Coherence, Locking and Consistency- III

Dr. A. Sahu
CSE, IIT Guwahati

Multicore Difficulties

- Multiprocessors are likely to be cost/power effective solutions
  - Because it share lots of resources
    - Personal room is costlier than dormitory
  - Sharing resource arise many other problems
    - Coherence
      - Shared data at all placed should be same
    - Critical Sections
      - Lock and Barrier Design
    - Consistency
      - Order should be similar to serial (ROB)
    - One processor Interference others
      - Share efficiently using some policy

Synchronization

- Processes run on different processors independently
- At some point they need to know the status of each other for
  - Communication, mutual exclusion etc
- Simple Locking

Simple Locking

lock : ld reg, loc // copy location to reg
cmp reg #0 //compare with 0
bnz lock  // if not zero try again
st loc #1 // store 1 at loc to mark it locked
return;

Unlock: st loc #0
return

Synchronization Problem

Hennessy Book 5th Ed, Sec 5.5, Culler Sec 5.5
- Hardware primitive for atomic read+write is required e.g.
  - Test & Set, // test for unlock (0) then set the lock (1)
  - Exchange,
  - Fetch & Increment
Spin Lock with Exchange Instr.

Lock: 0 indicates free and 1 indicates locked

Code to lock X:

\[
\begin{align*}
& \text{lockit: } r2 \leftarrow 1 \quad \text{atomic exchange} \\
& \text{if}(r2\neq 0) \rightarrow \text{lockit} \; \text{already locked}
\end{align*}
\]

locks are cached for efficiency, coherence is used

Better code to lock X:

\[
\begin{align*}
& \text{lockit: } r2 \leftarrow X \; \text{read lock} \\
& \text{if}(r2\neq 0) \rightarrow \text{lockit} \; \text{not available} \\
& r2 \leftarrow 1 \\
& r2 \leftarrow X \; \text{atomic exchange} \\
& \text{if}(r2\neq 0) \rightarrow \text{lockit} \; \text{already locked}
\end{align*}
\]

LD Locked & ST conditional

Simpler to implement

• Atomic exchange using LL and SC

try: \( r3 \leftarrow r2 \); move exchange value

LL \( r1, X \); load locked

SC \( r3, X \); store conditional

if \( r3=0 \) \( \rightarrow \) try; branch store fails

\( r2 \leftarrow r1 \); put loaded value in \( r2 \)

• Fetch & increment using LL and SC

try: \( \text{LL } r1, X \); load locked

\( r3 \leftarrow r1 + 1 \); increment

SC \( r3, X \); store conditional

if \( r3=0 \) \( \rightarrow \) try; branch store fails

Spin Lock with LL & SC

lockit:

\[
\begin{align*}
& \text{LL } r2, X \; \text{load locked} \\
& \text{if}(r2\neq 0) \rightarrow \text{lockit} \; \text{not available} \\
& r2 \leftarrow 1 \\
& \text{SC } r2, X \; \text{store cond} \\
& \text{if}(r2=0) \rightarrow \text{lockit} \; \text{branch store fails}
\end{align*}
\]

Performance of Locking

• Spinning waste time

• Recall MAC Protocol
  – Non Persistence CSMA protocol
  – Wait random time if medium if busy, then send

• Spin lock with exponential back-off reduces contention
  – Wait \( k \) amount of time for 1st attempt
  – Wait \( k^2c \) amount of time for 2nd attempt

Barrier Synchronization

DoWork() {
  Do Phase I work
  Barrier();
  Do Phase II work
  Barrier();
  Do Phase III work()
  Barrier();
}

for \( i=0; i<\text{NumProc}; i++ \) {
  P[i].Start(DoWork);
}
Print result();
Barrier Synchronization

\[
\text{Barrier}\()
\begin{align*}
\text{lock (X)} \\
\text{if(count=0) release } & \leftarrow 0 \\
\text{count++} \\
\text{unlock(X)} \\
\text{if(count=total) } & \{\text{count} \leftarrow 0; \text{release} \leftarrow 1\} \\
\text{else spin until(release==1)} \\
\end{align*}
\]

If All Processor Passes/completed then they go to next Phase

• Sum of all finished processor

\begin{itemize}
\item Every one accessing to same lock
\item Lock contention is distributed in a Tree Fashion
\end{itemize}

Improved Barrier Synchronization

\[
\text{Do Phase I work:} \\
\text{Barrier(bar1, p);} \\
\text{//After this release=1, but not} \\
\text{// visible to all some how, it may} \\
\text{// happened one process is not got this} \\
\text{// and waiting while other entered to} \\
\text{// NExt barrier} \\
\text{Do Phase II work:} \\
\text{Barrier(bar1, p);} \\
\text{// Some will enter to this not all, So} \\
\text{barrier will not end at all.}
\]

Memory Consistency Problem

\begin{itemize}
\item When must a processor see the value that has been written by another processor? Atomicity of operations – system wide?
\item Can memory operations be re-ordered?
\end{itemize}

Various models: Highly recommended by HP Book

http://rsim.cs.uiuc.edu/~sadve/Publications/models_tutorial.ps

Example

\[
\begin{align*}
P1: & \ A = 0 & P2: & \ B = 0 \\
\ldots & \ldots & \ldots & \ldots \\
L1: & \text{if}(B=0)S1 & L2: & \text{if}(A=0)S2
\end{align*}
\]

Which statements among S1 and S2 are done?

Both S1, S2 may be done if writes are delayed
**Sequential Consistency : Example**

S1: \( X = 10 \)  
L1: \( R1 = Y \)  
S2: \( Y = 10 \)  
L2: \( R2 = X \)

\[ \text{Time} \]

- S1: \( X = 10 \)  
- L1: \( R1 = Y \)  
- S2: \( Y = 10 \)  
- L2: \( R2 = X \)

- SC
- SC
- SC
- SC

Load are preferred by Store : Buffered

**Relaxed Consistency Models : Rules**

- \( X \rightarrow Y \)  
  - Operation X must complete before operation Y is done  
- Sequential consistency requires: \( R \rightarrow W, R \rightarrow R, W \rightarrow R, W \rightarrow W \)

- Relax \( W \rightarrow R \)  
  - ”Total store ordering”

- Relax \( W \rightarrow W \)  
  - ”Partial store order”

- Relax \( R \rightarrow W \) and \( R \rightarrow R \)  
  - ”Weak ordering” and ”release consistency”

- Consistency model is multiprocessor specific  
- Programmers will often implement explicit synchronization

**Relaxing \( \text{W} \rightarrow \text{R} \) order**

Loads are allowed to overtake stores  
\( \Rightarrow \) Write buffering is permitted

1. Total Store Ordering : \( \text{Writes are atomic} \)
2. Processor Consistency : \( \text{Writes need not be atomic - Invalidations may gradually propagate} \)
3. PSO: Write can be ordered

**Examples**

P1  
A = 1;    
while(\( \text{flag} = 0; \))  
flag = 1;    
print A;

SC ensures that "1" is printed  
TSO, PC also do so  
PSO does not

P1  
A = 1;    
print B;    
B = 1;    
print A;

SC ensures that if B is printed as "1" then A is also printed as "1"  
TSO, PC also do so  
PSO does not
Examples - continued

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

SC ensures that "1" is printed. TSO and PSO also do that but PC
does not
//PSO does as only Write per Process

P1
A = 1;      B = 1;
print B;    print A;

SC ensures that both can’t be
printed as "0". TSO, PC and
PSO do not

Relaxing all R/W order

Weak Ordering or Weak Consistency
- Loads and Stores are not restricted to follow
  an order
- Explicit synchronization primitives are used
- Synchronization primitives follow a strict order
- Easy to achieve
- Low overhead

Release Consistency
- Further relaxation of weak ordering
- Synch primitives are divided into acquire and
  release operations
- R/W operations after an acquire can not move
  before it but those before it can be moved after
- R/W operations before a release can not move
  after it but those after it can be moved before

WC and RC Comparison

WC

1
R/W
R/W
synch

R/W
R/W

2
R/W
R/W
synch

R/W
R/W

3
R/W
R/W

RC

1
R/W
R/W

R/W
R/W

2
R/W
R/W

acquire

release

3
R/W
R/W