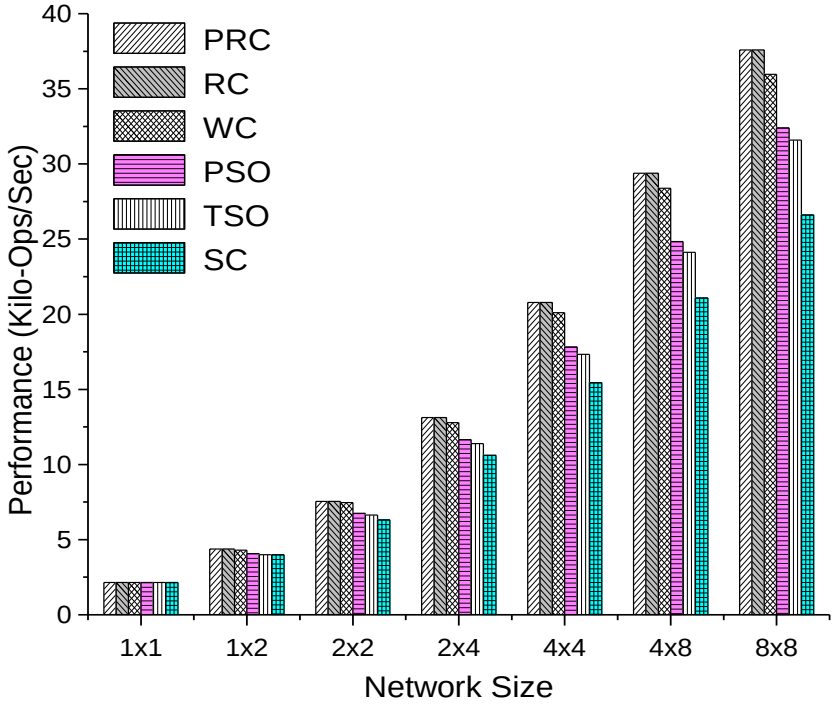



**Figure 4.27.** Performance under bit count application.

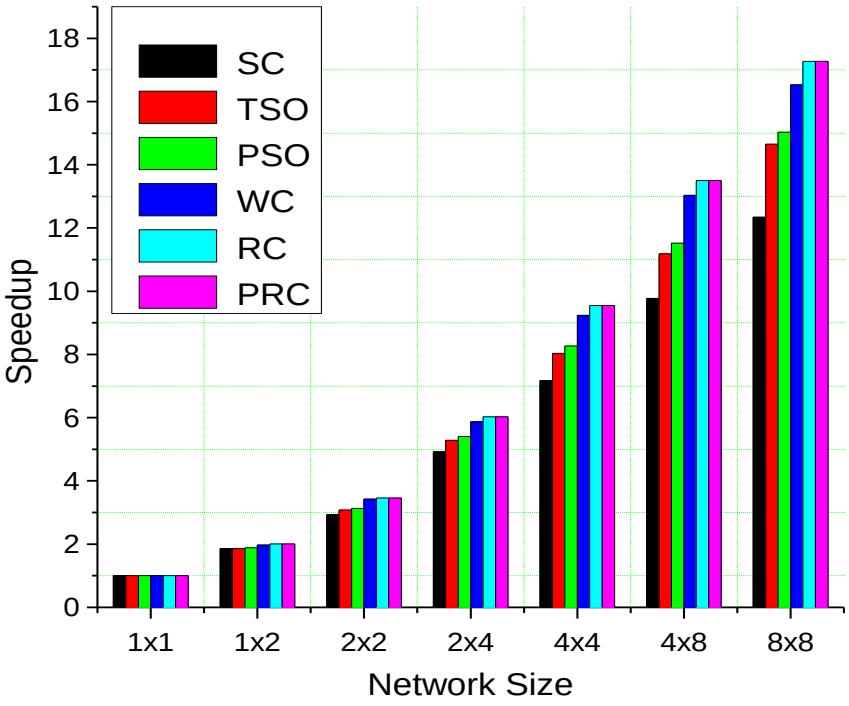



**Figure 4.28.** Speedup under bit count application.

The performance and speedup of six different memory consistency models are compared in Figures 4.27 & 4.28 under the bit count application. In contrast to the angle conversion application, the increase in the performance and speedup under the relaxed memory consistency models over the SC model is higher. This is due to low computation-to-communication ratio under the bit count application compared to the angle conversion application. The computation time per input data item under the angle conversion application is more (31 cycles) compared to that under the bit count application (21 cycles). The communication becomes significant under the bit count application due to less computation-to-communication ratio. Hence, the performance gap between the relaxed memory consistency models and SC model is even more compared to the angle conversion application. The communication overhead is efficiently handled under the PRC and RC models by allowing more outstanding data operations in the network which are pipelined and overlapped with respect to each other. The increase in the performance as observed in Figure 4.27 under the relaxed models over the SC model is 13.2% to 20.3% higher than that under the angle conversion application in the 64-cores system (Figure 4.25). Exactly, the same trend can be observed in the speedups of different memory consistency models as given in Figure 4.28.

### 4.3.2.3 Pattern Search

The pattern search application is also similar to that discussed before under the scalability analysis of the RC and SC models in the preceding section. It searches the data

patterns (*P)* against data elements (*D)*, which are initialized in the shared memory across the network. We have experimented with two different cases 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 computation process. *P64-D64:* For the 64 patterns and 64 data elements, one pattern and one data element are mapped on the 8x8 network and each node performs the computation. The calculated outputs are the number of times that the patterns appear in the data elements. The outputs are written in the shared memory of the local node.

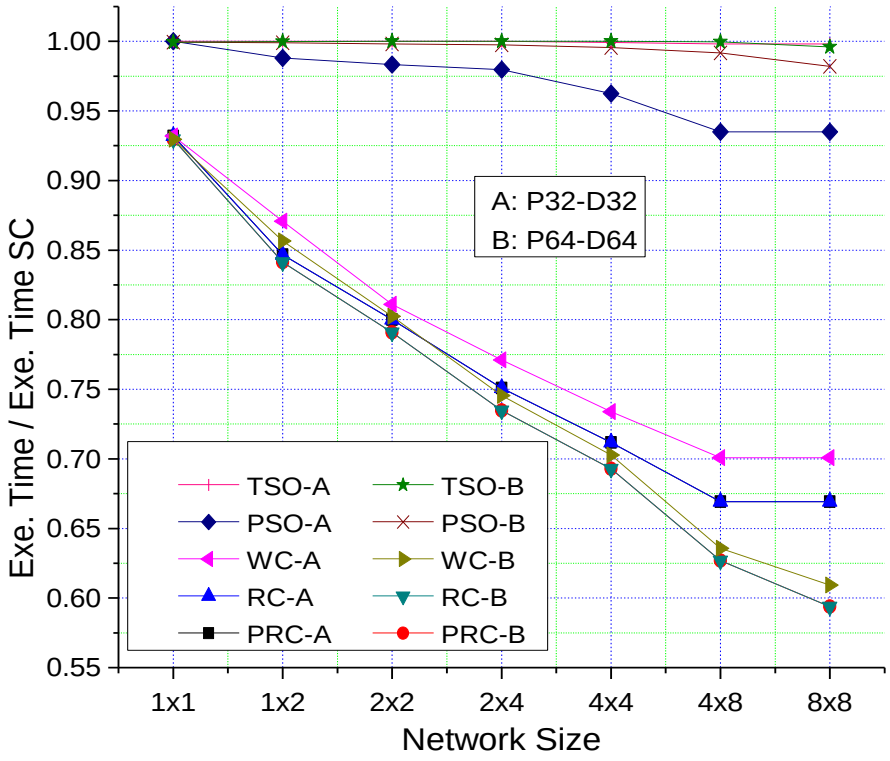



**Figure 4.29.** Pattern search: ratio of (Execution time of relaxed models / Execution time of SC).

The execution time of the relaxed memory consistency models *relative* to the SC model under the pattern search application is shown in Figure 4.29. The relative execution time under the relaxed memory consistency models decreases when the system is scaled up from 1 to 64-cores. The relative execution time under the PRC and RC models decreases more compared to the other stricter memory consistency models. It is due to the additional pipelining and overlapping among the shared memory operations which is permitted under these relaxed memory consistency models. The relative execution time under these memory consistency models at some point (e.g., network size) levels off. It is dependent on the chosen problem size and it's fitting to the platforms. For the (*P64-D64*) problem, the relative execution time under all these memory consistency models constantly decreases when the network size is increased. It is due to the fact that the problem size matches well to the increasing size of the network compared to the

(*P32-D32*) problem. Under the (*P64-D64*) problem, a better and scalable behavior can be observed, where the computation cost is further divided in the 8x8 network. It is because that all the processors in the system are involved in the computation process. Thus, a continuous performance improvement can be observed when the system size is scaled from 1 to 64-cores. The normalized execution time is not always 1.0 in the 1x1 network size under the relaxed memory consistency models. It is due to the fact that the virtual to physical address translation overhead associated with the local shared memory operations under these consistency models is reduced due to the overlapping of memory operations.

### 4.3.2.4 Matrix Multiplication

The matrix multiplication application calculates the product of two matrices (A, B) and a resultant matrix (C) is produced. All these input and output matrices are initialized in the distributed shared memory across the network. We have used two different sized matrices in the matrix multiplication. The A[32x1] cross B[1x32] resulting into C[32x32] matrix. The multiplication of A[64x1] and B[1x64] matrices produces C[64x64] matrix. The application is mapped on the increasing size of the network from 1 to 64-nodes. When A[32x1] and B[1x32] matrices are mapped on the 8x8 system, only 32-nodes are involved in the computations. When A[64x1] and B[1x64] matrices are mapped on the 8x8 network, all the nodes perform the computation. Each node operates on that part of the matrix-A which is on the same node and the entire matrix-B which is distributed across the network. Each node stores a part of the output resultant matrix in the shared memory of the local node.

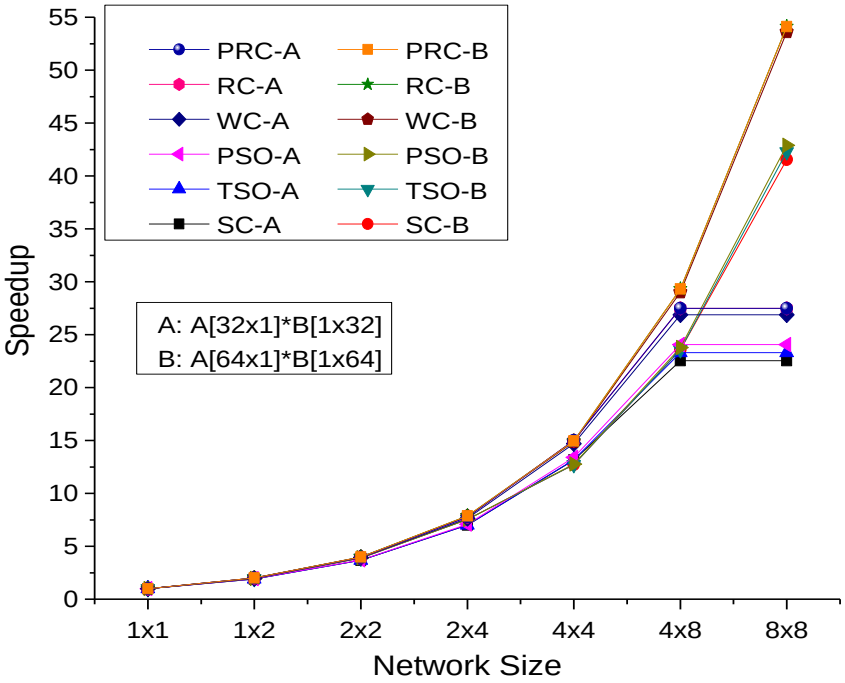



**Figure 4.30.** Speedup under matrix multiplication application.

The speedup of memory consistency models under the matrix multiplication application is shown in Figure 4.30. The speedup is lower under the case-A due to the smaller problem size compared to that under case-B. The speedup flattens off after 32-nodes up to 64-nodes under the case-A, because the potential parallelism in the 8x8 system is not exploited by the chosen problem size. Alternatively, it is utilized under the case-B, where the computation cost is further divided in the 8x8 system and all the nodes perform the computation. As a result, the computation cost is significantly reduced in the larger networks. The problem size like case-B scales well compared to the case-A. The relaxed memory consistency models offer more performance gain and benefits compared to the SC model by exploiting the higher level of parallelism as the network size is scaled up.

All these applications which are discussed above are the data intensive applications and do not use the synchronization primitives. Thus, the performance of the relaxed memory consistency models like: WC, RC, and PRC are almost similar, but note that, the performance under these memory consistency models are higher than the PSO, TSO and SC models due to the additional reordering and relaxation in the data operations. Next we consider the mapping of synchronization intensive applications, which uses both the data and synchronization operations. The performance gain among these relaxed memory consistency models can be observed under such applications which frequently use the synchronization operations. The wavefront computation applications are developed and mapped as the synchronization intensive applications. In the following, we compare the performance of different memory consistency models under these applications.

#### 4.3.2.5 Wavefront Computation-I (WFC-I)

The WFC- I application uses total (16x64) data computations which are performed in 16 different protected sections. The protected sections are *partially* overlapped with respect to each other. When WFC-I is mapped on a single core system, each data computation uses the data produced in the previous computation in the same protected section and also the data computed in the previous protected section. When WFC-I is mapped on the two-core system, each node performs half of the data computations (16x32). The number of protected sections remains the same. Only the number of data computations per core becomes half when the number of cores is doubled. The data computed by one node is then used in the computations on the next node. To ensure the data availability of the previous node, each node uses 16 additional locks acquires at the start of each protected section to the same lock. Similarly, this trend continues up to 64-cores system, where each core performs the (1x16) computations. The ratio of data-to-synchronization operations decreases when the network size is increased.

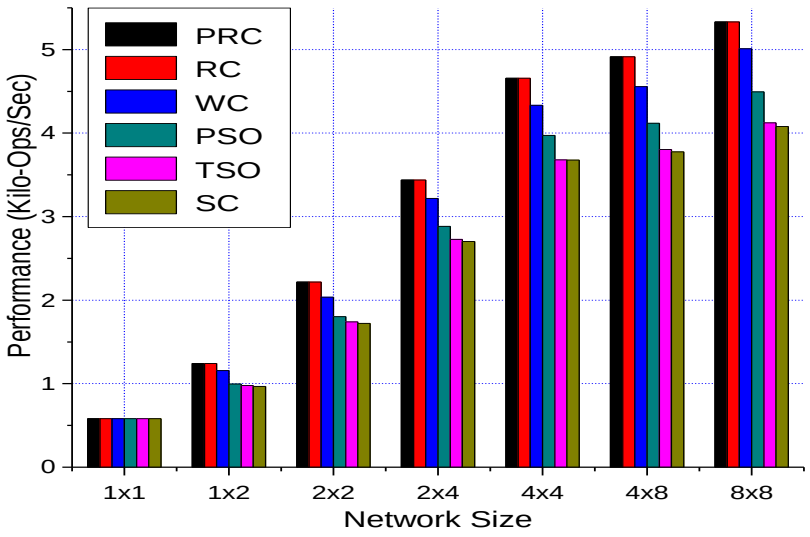



**Figure 4.31.** Performance under WFC-I application.

As observed in Figure 4.31, the performance gap between the relaxed memory consistency models like WC and RC/PRC is increased under the synchronization intensive WFC-I application. This is due to the fact that the WC model does not allow reordering and overlapping among the data and synchronization (acquire, release) operations. The RC and PRC models allow reordering and relaxation among the data and synchronization operations. As given in Figures 4.31 & 4.32, the performance and speedup of the memory consistency models quickly increase in the small networks up to 16 nodes, because of the high ratio of data-to-synchronization operations. Onward, the performance and speedup rise slowly due to the increasing synchronization overhead among the cores in the larger networks. The ratio of data/synchronization operations is decreased in the larger networks and the system performance is reduced due to the significant synchronization overhead.

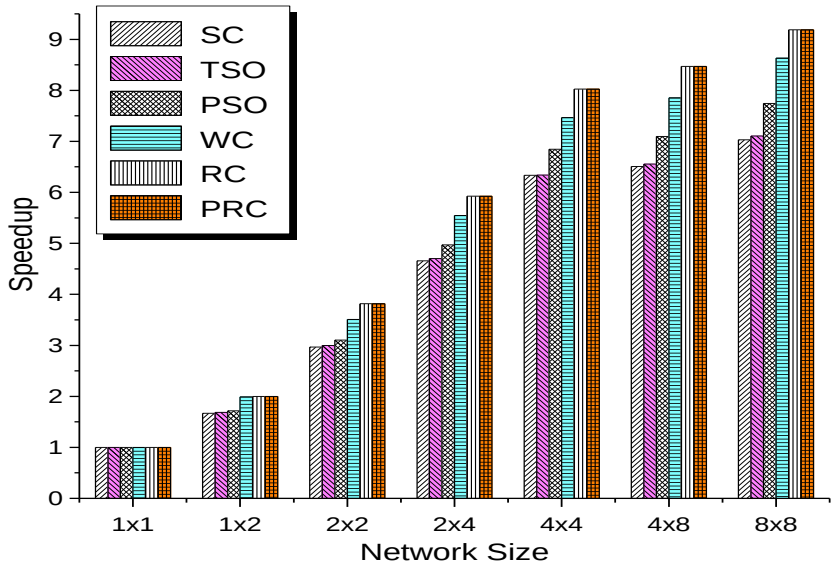



**Figure 4.32.** Speedup under WFC-I application.

#### 4.3.2.6 Wavefront Computation-II (WFC-II)

The WFC-II application uses total (32x64) data computations which are performed in (32x64) different *nested* protected sections. When the application is mapped on a single core system, each data computation uses the data produced in the previous *nested* protected sections. When the application is mapped on the two-core system, each node performs half of the total data computations (32x32) in the nested protected sections. Each computation uses the data computed under the previous nested protected section on the local (same) node and the data computed on the previous (remote) node. The data computed on the remote node are made available by using nested acquires to the common locks operations at the start of each protected section. Likewise, this trend continues up to 64-cores system, where each core performs (1x32) computations. The ratio of data-to-synchronization operations remains the same when the network size is increased. The number of protected sections which includes both the data and synchronization operations becomes half as the number of cores is doubled.

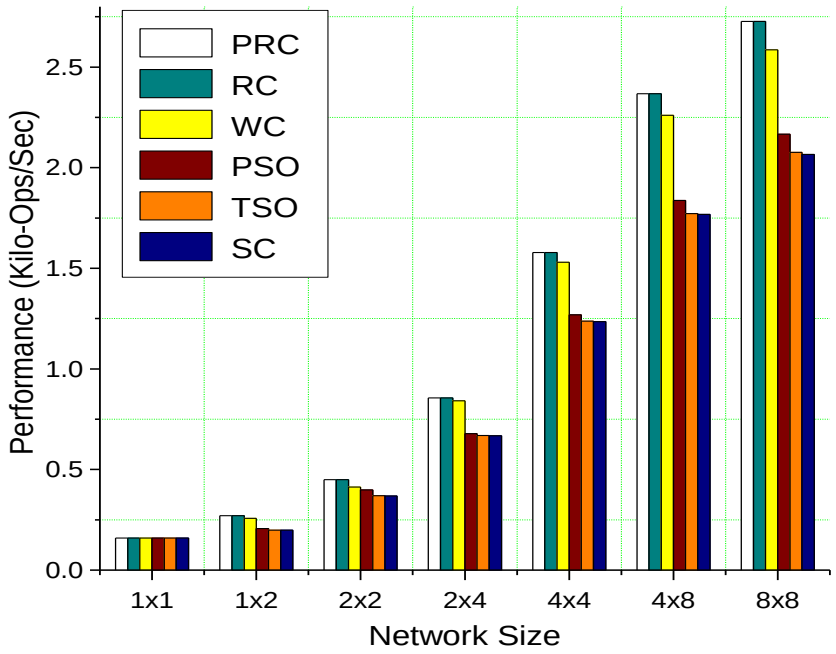



**Figure 4.33.** Performance under WFC-II application.

The performance and speedup of the memory consistency models under the WFC-II application are given in Figures 4.33 & 4.34, respectively. The increase in the performance and speedup under the relaxed models over the SC model are slightly higher compared to the WFC-I application. This is due to the increasing number of protected sections per node, which increase the amount of computation per node under the WFC-II application compared to the WFC-I application. Recall that, the WFC-II uses 32 protected sections per node, while WFC-I application uses 16 protected sections per node. The significant amount of computation is parallelized in the larger networks under the WFC-II

application. In addition, the synchronization overhead under the WFC-II application in contrast to WFC-I application does not arise in the larger networks. It is due to the fact that the ratio of data-to-synchronization operations remains the same under the WFC-II application, while it is decreased under the WFC-I application when the network size is increased. Thus, the performance and speedup of the memory consistency models increase quickly in the larger networks under the WFC-II application compared to the WFC-I application.

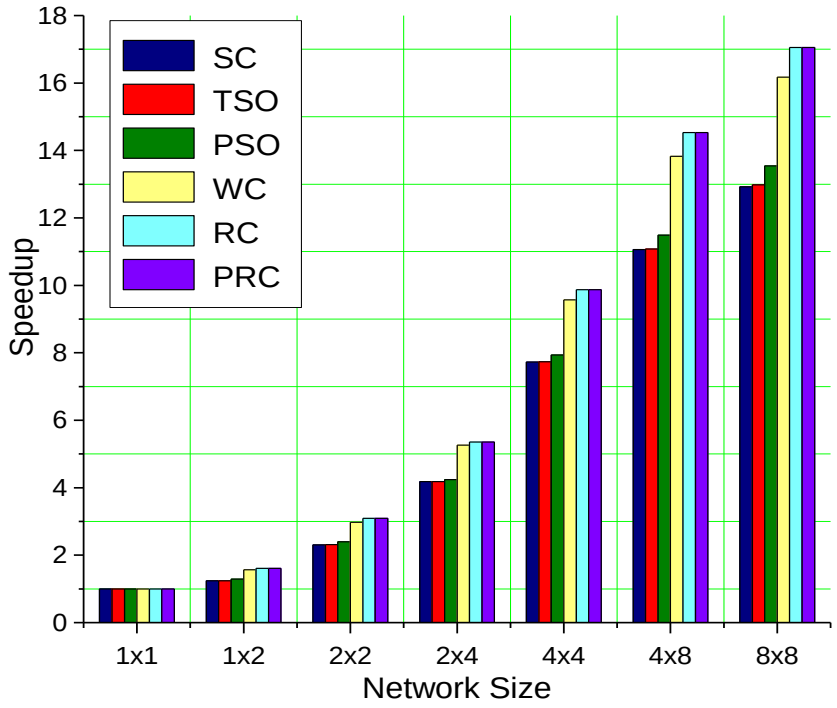



**Figure 4.34.** Speedup under WFC-II application.

Both these WFC-I and WFC-II applications do not use the unprotected data operations. Therefore, the performance gap between the PRC and RC models is less significant. In order to observe the performance gain of the PRC model over the RC model, the WFC-II application is supplemented with the *unprotected* data operations (WFC-II-UPD). We experimented in the 8x8 system. The number of integrated 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 (1x64) unprotected data computations. For the 64 protected sections per node, there are total (64x64) unprotected data computations together with (64x64) nested protected sections. All these data computations performed in the unprotected sections are independent of each other.

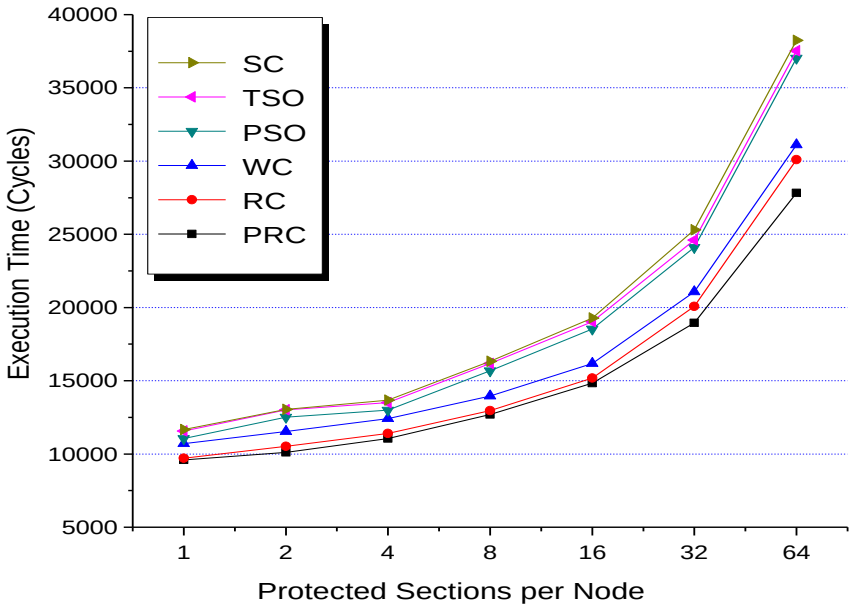



**Figure 4.35.** Execution time under WFC-II-UPD application.

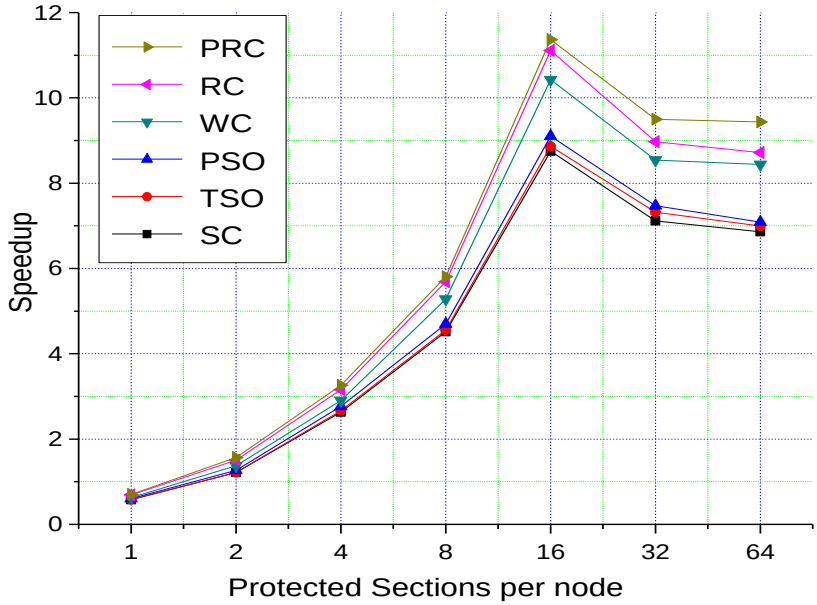



**Figure 4.36.** Speedup under WFC-II-UPD application.

As observed in Figures 4.35 & 4.36, the performance gain of the PRC model over RC model is clearly visible under the WFC-II-UPD application. This is due to the interleaving unprotected data operations which are integrated among the protected sections. As given in Figure 4.35, for the 64 protected sections per node, the execution time under the PRC model is reduced by 7.6% over the RC model and by 27.3% over the SC model. As shown in Figure 4.36, the speedup accelerates/increases until 16 protected

sections per node and then retards/decreases onward. This is due to the fact that the number of integrated unprotected data operations is proportional to the number of protected sections per node. As the number of protected sections per node increases, the amount of computation also increases due to the incorporation of unprotected data operations.

### 4.3.2.7 Average of all Applications

The average execution time under all these applications for the relaxed memory consistency models *relative* to the SC model in percentage are given in Table 4.2. 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.

**Table 4.2.** Average execution time relative to SC model in percentage

| Applications | SC | TSO | PSO | WC | RC | PRC |
|---|---|---|---|---|---|---|
| *Angle Conversion* | 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 Multiplication* | 100 | 97.74 | 95.77 | 78.23 | 77.13 | 77.13 |
| *WFC-I* | 100 | 98.93 | 90.77 | 81.41 | 76.52 | 76.52 |
| *WFC-II* | 100 | 99.54 | 95.41 | 79.92 | 75.79 | 75.79 |
| *WFC-II-UPD* | 100 | 98.11 | 96.8 | 81.32 | 78.71 | 72.73 |
| *Average* | 100 | 95.98 | 92.44 | 76.32 | 73.59 | 72.73 |

## 4.3.3 Summarizing the Scalability Analysis

To summarize, in all the above experiments, the execution time of the PRC and RC models are observed between 50% to 100% of the SC model. As discussed earlier, the specific numbers are highly sensitive to the application and depend on the problem size

that how well it matches to the platforms. The reduction in execution time under the relaxed memory consistency models compared to the strict memory consistency models also depends on the computation-to-communication ratio, traffic pattern and synchronization overhead in the system. However, the observed trends suggest that the relaxed memory consistency models like PRC and RC inherently scale better and perform well than the stricter memory consistency models when the network size is increased. As given in Figure 4.37, the execution time under the PRC and RC models relative to the SC model decreases more compared to the other memory consistency models when the network size is scaled up. The benefits of relaxed memory consistency models over the SC model increase when the network size is increased. But, sometimes the nature of the problem makes it harder to exploit the additional parallelism in the larger networks and the benefits of the PRC and RC models over the SC model saturate as well. However, problems that scale better, like the B-pattern search and B-matrix multiplication problems continue to collect more benefits from the higher level of parallelism that the RC and PRC models provide compared to the stricter memory consistency models. Therefore, 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.

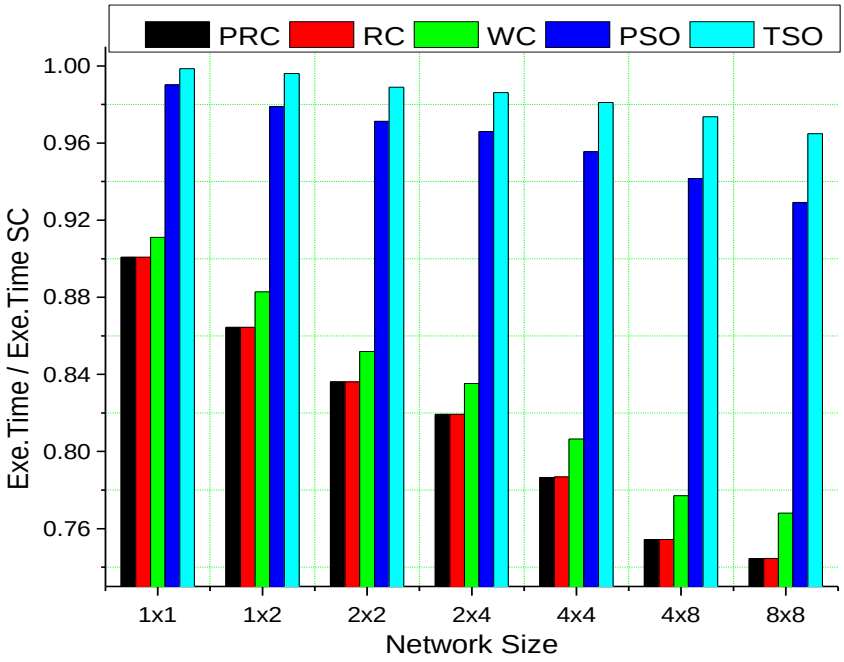



**Figure 4.37.** Average of all applications: ratio of (Execution time of relaxed models / Execution time of SC model).

From the experiments, it can also be observed that the reduction in the relative execution times under relaxed memory models like WC, RC, and PRC over the SC model is expected to be 3.33 percentage points approximately for each doubling of the core count up to 128 cores. The relaxed memory models could exploit the increasing parallelism in the larger networks beyond the 64-cores by reordering and pipelining the memory

operations. Similarly, for the TSO and PSO models Figure 4.37 suggests that this can be projected to 1.0 percentage points for each doubling of the core count up to 128 cores. In general, the performance gain of the relaxed memory consistency models can be even higher than the experimental values when the system size is further increased.

As discussed in earlier chapters, the main focus of our work is on the memory consistency issue and the memory consistency models are realized in the customized McNoC systems which are independent of the cache coherence protocols. However, the presence of caches could affect the observed results under these independent memory consistency models. It depends on several factors like cache write policy, cache hit and miss ratio and the exercised cache coherence protocols. Thus, it is very hard to predict the exact number by which the results would be scaled with the inclusion of data caches. Although, the results shown for the memory consistency models independent of the cache coherence protocols are still valuable and significant, because, 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 could be significantly decreased compared to the strict SC model. 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 memory consistency protocols.

## 4.4 Summary

This chapter has studied the scalability of different memory consistency models which are realized in the McNoC systems with 1 to 64-cores. The experimental setup and the platforms configuration parameters are discussed. For the experiments, we have developed the synthetic and application workloads. These workloads are used to assess and compare the performance of these different memory consistency models in the McNoC systems. Specifically, the scalability of various memory consistency models is analyzed by mapping these applications on the increasing size of the network. The performance metrics like *execution time*, *performance*, *speedup*, *overhead* and *efficiency* are evaluated and compared for the different memory consistency models when the network size is scaled up. The scaling behavior of these memory consistency models is analyzed by using different types of applications with different problem sizes. This study underlines some interesting and key factors which affect the performance and scalability of the relaxed memory consistency models over the stricter memory consistency models. For instance, the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio, problem sizes, and the types and natures of applications, etc., affect the performance improvement statistics of relaxed memory consistency models over the stricter memory consistency models.

# Chapter 5

# Summary and Future Work

This chapter summarizes the thesis and future directions are outlined.

## 5.1 Summary

There are general trends and developments in the area of parallel computation (multi-core), parallel communication (Network-on-Chip, NoC) and memory architectures (Distributed Shared Memory, DSM). These developments have led to some critical issues of memory consistency and cache coherence in the shared memory multi-core systems. The thesis discusses and elaborates on the *memory consistency* issue in the NoC-based DSM multi-core systems. The memory consistency issue arises due to the unconstrained memory operations which lead to the incorrect and unexpected behavior of the shared memory multi-core systems. The memory consistency *models* enforce the ordering constraints on the memory operations for the expected behavior of shared memory multi-core systems. We confine our study on the memory consistency issue by making it independent of the cache coherence issue. We have considered the memory consistency models like: SC, TSO, PSO, WC and RC in the customized McNoC systems. We have also proposed the PRC model as an extension of the well known RC model, which provides additional reordering and relaxation among the memory operations.

The main *goal* of the thesis is to realize different memory consistency models and their scalability is analyzed in the McNoC systems. The core contributions of this research work can be categorized into two groups: First, the novel *architecture support* is provided for these different memory consistency models in the McNoC systems. Second, the *scalability* of these memory consistency models is analyzed in the McNoC systems.

In chapter 2, we have described the background and motivation for the memory consistency issue. The cache coherence and memory consistency protocols are briefly described. Six different memory consistency models (e.g., SC, TSO, PSO, WC, RC, and PRC) are discussed and the ordering constraints under these memory models are analyzed using different program segments with some easy and understandable examples. A comprehensive analysis of the ordering constraints under the WC, RC, and PRC models is also presented using different program segments with non-overlapped, nested and partially overlapped protected sections. The requirements for the ordering constraints on the outstanding operations which are issued by a processor to the same memory location under these relaxed memory consistency models are also discussed. The relaxations offered under these different memory consistency models are summarized. The memory

models which are specified at the high level programming languages are also described. The compiler optimizations and their influence on the memory consistency models are discussed. We have provided an inclusive review of the related work on the memory consistency both in the general purpose multi-processors and customized McNoC systems.

In chapter 3, we propose the architectural support of these six different memory consistency models in the McNoC systems. We have developed the configurable McNoC platforms which support these different memory consistency models. Each platform uses *Nostrum* NoC as a communication infrastructure with the distributed shared memories, distributed locks and customized interfaces for the processors. The memory consistency models/protocols are implemented at the customized interface which integrates a processor with the rest of the system. These memory consistency models are realized in the McNoC systems by enforcing the required global orders on the memory operations by *stalling* the processor and by using the *transaction counter* and *address stack*-based novel approaches. At the end of chapter, impact of the NoC features (e.g., traffic patterns, routing algorithms, injection rate, and network congestion), non-blocking caches, pre-fetching and transactional memories on the performance of these memory consistency models is also discussed.

According to the realization scheme of the SC model, *program order* is enforced by stalling the processor on the issuance of each memory operation till its completion. A processor in each node of the network issues the next operation on the completion of a previously issued operation in the program. The memory operations are executed as per program order. The *sequential order* is enforced by sequentially accessing the critical memory locations among the multi-core in the system.

The TSO model is realized by enforcing the required global orders on the memory operations by stalling the processor and by using the *WTC* and *WA-Stack* structures in each node of the network. The *WTC* keeps track of the outstanding write operations which are issued by a processor in the system. The *WTC* is used to enforce the ordering constraints on the memory operation in the case of *a write followed by a write* operation. The global orders under the TSO model in the cases of *a read followed by a read* operation and *a read followed by a write* operation are enforced by stalling the processor at the issuance of a read operation till the return of data. The ordering constraints with respect to the synchronization operations are also enforced. The *WA-Stack* keeps track of the addresses of outstanding write operations which are issued by a processor in the system. The *WA-Stack* constrains the outstanding operations which are issued by a processor to the same location in the memory for the purpose of correctness.

The PSO model is also realized by enforcing the required global orders on the memory operations by stalling the processor and by using the *WTC* and *WA-Stack* structures in each node of the network. In contrast to the TSO model, the *WTC* is not checked at the issuance of each write operation and the issuance of a write operation is not delayed till the completion of a previously issued outstanding write operation. The PSO model compared to the TSO model allows reordering and relaxation among the write operations which are issued by a processor in the system. As a result of additional relaxation under the PSO model, the *WA-Stack* is also checked at the issuance of each write operation. Hence, the outstanding write operations which are issued by a processor to the same memory location

are constrained to complete in the order specified by the program. The rest of the realization scheme of the PSO model is similar to that described under the TSO model.

The WC model is realized by stalling the processor and by using a *TC* and an *A-Stack* in each node of the network. A *TC* is used to keep track of the outstanding data (read, write) operations which are issued by a processor in between the two consecutive synchronization points. The *TC* is checked at each synchronization point and the issuance of a synchronization operation is delayed by stalling the processor till the completion of previously issued outstanding data operations. This enforces the global order under the WC model in the case of a data operation followed by a synchronization operation. The processor is stalled on the issuance of a synchronization operation till its completion. This enforces the required global orders under the WC model in the cases of a synchronization operation followed by a data operation and a synchronization operation followed by a synchronization operation. The global orders among the synchronization operations are also enforced by sequentially accessing the synchronization variables among the multiple processors in the system. The *A-Stack* is used to keep track of the addresses of the outstanding data operations which are issued by a processor in the system. The *A-Stack* constrains the outstanding data operations which are issued by a processor to the same memory location. The outstanding data operations issued by a processor to the same location in the memory are executed as per *program order* for the purpose of correctness. To sum up, the global orders required under the WC model are enforced by *stalling* the processor and by using the *TC* and *A-Stack* hardware structures in each node of the network.

For the realization of RC model, the required global orders are enforced by stalling the processor and by using the *TC* and *A-Stack* hardware structures in each node of the network. A *TC* is used to keep track of the outstanding data operations which issued by a processor in between the two consecutive release points. The *TC* keeps track of the data operations which are issued both in the non-critical and critical sections. The *TC* is checked at each release point and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding data operations both in the non-critical and critical sections. This enforces the global order under the RC model in the case of a data operation followed by a release operation. The processor is stalled on the issuance of an acquire operation till its completion. The *A-Stack* is used to keep track of the addresses of outstanding data operations which are issued by a processor in the system. The *A-Stack* works in the same way as described under the WC model.

The realization scheme of the PRC model uses an *Acquire Counter* (*AC*), a *Transaction Counter for protected data* ($TC_{PD}$) and an *A-Stack* in each node of the network. The *AC* is used for the classification of data operations as unprotected and protected operations. The protected data operations are protected under a lock acquire and release operation, while the rest are the unprotected data operations. A $TC_{PD}$ is used to keep track of the outstanding *protected* data (read, write) operations which are issued by a processor before the release operation. The $TC_{PD}$ is checked at the issuance of each release operation and the issuance of a release operation is delayed by stalling the processor till the completion of previously issued outstanding protected data operations, which is indicated by the zero value of $TC_{PD}$. This enforces the global order under the PRC model in the case of a protected data operation followed by a release operation. The issuance of a release operation is not delayed for the completion of previously issued unprotected data

operations. The processor is stalled on the issuance of an acquire operation till its successful completion. This enforces the required global orders under the PRC model in the case of an acquire operation followed by a protected data operation and an acquire operation followed by a release operation. The global orders in the case of a release operation followed by an acquire operation is enforced by sequentially accessing the synchronization variables among the multi-core in the system. The *A-Stack* is used to keep track of the addresses of outstanding data operations which are issued by a processor for the purpose of correctness.

In chapter 4, we have presented the performance and scalability analysis of different memory consistency models in the McNoC systems. The experimental framework is described. We have developed the synthetic and application workloads which are mapped on the increasing size of the network. The applications are selected from different domains. The key performance metrics like *execution time, performance, speedup, overhead* and *efficiency* are evaluated as a function of the network size. The scalability analysis is performed in the McNoC systems using 1 to 64-cores. The experiments are planned and conducted to study the scalability of these different memory consistency models in the McNoC systems. We have analyzed the scalability of these memory models by mapping different application workloads on the various sized networks using different problem sizes. The scalability is also studied on the basis of different application types and from the system design perspectives. The scalability analysis of the RC and SC models is conducted both with the synthetic and application workloads. We have also provided the scalability analysis of six different memory consistency models (e.g., SC, TSO, PSO, WC, RC, and PRC) in the McNoC systems. The scalability study is conducted by developing and mapping new application workloads on the increasing size of the network. In all the experiments, the execution time of the PRC and RC models are observed between 50% to 100% of the SC model. The specific numbers are sensitive and dependant on several factors. For instance, the performance gain of the PRC and RC models over the stricter models improves with the network size up to the point where the given application matches to the available parallelism. Beyond this point, the improvements flatten or drop. The study shows the dependence of performance improvement on the computation-to-communication ratio, traffic patterns, data-to-synchronization ratio and the problem size. However, the observed trends suggest that the relaxed models like PRC and RC inherently scale better and perform better than the stricter memory consistency models as the network size is increased. The execution time under the PRC and RC models relative to the SC model decreases more compared to the other memory consistency models when the network size grows up. The benefits of the PRC and RC models over the SC model depend upon the fact that how well the additional relaxation under these relaxed memory models is utilized. Some problems scale better than the others and the benefits of the PRC and RC over the SC model is continuously exploited. The performance gain of the PRC and RC models over SC model can be higher than 50% of the observed gain by further increasing the size of the network. The presence of caches could also affect the performance of these memory consistency models. The relaxed memory models compared to the strict memory models could perform better by overlapping and pipelining the extra coherence traffic under the directory-based coherence protocols. Therefore, the same relative execution time and trends are expected under the relaxed memory consistency models by using the cache coherence protocols as well.

## 5.2 Future Directions

The memory consistency is a critical issue in the shared memory multi-core systems. This is an active research area with some good contributions in the past two decades. Since the memory consistency is a complex issue at the abstract level in the shared memory multi-core systems, therefore, the nature of topic demands the exploration of some relevant and alternative topics in the future. In the following, we direct towards some of the related issues that require more emphases and considerations:

- *Coupling Consistency and Coherence:* The cache coherence and memory consistency are two different issues of a similar nature in the DSM systems. The cache coherence achieves the consistency at the cache line/block level, while the memory consistency targets the consistency of entire shared memory. These two problems arise due to the incorporation of different optimizations in the multi-core systems. The shared memory multi-core systems using data caches confront both these challenges of *coherence* and *consistency* issues. To address the cache coherence and memory consistency issues simultaneously by using a cache block as the atom of the consistency requires further consideration. Some solutions [13][32][69] with such approaches in the general multi-processors DSM systems are elegant, but suffer from *false sharing*, the massive communication overhead due to *extra coherence traffic* and *scalability* issues. On the other hand, we have proposed an orthogonal approach by decoupling the memory consistency and the cache coherence issues for the hard real time applications, customized NoC-based systems, embedded architectures, application specific systems and some other applications. Our independent solution of the memory consistency issue does not mean to reject the caches and cache coherence protocols at all. The cache coherence protocols could still be accommodated along with these independent memory consistency models to get the benefits of the data caches. The directory-based cache coherence protocol could be implemented on top of these independent memory models. This approach will allow for the independent optimal selection of the cache block size to optimize the coherence traffic, energy/power consumption and the directory overheads. In such situations, the issuance and completion of the data transactions should be redefined to be tracked by the *transaction counter* and *address stack* which are maintained in the processor interface. The cache coherence traffic could be handled efficiently by developing more flexible, configurable and sophisticated processor interfaces.

- *Parallel Programming Model:* The parallel programming is also a challenging issue in the parallel computing (multi-core) systems. *Automatic* parallelization, mapping and execution of the parallel programs on the NoC-based multi-core platforms are also interesting and relevant research fields to be explored. This requires the support of software tools or operating systems that run on the software platform above the hardware platform. These software layers between

the end users and the hardware platform could provide various services regarding the memory management, I/O handling, process scheduling, process migration, system monitoring, and inter-process communication. It can also provide the programming models and automatic and efficient mapping of the parallel programs on the hardware platforms. For instance, the IMEC [77] has developed some tools like ATOMIUM/MPA (Multi-Processor Assist) and ATOMIUM/MH (Memory Hierarchy) to effectively handle these issues of parallelization and mapping in the multimedia applications. An interesting research direction would be to stack such software platforms on top of the hardware McNoC platform which could support both the relaxed memory consistency model and cache coherence protocol.

- *Transactional Memories:* Transactional memory is an appealing programming discipline. It targets to scale the programmer productivity by moving the synchronization burden from the user to the hardware or/and software platform support. The operations in the critical section are treated as one transaction. For the execution of a transaction, ACID (Atomicity, Consistency, Isolation and Durability) properties are ensured. Transactional memory uses the concept of database transactions. It takes advantage of the optimistic concurrency that exists within the critical section/transaction compared to the systems which use the lock-based synchronization. Independent transactions from different threads/processors operate in parallel on disjoint shared data structures. If there is no conflict among the transactions which are issued by different threads/processors then they are *committed* (successfully completed). When there is a conflict among these transactions of different threads, then they are *aborted* and executed again for the consistent behavior of the system. The Hardware Transactional Memories (HTM) [38] rely on the additional transactional caches and the required/associated coherence protocols. The restriction of this approach is that the transaction size is bounded by the set size of the transactional caches. On the other hand, the Software Transactional Memories (STM) [56] have no such restrictions. The STM relies on the runtime data structures, but is less efficient compared to the HTM. A hybrid approach (HyTM) [39] combines the benefits of both the HTM and STM. Intel has recently announced that they will support transactional memory in their upcoming processor code named Haswell [78]. The Haswell will support the HTM and expected to be released soon.

To be precise, the research on the memory consistency and cache coherence in the shared memory systems has a huge potential to expand further. These critical issues need to be addressed for the correct behavior of the future multi-core systems, i.e., application-specific McNoC systems as well as for the general-purpose multi-processor systems. These topics require multi-disciplinary research which is essential to drive and bring some innovations.

# Bibliography


[1] A. Jantsch, X. Chen, A. Naeem, Y. Zhang, S. Penolazzi, and Z. Lu. Memory Architecture and Management in an NoC Platform. In Axel Jantsch and D. Soudris, editors, *Scalable Multicore Architectures: Design Methodologies and Tools*, Eds. Berlin, Germany: Springer, 2011, pages 3 – 28.

[2] M. Horowitz, W. Dally. How scaling will change processor architecture. *In Proceedings of International Solid-State Circuits Conference (ISSCC'04)*, Digest of Technical Papers, pages 132–133, February 2004.

[3] S. Borkar. Thousand core chips: A technology perspective. *In Proceedings of 44th Design Automation Conference (DAC'07)*, pages 746–749, June 2007.

[4] S. Vangal, J. Howard, G. Ruhl. An 80-tile 1.28tflops network-on-chip in 65nm cmos. *In Proceedings of International Solid-State Circuits Conference (ISSCC'07)*, Digest of Technical Papers, pages 98–100, February 2007.

[5] A. Jantsch and H. Tenhunen, editors. *Networks on Chip*. Kluwer Academic Publishers, 2003.

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

[7] T. Bjerregaard and S. Mahadevan. A survey of research and practices of network-on-chip. *ACM Computing Surveys*, vol. 38, no. 1, pages 1–51, March 2006.

[8] J. D. Owens and W. J. Dally. Research challenges for on-chip interconnection networks. *IEEE MICRO*, vol. 27, no. 5, pages 96–108, October 2007.

[9] L. Lamport. How to Make a Multiprocessors Computer That Correctly Executes Multiprocessor Programs. *IEEE Transaction on Computers*, Vol. C-28. No. 9, pages 690-691, September 1979.

[10] S. V. Adve and K. Gharachorloo. *Shared Memory Consistency Models: A Tutorial*. Digital Western Research Laboratory, report no. 95/7, USA, 1995.

[11] D. E. Culler, J. Pal Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers, 1999.

[12] M. Dubois, C. Scheurich and F. Briggs. Memory access buffering in multiprocessors. *In Proceedings of Annual International Symposium on Computer Architecture (ISCA'86)*, 1986.

[13] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. *Computer Architecture News*, vol. 18, no. 2, pages 15-26, June 1990.

[14] D. J. Sorin, M. D. Hill, and D. A. Wood. *A Primer on Memory Consistency and Cache Coherence*. Morgan & Claypool Publishers, 2011.

[15] M. Hill, S. Eggers, J. Larus, G. Taylor, G. Adams, B. Bose, G. Gibson, P. Hansen, J. Keller, S. Kong, C. Lee, D. Lee, J. Pendleton, S. Ritchie, D. Wood, B. Zorn, P. Hilfinger, D. Hodges, R. Katz, J. Ousterhout, and D. Patterson. Design decisions in spur: A vlsi multiprocessor workstation. *IEEE Computer*, vol. 19, no. 11, November 1986.

[16] L. M. Censier and P. Feautrier. A new solution to coherence problems in multi cache systems. *IEEE Transactions on Computer*, c- 27(12):1112–1118, November 1978.

[17] A. Naeem, X. Chen, Z. Lu, and A. Jantsch. Scalability of Weak Consistency in NoC based Multicore Architectures. *In Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS 2010)*, pages 3497 – 3500, Paris, France, June 2010.

[18] 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, pages 8 – 15, April 2010.

[19] A. Naeem, X. Chen, Z. Lu, and A. Jantsch. Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems. *In Proceedings of 16th ACM/IEEE Asia and South Pacific Design Automation Conference (ASPDAC 2011)*, pages 154 – 159, Yokohama, Japan, January 2011.

[20] A. Naeem, A. Jantsch, X. Chen, and Z. Lu. Realization and Scalability of Release and Protected Release Consistency Models in NoC based Systems. *In Proceedings of 14th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2011)*, pages 47 – 54, Oulu, Finland, August-September 2011.

[21] A. Naeem, A. Jantsch, and Z. Lu. Architecture Support and Comparison of Three Memory Consistency Models in NoC based Systems. *In Proceedings of 15th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools (DSD 2012)*, pages 304 – 311, Cesme, Izmir, Turkey, September 2012.

[22] A. Naeem, A. Jantsch, and Z. Lu. Scalability Analysis of Release and Sequential Consistency Models in NoC based Multicore Systems. *In Proceedings of IEEE International Symposium on System-on-Chip (SOC 2012)*, pages 1 – 7, Tampere, Finland, October 2012.

[23] A. Naeem, A. Jantsch, and Z. Lu. Scalability Analysis of Memory Consistency Models in NoC-based Distributed Shared Memory SoCs. *IEEE Transactions on Computer Aided Designs,* 2013, accepted for publication. doi: 10.1109/TCAD.2012.2235914

[24] A. Naeem. Shared Memory Consistency Models Evaluation in NoC based Multicore Systems. *In Proceedings of PhD Forum at Design, Automation & Test in Europe Conference (DATE'12)*, Dresden, Germany, March 2012.

[25] A. Naeem, A. Jantsch, and Z. Lu. Scalability and Performance Evaluation of Memory Consistency Models in NoC based Multicore SoCs. *Poster in ICES 5th Annual Conference: World-wide Trends and Challenges in Embedded Systems (ICES 2012)*, Stockholm, Sweden, August 2012.

[26] Sun Microsystems Inc. *The SPARC Architecture Manual*. Version 8, No. 800-199-12, January 1991.

[27] 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. *Communications of the ACM*, 2010.

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

[29] P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy Consistency for Software Distributed Shared Memory. *In Proceedings of the 19th Annual Symposium on Computer Architecture (ISCA'92)*, pages 13–21, May 1992.

[30] B. N. Bershad, M. J. Zekauskas, and W. A. Sawdon. The Midway Distributed Shared Memory System. *In Proceedings of IEEE Computer Society International Conference (COMPCON '93)*, pages 528–537, February 1993.

[31] L. Iftode, J. P. Singh, and K. Li. Scope Consistency: A Bridge between Release Consistency and Entry Consistency. *In Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'96*), Padua, Italy, June 1996.

[32] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. S. Lam. The Stanford Dash Multiprocessor. *IEEE Computer*, vol. 25, no. 3, pages 63- 79, March 1992.

[33] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos. A Tagless Coherence Directory. *In Proceedings of 42st International Symposium on Microarchitecture (MICRO '09)*, New York, NY, 2009.

[34] M. Ferdman, P. L. Kamran, K. Balet, and B. Falsafi. Cuckoo directory: A scalable directory for many-core systems. *In Proceedings of 17th International Symposium*

*on High Performance Computer Architecture (HPCA-17*), San Antonio, Texas, USA, February 2011.

[35] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token Coherence: Decoupling Performance and Correctness. *In Proceedings of 30th Annual International Symposium on Computer Architecture (ISCA '03),* ACM Press, pages 182-193, 2003.

[36] B. F. Romanescu, A. R. Lebeck, and D. J. Sorin. Address Translation Aware Memory Consistency. *IEEE Micro,* vol. 31, no.1, 2011.

[37] Arvind and J.-W. Maessen. Memory consistency model = Instruction Reordering + Store Atomicity. *In proceedings of 33rd International Symposium on Computer Architecture (ISCA '06),* pages 29-40, 2006.

[38] M. Herlihy and J. E. B. Moss. Transactional memory: Architectural support for lock-free data structures. *In Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA'93),* pages 289–300, San Diego, CA, 1993.

[39] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir, and D. Nussbaum. Hybrid Transactional Memory. *In Proceedings of the 12th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06),* California, October 2006.

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

[41] F. Petrot, A. Greiner, and P. Gomez. On cache coherence and memory consistency issues in NoC based shared memory multiprocessor SoC architectures. *In Proceedings of 9th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD 2006),* pages 53-60, Croatia, 2006.

[42] A. Hansson and K. Goossens. An On-Chip Interconnect and Protocol Stack for Multiple Communication Paradigms and Programming Models. *In Proceedings of International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'09),* France, 2009.

[43] J. W. van den Brand, and M. Bekooij. Streaming consistency: a model for efficient MPSoC design. *In Proceedings of 10th Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD 2007),* pages 27-34, Luebeck, Germany, August 2007.

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

[45] OCP International Partnership. (2007). *OCP Specification v2.2* [Online]. Available: http://www.ocpip.org/get_the_specifications.php

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

[47] Aeroflex Gaisler. *Leon3 32-Bit processor core* [Online]. Available: http://www.gaisler.com

[48] C. Grecu, A. Ivanov, A. Jantsch, P. P. Pande, E. Salminen, U. Y. Ogras, R. Marculescu. Towards Open Network-on-Chip Benchmarks. *In Proceedings of first ACM/IEEE International Symposium on Networks-on-Chip (NOCS'07)*, Princeton, New Jersey, USA, May 2007.

[49] Z. 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.

[50] B. Mathewson. The Evolution of SOC Interconnect and How NOC Fits Within It. *In Proceedings of 47th Design Automation Conference (DAC'10)*, pages 312-313, 2010.

[51] D. Mosberger. Memory Consistency Models. *ACM SIGOPS Operating Systems Review*, vol. 27, no. 1, USA, January 1993.

[52] F. Corella, J. M. Stone, and C. M. Barton. A formal specification of the PowerPC shared memory architecture. Technical Report Computer Science Technical Report RC 18638(81566), *IBM Research Division*, T. J. Watson Research Center, January 1993.

[53] C. May, E. Silha, R. Simpson, and H. Warren, editors. *The PowerPC Architecture: A Specification for a New Family of RISC Processors*. Morgan Kaufmann Publishers, Inc., 1994.

[54] R. L. Sites, editor. *Alpha Architecture Reference Manual*. Digital Press, 1992.

[55] H. Zhao, A. Shriraman, S. Dwarkadas. SPACE: sharing pattern-based directory coherence for multicore scalability. *In Proceedings of 19th international conference on Parallel architectures and compilation techniques (PACT 2010)*, pages 135-146, Vienna, Austria, September 2010.

[56] N. Shavit, and D. Touitou. Software Transactional Memory. *In Proceedings of 12th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, Ottawa, Canada, pages 204-213, 1995.

[57] D. C. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, P. M. Harvey, P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. L. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. Overview of the

architecture, circuit design, and physical implementation of a first-generation cell processor. *IEEE Journal of Solid-State Circuits,* vol. 41, no. 1, pages 179–196, 2006.

[58] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook. TILE64TM processor: A 64-core SoC with Mesh Interconnect. Digest of Technical Papers, *In Proceedings of IEEE International Solid-State Circuits Conference (ISSCC 2008),* vol. 51, pages 88–598, February 2008.

[59] B. Stackhouse, B. Cherkauer, M. Gowan, P. Gronowski, and C. Lyles. A 65-nm 2-billion-transistor quad-core Itanium processor. Digest of Technical Papers. *In Proceedings of IEEE International Solid-State Circuits Conference (ISSCC 2008),* vol.51, pages 92–598, February 2008.

[60] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many core x86 architecture for visual computing. *ACM Transactions on Graphics (TOG),* vol. 27, no. 3, Article 18, August 2008.

[61] W. J. Dally and B. Towles. Route packets not wires: on-chip interconnectoin networks. *In Proceedings of the 38th Design Automation Conference (DAC'01),* pages 684–689, New York, NY, USA, 2001.

[62] K. Gharachorloo. Memory Consistency Models for Shared-Memory Multiprocessors. *PhD thesis*, Stanford University, December 1995.

[63] S. V. Adve and M. D. Hill. Weak ordering - A new definition. *In Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA'90),* pages 2–14, May 1990.

[64] VSI Alliance. (2000). *Virtual Component Interface v2* [Online]. Available: http://www.vsi.org

[65] IBM Corporation. *CoreConnect bus architecture - A 32-, 64-, 128-bit core on-chip bus structure* [Online]. Available: http://www-03.ibm.com/chips/products/coreconnect.

[66] Philips Semiconductors. (2002). *Device Transaction Level (DTL) protocol specification v2.2*.

[67] J. L. Hennesy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, San Francisco, 2nd edition, 2003.

[68] P. S. Sindhu, J.-M. Frailong, and M. Cekleov. *Formal specification of memory consistency models*. Technical Report (PARC) CSL-91-11, Xerox Corporation, Palo Alto Research Center, December 1991.

[69] K. Gharachorloo, A. Gupta, and J. Hennessy. *Revision to Memory consistency and event ordering in scalable shared-memory multiprocessors*. Technical Report CSL-TR-93-568, Stanford University, April 1993.

[70] D. Shasha and M. Snir. Efficient and correct execution of parallel programs that share memory. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 10, no. 2, pages 282–312, April 1988.

[71] IBM Corporation. *IBM System/370 Principles of Operation*. May 1983. publication number GA22-7000-9, file number S370-01.

[72] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. *In Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA'90)*, pages 15–26, May 1990.

[73] D. L. Weaver and T. Germond, editors. *The SPARC Architecture Manual*. Prentice Hall, 1994. SPARC International, version 9.

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

[75] S. S. Iyer. Three Dimensional integration-memory applications. *In Proceedings of IEEE International Silicon-on-Insulator (SOI) Conference*, Page(s): 1-5, CA, USA, October 2009.

[76] X. Chen, Z. Lu, A. Jantsch and S. Chen. Supporting Distributed Shared Memory on Multi-core Network-on-Chips Using a Dual Microcoded Controller. *In Proceedings of Design, Automation & Test in Europe Conference (DATE'10)*, Dresden, Germany, March 2010.

[77] IMEC Corporation. *MPSoC Design* [Online]. Available: http://www2.imec.be/be_en/collaboration/ip-licensing-service/mpsoc-design/parallelization-and-memory-hiera.html

[78] Intel Corporation. *Transactional memory going mainstream with Intel Haswell* [Online]. Available: http://arstechnica.com/business/2012/02/transactional-memory-going-mainstream-with-intel-haswell/

[79] K. C. Yeager. *The MIPS R10000 Superscalar Microprocessor*. IEEE Micro, 16(2):28–40, April. 1996. doi:10.1109/40.491460

[80] MIPS Technologies, Inc., *MIPS R10000 Microprocessor User's Manual*, version 2.0, 1996.

[81] A. Meixner and D. J. Sorin. Dynamic Verification of Memory Consistency in Cache-Coherent Multithreaded Computer Architectures. *In Proceedings of the*

*International Conference on Dependable Systems and Networks*, pages 73–82, June 2006. doi:10.1109/DSN.2006.29

[82] A. Meixner and D. J. Sorin. Dynamic Verification of Memory Consistency in Cache-Coherent Multithreaded Computer Architectures. *IEEE Transactions on Dependable and Secure Computing*, vol. 6, no. 1, pages 282–312, 2009.

[83] K. Gharachorloo, A. Gupta, and J. Hennessy. Two Techniques to Enhance the Performance of Memory Consistency Models. *In Proceedings of the International Conference on Parallel Processing*, volume I, pages 355–64, August 1991.

[84] C. Blundell, M. M. K. Martin, and T. F. Wenisch. InvisiFence: Performance-Transparent Memory Ordering in Conventional Multiprocessors. *In Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*, June 2009.

[85] L. Ceze, J. Tuck, P. Montesinos, and J. Torrellas. *BulkSC:* Bulk Enforcement of Sequential Consistency. *In Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*, June 2007.

[86] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. OlukotunL. Transactional Memory Coherence and Consistency. *In Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA '04)*, June 2004.

[87] T. F. Wenisch, A. Ailamaki, B. Falsafi and A. Moshovos, Mechanisms for Store-wait free Multiprocessors. *In Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*, San Diego, CA, June 2007.

[88] C. Gniady, B. Falsafi, and T. Vijaykumar. Is SC + ILP = RC?. *In Proceedings of the 26th Annual International Symposium on Computer Architecture (ISCA'99)*, pages 162–71, May 1999.

[89] P. Ranganathan, V. S. Pai, and S. V. Adve. Using Speculative Retirement and Larger Instruction Windows to Narrow the Performance Gap between Memory Consistency Models. *In Proceedings of the Nineth ACM Symposium on Parallel Algorithms and Architectures*, pages 199–210, June 1997. doi:10.1145/258492.258512

[90] S. V. Adve. *Designing Memory Consistency Models for Shared-Memory Multiprocessors*. PhD thesis, Computer Sciences Department, University of Wisconsin–Madison, November 1993.

[91] J. R. Goodman. *Cache Consistency and Sequential Consistency*. Technical Report 1006, Computer Sciences Department, University of Wisconsin–Madison, February 1991.

[92] W. W. Collier. *Reasoning about Parallel Architectures*. Prentice-Hall, corporation. 1990.

[93] S. Sarkar, P. Sewell, F. Z. Nardelli, S. Owens, T. Ridge, T. Braibant, M. O. Myreen, and J. Alglave. The Semantics of x86-CC Multiprocessor Machine Code. *In Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 379–391, 2009. doi:10.1145/1480881.1480929

[94] S. Owens, S. Sarkar, and P. Sewell. A Better x86 Memory Model: x86-TSO. *In Proceedings of the Conference on Theorem Proving in Higher Order Logics (TPHoLs)*, 2009.

[95] S. V. Adve and M. D. Hill. A Unified Formalization of Four Shared-Memory Models. *IEEE Transactions on Parallel and Distributed Systems*, June 1993. doi:10.1109/71.242161

[96] S. V. Adve, M. D. Hill, B. P. Miller, and R. H. B. Netzer. Detecting Data Races on Weak Memory Systems. *In Proceedings of the 18th Annual International Symposium on Computer Architecture (ISCA'91)*, pages 234–43, May 1991. doi:10.1145/115952.115976

[97] J. Alglave, L. Maranget, S. Sarkar, and P. Sewell. Fences in Weak Memory Models. *In Proceedings of the International Conference on Computer Aided Verification*, July 2010. doi: 10.1007/978-3-642-14295-6_25

[98] J. Alglave, L. Maranget, S. Sarkar, and P. Sewell. Litmus: Running Tests Against Hardware. *In Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Mar. 2011. doi:10.1007/978-3-642-19835-9_5

[99] S. V. Adve and M. D. Hill. Weak Ordering—A New Definition. *In Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA'90)*, pages 2–14, May 1990. doi:10.1109/ISCA.1990.134502

[100] ARM Limited. *ARM Architecture Reference Manual*. ARMv7-A and ARMv7-R Edition Errata Markup, 2011.

[101] ARM Limited. *ARM v7A+R Architectural Reference Manual*. Available: from ARM Ltd.

[102] H.-J. Boehm and S. V. Adve. Foundations of the C++ Concurrency Memory Model. *In Proceedings of the Conference on Programming Language Design and Implementation*, June 2008. doi:10.1145/1375581.1375591

[103] H. W. Cain and M. H. Lipasti. Memory Ordering: A Value-Based Approach. *In Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)*, June 2004. doi: 10.1109/ISCA.2004.1310766

[104] M. D. Hill. Multiprocessors Should Support Simple Memory Consistency Models. *IEEE Computer,* vol. 31, no. 8, pages 28–34, August 1998. doi:10.1109/2.707614

[105] M. Meneghin, D. Pasetto, H. Franke, F. Petrini, and J. Xenidis. *Performance evaluation of inter-thread communication mechanisms on multicore/multithreaded architectures*. IBM Technical Report, May 2012.

[106] M. Meneghin, D. Pasetto, H. Franke, F. Petrini, and J. Xenidis. Performance evaluation of inter-thread communication mechanisms on multicore/multithreaded architectures. *In proceedings of 21st International ACM Symposium on High-Performance Parallel and Distributed Computing*, Delft, Netherlands, June 2012.

[107] S. M.-Haim, R. Alur, and M. M. K. Martin. Generating Litmus Tests for Contrasting Memory Consistency Models. *In Proceedings of the 22nd International Conference on Computer Aided Verification*, July 2010. doi:10.1007/978-3-642-14295-6_26

[108] W. Pugh. *The Java Memory Model is Fatally Flawed*. Concurrency: Practice and Experience, vol. 12, no. 1, pages 1–11, 2000.

[109] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[110] E. C. Lewis and L. Snyder. Pipelining wavefront computations: Experiences and performance. *In Proceedings of the 15th IEEE international Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2000)*, April 2000.

[111] G. R. Gao and V. Sarkar. Location consistency-a new memory model and cache consistency protocol. *IEEE Transactions on Computers*, vol. 49, no. 8, pages 798-813, August 2000.

[112] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. *In Proceedings of the 22nd International Symposium on Computer Architecture*, pages 24–36, June 1995.

[113] A. Phansalkar, A. Joshi, and L. K. John. Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite. *In Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA'07)*, pages 412–423, New York, NY, USA, 2007.

[114] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC Benchmark Suite: Characterization and Architectural Implications. *In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, October 2008.

[115] C. Bienia, S. Kumar, and K. Li. PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors. *In Proceedings of IEEE International Symposium on Workload Characterization (IISWC 2008)*, pages 47-56, September 2008. doi:10.1109/IISWC.2008.4636090

[116] K. Aasaraai and A. Moshovos. An Efficient Non-blocking Data Cache for Soft Processors. *In Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs (RECONFIG '10)*, pages 19–24, December 2010. doi: 10.1109/ReConFig.2010.61