Shared Memory Implementation

- Coherence - defines the behavior of reads and writes to the same memory location
  - ensuring that modifications made by a processor propagate to all copies of the data
  - Program order preserved
  - Writes to the same location by different processors serialized
- Synchronization - coordination mechanism
- Consistency - defines the behavior of reads and writes with respect to access to other memory locations
  - defines when and in what order modifications are propagated to other processors

150

Memory Consistency Model

- Specifies constraints on the order in which memory operations to different locations must appear to be performed with respect to one another

151

Sequential Consistency

- ``A 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." [Lamport 79]

152

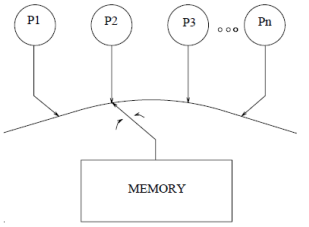


Figure 3: Programmer's view of sequential consistency.

153

Implications

- Program order (among operations from individual processors)

P1	P2
A = 1;	while (flag == 0);
flag = 1;	print A;

- Write atomicity (single sequential order among operations from all procesors)

P1	P2	P3
A = 1;	while (A == 0);	
	B = 1;	while (B == 0);
		print A;

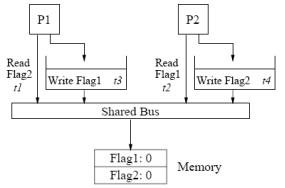
Dekker's Algorithm

P1: A = 0;	P2: B = 0;
…	…
A = 1;	B = 1;
L1: while (B == 1) {..}	L2: while (A == 1) {…}
…	…

Can B = 0 at P1 and A = 0 at P2 at the corresponding if statements?

Write Buffers [Bypassing Capability]

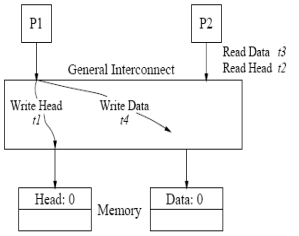
- Reads bypass writes, reads are blocking



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

Overlapping Write Operations

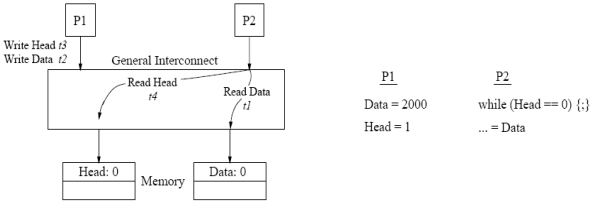
- Writes may bypass other writes in write buffer



P1	P2
Data = 2000	while (Head == 0) {;}
Head = 1	... = Data

Non-blocking Reads

- Reads are allowed to bypass reads and writes



(c) non−blocking reads

158

Drawbacks of Sequential Consistency

- SC imposes a 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

159

Consistency Model Classification

- Models vary along the following dimensions
  - Local order - order of a processor's accesses as seen locally
  - Global order - order of a single processor's accesses as seen by each of the other processors
  - Interleaved order - order of interleaving of different processor's accesses on other processors

160

Memory Model Relaxations

- Possible relaxations
  - Write ⟶ Read
  - Write ⟶ Write
  - Read ⟶ Read, Write
  - Read other's write early
  - Read own write early
- All Models provide some Safety net
- All models maintain uni-processor data and control dependencies
- Write atomicity is maintained by all the models except PC, RCpc, PowerPC

161

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

162

Maintaining Write Atomicity

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

163

Write Atomicity (continued)

P1	P2	P3
A = 1;	while (A == 0);	
	B = 1;	while (B == 0);
		print A;

370? TSO? PC?

164

Consider Peterson's Algorithm

- Process P_i

```
do {
  flag[i] = true;
  turn = j;
  while (flag[j] && turn==j) ;
    critical section
  flag[i] = false;
    remainder section
} while (1);
```

165

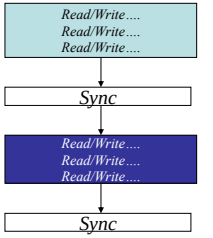
Relaxing All Program Orders

- 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.

166

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.



167

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**

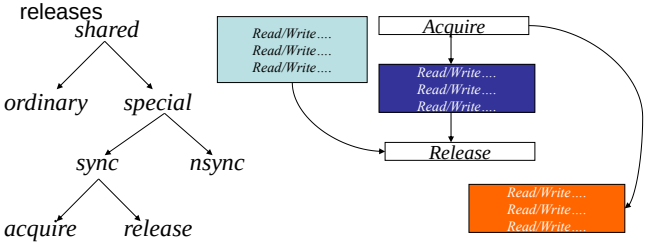
OK

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

168

Release Consistency

- Extends WO and makes distinction among sync and non-sync operations
- Ordinary accesses are completely unordered with respect to each other
- Synchronization operations divided into acquires and releases

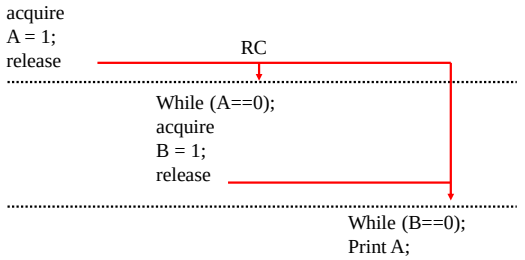


169

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.



170

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.

171

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.

172

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 basis of data races

- An operation forms a race with another operation if,
  - They access the same location && at least 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;

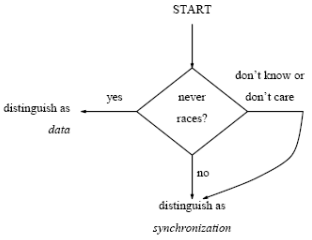
Flag = Synchronization, Data = A, B

Can optimize operations that never race

173

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

174

Distinguishing Memory Operations

- At the Programming Language Level
  - Special synchronizaiton operation (library call)
  - High-level paradigms
  - Data or synchronization attribute with code or data
- At the hardware level
  - Address regions
  - Special instructions

175