Realization and Performance Comparison of Sequential and Weak Memory Consistency Models in Network-on-Chip based Multi-core Systems

By : Abdul Naeem
Xiaowen Chen
Zhonghai Lu
Axel Jantsch

Department Electronic Systems, School of ICT, Royal Institute of Technology (KTH), Stockholm, Sweden
Presentation outline

- Distributed shared memory multicore systems
- Memory consistency
- Sequential and weak memory consistency models
- Realization sequential and weak consistency models in the Network-on-chip based multi-core platform
- Experimental results
- Conclusion
Shared memory multicore systems

McNoC (NoC based multicore systems)
- Promising solution to the design of multicore systems
- Integration of computation and communication

Shared memory organization:
- UMA or SMP architectures
- NUMA or DSM architectures
- PM node: Processor-memory node

Figure: Homogeneous McNoC
Memory consistency

- A read should return the (most recent) value.
- Write propagation or atomicity
- Write serialization: writes seen in the same order
- Synchronization: avoid dependencies and data races
  - Synchronization primitives/APIs
  - Underlying hardware support

Memory consistency is related to the:

- Memory consistency determines the order on shared memory operations execution
  - Ordering constraint: on shared memory operations
    - What kind of shared memory operations can be overlapped for what program segment?
Memory consistency models (MCMs)

- Various MCMs are based on different ordering constraints.
  - Strict consistency (atomic consistency) [Hutto et al]
  - Cache consistency (Cache coherence)
  - Sequential consistency [Lamport et al]
  - Causal Consistency [Hutto et al]
  - Relaxed consistency models
    - Weak consistency [Dubois et al] (our focus)
    - Release consistency [Gharachorloo et al]
    - PRAM consistency (also known as FIFO consistency) [Lipton et al]
    - Proccessor consistency [Goodman et al]
Sequential consistency model

The sequential consistency has to maintain:

- The program order among operations of each individual processor in multiprocessor system
- The sequential order among multiple processors in the system.

Dekker’s algorithm for critical sections has the problems:

- Only software solution
- Deadlock
- Mutual exclusion
Global orders to enforce for SC

- The sequential consistency (often called Strong Ordering) model does not allow the reordering in the shared memory operations in the multi-processor system:
  - Read → Read
  - Read → Write
  - Write → Read
  - Write → Write

Figure: a) Strong Ordering  b) Global Orders
Realization of Sequential consistency

- The processor is stalled on issuance of a shared memory operation.
- On completion of a shared memory operation, the next operation is issued to the shared memory (1).
- All the memory operations are issued and completed in the order specified in the program. Program order is maintained.
- Sequential order is maintained by read-modify-write operation.

Figure: Memory operations flow in Sequential consistency
Limitations in sequential consistency

Sequential consistency model restricts system optimizations [S. V. Adve et al]:

Hardware optimizations:
- Write buffers: read after write bypass store buffer
- General interconnection network:
- Caches: coherence protocol, write completion

Software (compiler) optimizations:
- The compiler shifting to avoid data dependency.
- Loop unrolling: to reduce the control dependency
- Register allocation: to a memory variable to reduce memory references
Relax the requirements

- As SC does not allow these performance optimizations
- Relaxed consistency models emergence
- Relaxation among the independent shared memory operations

**Relax program order requirement:**
- Read → Read
- Read → Write
- Write → Read
- Write → Write

**Relax write atomicity requirement:**
- Write overlapping with following operations in a synchronized program
Weak memory consistency

Principle:

- Weak consistency (weak ordering) model classify shared memory operations as data and synchronization operations.
- Data operations between two consecutive synchronization points can be reordered.
- Atomic or sequential synchronization operations must be uninterrupted.

Working:

- All previously issued outstanding data operations must be completed before the issuance of synchronization operation and vice versa.
Global orders to enforce for WC

There is possible **interference** between the data and synchronization operations:

- data $\rightarrow$ Synchronization
- Synchronization $\rightarrow$ data
- Synchronization $\rightarrow$ Synchronization

**Enforcement of global orders** avoid interference

![Diagram](image)

Figure: a) Weak consistency  b) Global Orders
Realization of weak consistency

Transaction counter approach:
- A counter in each processor to keep track of outstanding data operations.
- The data operations affect the counter.
- The counter zero value.
- The synchronization operations does not affect but check the counter.

Scalability study:
- Study the two consistency models in the context of NoC based multicore architectures.

Figure: Memory operations flow in weak consistency.
Comparison of both the SC and WC

Sequential Consistency:
- Allows no overlapping
- Processor is stalled till completion of previous operation

Weak Consistency:
- Data operations are overlapped
- But cannot be overlapped with the synch operations
- Transaction counter based realization approach

Figure: a) Strong Ordering  b) Weak Ordering
NoC based McNoc platform

Platform features:
- Homogenous McNoC
- Support 2D mesh topology.
- Deflection routing
- Synchronization Supporter (SS)
- Transaction counter (TC)
- Distributed shared memory (DSM)

Figure: a) Homogeneous McNoC  
b) PM node
Data Management Engine (DME)

DME features:

- Two mini-processors (concurrency)
- Micro-code (Flexibility)
- Distributed shared memory (DSM)
- Synchronization Supporter (SS)
- Processor, Network interfaces (CICU, NICU)
- Transaction counter (TC)

Figure: DME Structure
Experiments

Experimental setup:

- McNoC platform with DSM architecture
- Hardware synchronization support (SS)
- Tests with simple short Pseudo-code
- Different traffic patterns for NCS data.
- The critical section in the CS-node is protected by the lock in the SYNC-node

```c
// NONCRITICAL SECTION
<br>MEMORY_WRITE>, <ADDRESS>, <DATA>; // REMOTE SHARED WRITE
<br>MEMORY_READ>, <ADDRESS>; // REMOTE SHARED READ

// LOCK ACQUIRE
<br>LOCK_ACQUIRE>, <ADDRESS>; //REMOTE LOCK ACQUIRE

// CRITICAL SECTION
<br>MEMORY_WRITE>, <ADDRESS>, <DATA>; // REMOTE SHARED WRITE
<br>MEMORY_READ>, <ADDRESS>; // REMOTE SHARED READ

// LOCK RELEASE
<br>LOCK_RELEASE>, <ADDRESS>; //REMOTE LOCK RELEASE

// NONCRITICAL SECTION
<br>MEMORY_WRITE>, <ADDRESS>, <DATA>; // REMOTE SHARED WRITE
<br>MEMORY_READ>, <ADDRESS>; // REMOTE SHARED READ
```

Figure: Test-code

Figure: Synchronization and data requests
Results

Impact of network size on code and consistency latencies:

- Scalability study of the two consistency models
- The synchronization latency rises due to network traffic, delay and waiting time for acquiring lock (lock is locally polled).
- The hotspot traffic pattern for the synchronization operations suggest the clustered networks up to 16 nodes cluster size
Conclusion and future work

Conclusion:
- Weak consistency scale well as compared to the sequential consistency model
- Average synchronization latencies increase exponentially as the network scales. Suggest network clusters.

Future work:
- Exploration and analysis of the other relaxed memory consistency models
Questions!