Shared Memory Consistency Models

Authors: Sarita V. Adve and Kourosh Gharachorloo

- Nitin Bhardwaj

Outline

- What is Memory Consistency
- Sequential Consistency
  - Optimizations to SC
- Relaxed Memory Consistency
  - Processor Consistency
  - Week Consistency
  - Release Consistency
- Program Centric approach for relaxed models
- Comparison between different memory models
- Summary

Memory Consistency

Def: A *memory consistency model* for a shared address space specifies constraints on the order in which memory operations must appear to be performed (i.e. to become visible to the processors) with respect to one another.

P1	P2	(A, flag are zero initial)
A=1	while(flag == 0);	
flag=1	print A;	

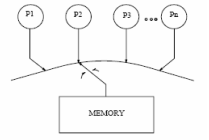
Sequential Consistency

Sequential Consistency(Lamport) "A multiprocessor 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 occur in this sequence in the order specified by its program."

Two Aspects:

Program Order

Write Atomicity



Understanding Program Order

P1	P2
Initially X = 2	
…..	…..
reg0=Read(X)	reg1=Read(X)
reg0=reg0+4	reg1=reg1+1
Write(reg0,X)	Write(reg1,X)

Possible execution sequences:

P1:reg0=Read(X) P2:reg1=Read(X) P1:reg0=reg0+4 P2:reg1=reg1+1 P1:Write(reg0,X) P2:Write(reg1,X) X=3	P2:reg1=Read(X) P2:reg1=reg1+1 P2:Write(reg1,X) P1:reg0=Read(X) P1:reg0=reg0+4 P1:Write(reg0,X) X=6	.....

Sequential Consistency Example

Initially Flag1 = Flag2 = 0

P1	P2
Flag1 = 1	Flag2 = 1
if (Flag2 == 0)	if (Flag1 == 0)
critical section	*critical section*

(a)

Initially A = B = 0

P1	P2	P3
A = 1		
	if (A ==1)	
	B = 1	
		if (B==1)
		register1 = A

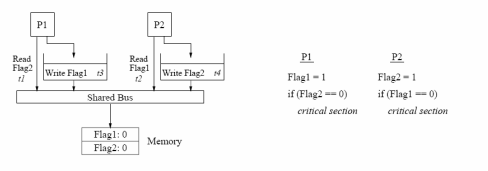
(b)

SC with common hardware optimizations

- **Architectures without caches**
- Architectures with caches

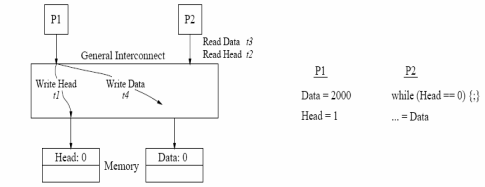
Write Buffers [Bypassing Capability]

- Reads bypass writes, reads are blocking



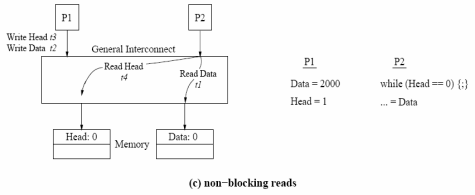
Overlapping Write Operations

- Writes may bypass other writes in write buffer



Non-blocking Reads

- Reads are allowed to bypass reads and writes



(c) non−blocking reads

SC with common hardware optimizations

- Architectures without caches
- **Architectures with caches**

Cache Coherence Vs SC

- Cache coherence ensures
  - Write is eventually visible to all processors.
  - Serialization of writes to the same location.
- Sequential consistency requires
  - Writes to all locations be seen in the same order by all processors.
  - Operations of a single processor to execute in program order.

Cache coherence can be viewed as a means to propagate newly written values to any given processor

Violation of SC

Initially A = B = C = 0

P1	P2	P3	P4
A = 1	A = 2	while (B != 1) {;}	while (B != 1) {;}
B = 1	C = 1	while (C != 1) {;}	while (C != 1) {;}
		register1 = A	register2 = A

- Result : register1=1 and register2=2 is possible. Violation of SC model due to general network
- Solution:
  - All updates or invalidations originate from a single point (e.g. Directory) maintaining order of these messages.
  - Wait for acknowledgment of previous write, before issuing another update or invalidation message to the same address.

Drawbacks of SC

- SC imposes heavy performance penalty.
- SC restricts any compiler optimization that can result in reordering memory operations
  - Code motion, register allocation, common sub-expression elimination, loop blocking, software pipelining
- SC restricts hardware generated memory re-orderings because of program-order and write-atomicity requirements.
  - Write Buffers, OOO instruction issue, pipelining of memory operations, lock-up free caches, non-atomic memory operations.

Above restrictions motivates for Relaxed Memory Models

Classification of RMM

- Optimizations
  - Write ⟶ Read
  - Write ⟶ Write
  - Read ⟶ Read, Write
  - Read other's write early
  - Read own write early
- All Models provide Safety net
- All models maintain uni-processor data and control dependencies
- Write serialization is maintained by all the models except PC, RCpc, PowerPC *[for most practical purposes where all processors observe all write operations in the same order (write serialization), is indistinguishable from a system where all writes are executed atomically]*

Categorization of Relaxed Models

Relaxation:	W →R Order	W →W Order	R →RW Order	Read Others' Write Early	Read Own Write Early	Safety Net
IBM 370	✓					serialization instructions
TSO	✓				✓	RMW
PC	✓			✓	✓	RMW
PSO	✓	✓			✓	RMW, STBAR
WO	✓	✓	✓		✓	synchronization
RCsc	✓	✓	✓		✓	release, acquire, nsync, RMW
RCpc	✓	✓	✓	✓	✓	release, acquire, nsync, RMW
Alpha	✓	✓	✓		✓	MB, WMB
RMO	✓	✓	✓		✓	various MEMBARs
PowerPC	✓	✓	✓	✓	✓	SYNC

Relaxing Writes to Reads

- IBM370, TSO, PC allows reads to be re-ordered w.r.t previous writes.

The three models differ in:

- IBM 370 stalls on read before write made visible to system.
  - Provides Serialization Instructions e.g. CAS, T&S
- TSO permits early read of writes from own processor before serialization
  - Provides RMW instructions. Replace Read and Write with RMW (dummy write/read).
- PC permits read of all write without serialization.
  - PC requires all accesses to memory location to be replaced with a RMW instruction since other processor may write to same location.

Processor Consistency

- Processor consistency (PC)
  - Writes done by a single processor are received by all other processors in the order in which they were issued, but writes from different processors may be seen in a different order by different processors
- The basic idea
  - To better reflect the reality of networks in which the latency between different nodes can be different.

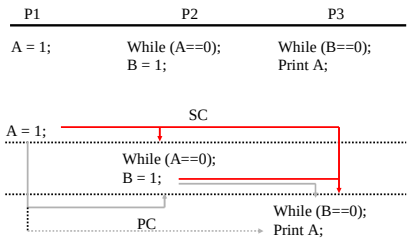
Processor Consistency (Cont'd)

- Two memory access conditions
  - Before a read is allowed to perform with respect to any other processor, all previous read accesses must be performed.
  - Before a write is allowed to perform with respect to any other processor, all previous read or write accesses must be performed.

Initially x=y=0;	
P1: W(y)2	**W(x)1**
P2:	**R(x)1 R(y)0**

NO

PC Example



IBM 370	**0 (No)**	**SC: print 1**
TSO	**0 (No)**	**PC: print 0 or 1**

Relaxing W to R and W to W

(Sparc V8 PSO)

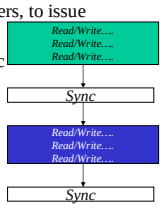
- Write to different locations can be pipelined and are allowed to reach memory and other cached copies out of order
- PSO is similar to TSO w.r.t write atomicity requirements
- Provides STBAR instruction for imposing program order between two writes.

Relaxing All Program Orders

- Read or a Write operation may be reordered w.r.t following read or write to a different location
  - Weak Ordering Model
    - Release Consistency Model (RCsc / RCpc)
    - Digital Alpha, Sparc V9 RMO, IBM Power PC
- Except Alpha, the above models allow reordering of two reads to the same location.
- RCpc and PowerPC allow a read to return the value of another processors write early.

Weak Ordering

- Classifies instructions into "Data" and "Sync"
- Reordering memory operations between sync operations.
- Hardware Implementation using WO counters, to issue sync operation counter must be zero
- No operations are issued until previous sync operation completes
- Synchronization accesses are sequentially consistent with respect to one another.



Weak Ordering (Cont'd)

- Open up opportunities for buffering of reordered write operations between two synchronization points.

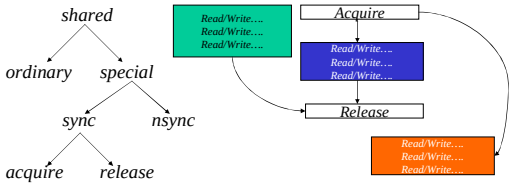
P1: W(x)1	**W(x)2**		**S**
P2:		**R(x)0 R(x)2**	**S R(x)2**
P3:			**R(x)1 S R(x)2**

OK

```
TOP: while (flag2 == 0)
       A = 1;
       u = B;
       v = C;
       D = B*C;
     flag2 = 0;
     flag1 = 1;
     goto TOP;
```

Release Consistency

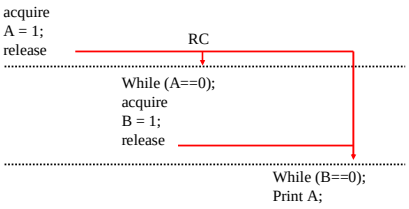
- Extends WO and makes distinction among sync and non-sync operations
- RCsc maintains sequential consistency among special operations
- RCpc maintains processor consistency among special operations



RC Example

Before an ordinary access to a shared variable is performed, all previous acquires done by the process must have completed successfully.

Before a release is allowed to be performed, all previous reads and writes done by the process must have completed.



Alpha, RMO and PowerPC

- Alpha employ RCsc model with Memory Barrier and Write Memory Barrier (WMB) fence instructions.
- Sparc V9 (RMO) employ RCsc model with MemBar instruction to specify any combination of RtoR, RtoW, WtoR, WtoW ordering.
  - No need for RMW to preserve WtoR ordering
  - Write atomicity is maintained
- PowerPC employ RCpc
  - SYNC instruction similar to MB instruction except for RtoR order.
  - RMW required to make writes atomic and preserve RtoR order.

Programmer Centric View

- System Centric view is accompanied by higher level of complexity for programmers.
- Varied semantics for different models complicates the task of porting programs across systems.

Motivates for higher level of abstraction for programmers

- Provide informal rules for correct results defined by SC

i.e. Consistency Model is defined in terms of program level information provided by the programmer.

  - DRF0 is one such approach which explores the information that is required to allow optimization similar to Weak Ordering.
  - PL (Properly Labeled) approach for defining RCsc optimizations.

The Data-race-free-0 Model

- Weak Ordering classifies instruction into “Data” and “Sync”

Key Goal is to formally distinguish operations as data or Synchronization on the bases of races.

- An operation forms a race with another operation if,
  - They access the same location && atleast one operation is a write && there are no intervening operations between the two operations.

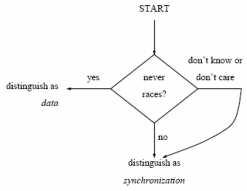
P1	P2
A = 23;	while (Flag != 1) {;}
B = 37;	… = B;
Flag = 1;	… = A;

Flag = Synchronization, Data = A, B

Can optimize operations that never race.

Programming With DRF-0

- Write operation assuming SC.
- For every memory operation specified in the program do:



•Language Support:

Synchronization with special constructs

Support to distinguish individual accesses

Comparison of Sync/Swym and Niagara

- Sync/Swym are Sparc V9 architecture based systems which defines three different memory models: TSO, PSO, RMO (Relaxed Memory Ordering) model.
- Programs written for RMO will work in PSO and TSO as well. Programs written for PSO will work in TSO. MEMBAR inst. induce ordering in the inst. stream of a single processor.
  - Portability issues: Programs which use single-writer/multiple reader locks for all shared accesses are portable across all models.
  - Programs that use write locks to protect write accesses but read without locking will be portable across all memory models, only if writes to shared data are separated by MEMBAR #StoreStore instructions, and if reading the lock is followed by a MEMBAR #Load-Load instruction.
- Niagara has 2 flavors of Stores: TSO and RMO (Read memory Order)

Consistency Model of Power4

- For certain instructions which require completion serialization. Groups so marked will not be issued until that group is the next to complete, i.e., all prior groups have successfully completed.
- Additionally, instructions that read a non-renamed register cannot be executed until we are sure all writes to that register have completed.
- Load Hit Store: Read Owns Write Early.
- Store Hit Load: Write -> Read Program Order is Relaxed.
- Load Hit Load: Read -> Read Program Order is Relaxed.

  To guard against Read Others Write Early, if a younger load obtains old data then the older store must obtain new data. This requirement is called sequential load consistency.

Consistency Model of Intel P4

Named as a Processor Ordering with Following rules built in:

- Reads can be carried out speculatively and in any order.
- Reads can pass buffered writes, but the processor is self-consistent.
- Writes to memory are always carried out in program order, with the exception of writes executed with [CLFLUSH instruction and streaming stores (writes)
- Writes can be buffered.
- Writes are not performed speculatively; they are only performed for instructions that have actually been retired.
- Data from buffered writes can be forwarded to waiting reads within the processor.
- Reads or writes cannot pass (be carried out ahead of) I/O instructions, locked instructions, or serializing instructions.
- Reads cannot pass LFENCE and MFENCE instructions.
- Writes cannot pass SFENCE and MFENCE instructions.

Summary

- Defined Sequential Consistency
- Optimizations to SC
- Relaxed Memory Models
  - IBM 370, TSO, PC
  - PSO and SPARC V9
  - WO
  - RC (RCsc and RCpc)
- Provided a Programmer Centric view for identifying different operations in a program.
- Discussed consistency models of Sparc-V9, Power4 and Intel P4.