Reading

- Patterson & Hennesey, Computer Architecture (2nd Ed.) secn 8.6 – a condensed treatment of consistency models
Coherence and Consistency

• Memory coherence
  – behavior of a single memory location M
  – viewed from one or more processors
  – informally
    • all writes to M are seen in the same order by all processors

• Memory consistency
  – behavior of multiple memory locations read and written by multiple processors
  – viewed from one or more processors
  – informally
    • concerned with the order in which writes on different locations may be seen
Coherence of memory location $x$

- Defined by three properties (assume $x = 0$ initially)

(a) $P_1$: $W(x,1)$ \[ 1 = R(x) \]
   
   no intervening write of $x$ by $P_1$ or other processor

(b) $P_1$: $W(x,1)$
$P_2$: \[ 1 = R(x) \]

sufficiently large interval and no other write of $x$

(c) $P_1$: $W(x,1)$ \[ a = R(x) \]
$P_2$: $W(x,2)$ \[ a = R(x) \]
$P_3$: \[ a = R(x) \]

\[ a \in \{1,2\} \]

and has same value at all processors

sufficiently large interval and no other writes of $x$
Consistency Models

• The consistency problem
  – Performance motivates replication
    • Keep data in caches close to processors
  – Replication of read-only blocks is easy
    • No consistency problem
  – Replication of written blocks is hard
    • In what order do we see different write operations?
    • Can we see different orders when viewed from different processors?
  – Fundamental trade-offs
    • Programmer-friendly models perform poorly
Consistency Models

• The importance of a memory consistency model

\[
\text{initially } A = B = 0
\]

\[
\begin{align*}
P1 & : A := 1; \\
& \text{if } (B \equiv 0) \\
& \ldots \text{P1 “wins”}
\end{align*}
\]

\[
\begin{align*}
P2 & : B := 1; \\
& \text{if } (A \equiv 0) \\
& \ldots \text{P2 “wins”}
\end{align*}
\]

– P1 and P2 may both win in some consistency models!
  • Violates our (simplistic) mental model of the order of events

• Some consistency models
  • Strict consistency
  • Sequential consistency
  • Processor consistency
  • Release consistency
Strict Consistency

• Uniprocessor memory semantics
  – Any read of memory location x returns the value stored by the most recent write operation to x
    • Natural, simple to program

P₁: \[ W(x, 1) \]

P₂: \[ 1 = R(x) \]

Strictly Consistent

P₁: \[ W(x, 1) \]

P₂: \[ 0 = R(x) \]

Not Strictly Consistent
Strict Consistency

- Implementable in a real system?
  - Requires...
    - absolute measure of time (i.e., global time)
    - slow operation else violation of theory of relativity!

- Claim: Not what we really wanted (or needed) in the first place!
  - Bad to have correctness depend on relative execution speeds
Sequential Consistency

- Mapping concurrent operations into a single total ordering
  - The result of any execution is the same as if
    - the operations of each processor were performed in sequential order and are interleaved in some fashion to define the total order
  - Example

\[
P_1: \ W(x, 1) \\
P_2: \ 0 = R(x) \quad 1 = R(x)
\]

\[
P_1: \ W(x, 1) \\
P_2: \quad 1 = R(x) \quad 1 = R(x)
\]

Both executions are sequentially consistent
Sequential Consistency: Example

• Earlier in time does not imply earlier in the merged sequence
  – is the following sequence of observations sequentially consistent?
  – what is the value of y?

\[
\begin{align*}
P_1: & \quad W(x, 1) \quad ? = R(y) \\
P_2: & \quad W(y, 2) \\
P_3: & \quad 2 = R(y) \quad 0 = R(x) \quad 1 = R(x)
\end{align*}
\]
Processor Consistency

- Concurrent writes by different processors on different variables may be observed in different orders
  - there may not be a single total order of operations observed by all processors
- Writes from a given processor are seen in the same order at all other processors
  - writes on a processor are “pipelined”

<table>
<thead>
<tr>
<th>P1:</th>
<th>W(x, 1)</th>
<th>0 = R(y)</th>
<th>1 = R(y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2:</td>
<td>W(y, 1)</td>
<td>0 = R(x)</td>
<td>1 = R(x)</td>
</tr>
<tr>
<td>P3:</td>
<td></td>
<td>1 = R(x)</td>
<td>0 = R(y)</td>
</tr>
<tr>
<td>P4:</td>
<td></td>
<td>0 = R(x)</td>
<td>1 = R(y)</td>
</tr>
</tbody>
</table>
Processor consistency

- Typical level of consistency found in shared memory multiprocessors
  - insufficient to ensure correct operation of many programs
- Ex: Peterson’s mutual exclusion algorithm

```plaintext
program mutex
var enter1, enter2 : Boolean;
  turn: Integer

process P1
  repeat forever
    enter1 := true
    turn := 2
    while enter2 and turn=2 do skip end
    ... critical section ...
    enter1 := false
    ... non-critical section ...
  end repeat
end P1;

process P2
  repeat forever
    enter2 := true
    turn := 1
    while enter1 and turn=1 do skip end
    ... critical section ...
    enter2 := false
    ... non-critical section ...
  end repeat
end P2;

begin
  enter1, enter2, turn := false, false, 1
  cobegin P1 || P2 coend
end
```
Weak Consistency

• Observation
  – memory “fence”
    • if all memory operations up to a checkpoint are known to have completed, the detailed completion order may not be of importance
  – defining a checkpoint
    • a synchronizing operation $S$ issued by processor $P_i$
      – e.g. acquiring a lock, passing a barrier, or being released from a condition wait
      – delays $P_i$ until all outstanding memory operations from $P_i$ have been completed in other processors

• Execution rules
  – synchronizing operations exhibit sequential consistency
  – a synchronizing operation is a memory fence
  – if $P_i$ and $P_j$ are synchronized then all memory operations in $P_i$ complete before any memory operations in $P_j$ can start
Weak Consistency: Examples

P_1: \( W(x, 1) \quad W(y, 2) \quad S \)

P_2: \( 1 = R(x) \quad 0 = R(y) \quad S \quad 1 = R(x), 2 = R(y) \)

P_3: \( 0 = R(x) \quad 2 = R(y) \quad S \quad 1 = R(x), 2 = R(y) \)

Weakly consistent

P_1: \( W(x, 1) \quad W(x, 2) \quad S \)

P_2: \( S \quad 1 = R(x) \)

Not weakly consistent
Memory consistency: processor-centric definition

- A memory consistency model defines which orderings of memory-references made by a processor are preserved for external observers
  - Reference order defined by
    - Instruction order →
    - Reference type \{R,W\} or synchronizing operation (S)
    - location referenced \{a,b\}
  - A memory consistency model preserves some of the reference orders
    - Sequential Consistency (SC), Processor consistency = Total store ordering (TSO), Partial store ordering (PSO), weak consistency

<table>
<thead>
<tr>
<th>reference order</th>
<th>a = b</th>
<th>Consistency Model</th>
<th>a ≠ b</th>
</tr>
</thead>
<tbody>
<tr>
<td>(coherence)</td>
<td></td>
<td>SC</td>
<td>TSO</td>
</tr>
<tr>
<td>Ra → Rb</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Ra → Wb</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Wa → Wb</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Wa → Rb</td>
<td>*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>?a → S → ?b</td>
<td>*</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Consistency models: ordering of “writes”

• Sequential consistency
  – all processors see all writes in the same order

• Processor consistency
  – All processors see
    • writes from a given processor in the order they were performed (TSO) or in some unknown but fixed order (PSO)
    • writes from different processors may be observed in varying interleavings at different processors

• Weak consistency
  – All processors see same state only after explicit synchronization
Memory consistency: Summary

• Memory consistency
  – contract between parallel programmer and parallel processor regarding observable order of memory operations
    • with multiple processors and shared memory, more opportunities to observe behavior
    • therefore more complex contracts

• Where is memory consistency critical?
  – fine-grained parallel programs in a shared memory
    • concurrent garbage collection
    • avoiding race conditions: Java instance constructors
    • constructing high-level synchronization primitives
    • wait-free and lock-free programs
Memory consistency: Summary

• Why memory consistency contracts are difficult to use
  – What memory references does a program perform?
    • Need to understand the output of optimizing compilers
  – In what order may they be observed?
    • Need to understand the memory consistency model
  – How can we construct correct parallel programs that accommodate these possibilities?
    • Need careful thought and formal methods

• What is a parallel programmer to do, then?
  – Use higher-level concurrency constructs such as loop-level parallelization and synchronized methods (Java)
    • the synchronization inherent in these constructs enables weak consistency models to be used
  – Use machines that provide sequential consistency
    • Increasingly hard to find and invariably “slower”
  – Leave fine-grained unsynchronized memory interaction to the pros