Memory consistency models

Presented by:

Gabriel Tanase
Summary

- Memory Consistency (definition)
- Sequential Consistency
- Processor Consistency
- Week Consistency
- Release Consistency
  - Early Release Consistency
  - Lazy Release Consistency
  - Entry Consistency

Parasol
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.

\[ \begin{array}{ll}
\text{P1} & \text{P2} \\
A=1 & \text{while}(\text{flag} == 0); \\
\text{flag}=1 & \text{print } A; \\
\end{array} \]

\((Culler, Singh, Gupta)\)
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.”

Figure 5-10 Programmer’s abstraction of the memory subsystem under the sequential consistency model

The model completely hides the underlying concurrency in the memory-system hardware, for example, distributed main memory, the presence of caches and write buffers, from the programmer.
Sequential Consistency

1. Every process issues memory operations in program order.

2. After a write operation is issued, the issuing process waits for the write to complete before issuing its next operation.

3. After a read operation is issued, the issuing process waits for the read to complete, and for the write whose value is being returned by the read to complete, before issuing its next operation. That is, if the write whose value is being returned has performed with respect to this processor (as it must have if its value is being returned) then the processor should wait until the write has performed with respect to all processors.

(Culler, Singh, Gupta)
Processor Consistency

1. Before a read is allowed to perform with respect to any other processor, all previous read must be performed and

2. Before a write is allowed to performed with respect to any other processor all previous accesses(reads and writes) must be performed

- The above conditions relax sequential consistency by allowing reads following a write to bypass the write;

- Writes from the same processor should be observed in program order; The order in which the writes from two processors occur (as observed by themselves or a third processor need NOT be identical(Gharachorloo & al.)
Example

Case 1 (SC, PC)

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>(A, flag are zero initial)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=1</td>
<td>while(flag == 0);</td>
<td></td>
</tr>
<tr>
<td>flag=1</td>
<td>print A;</td>
<td></td>
</tr>
</tbody>
</table>

Case 2 (SC but not PC)

<table>
<thead>
<tr>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>(A, flag are zero initial)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=1</td>
<td>while(A == 0);</td>
<td>while(B == 0);</td>
<td></td>
</tr>
<tr>
<td>B=1;</td>
<td>print A;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Weak Consistency

- Ordinary shared accesses and synchronization accesses
- Conditions for weak consistency
  - Before an ordinary read/write access is allowed to perform with respect to any other processor, all previous synchronization accesses must be performed and
  - Before a synchronization access is allowed to performed with respect to any other processor, all previous ordinary read/write accesses must be performed and
  - Synchronization accesses are sequentially consistent.
Release Consistency

- Categorization of shared memory accesses

  Shared access
  - Competing
    - Synchronization
      - Acquire
      - Release
  - non-competing
    - Synchronization
      - non-synchroniztion
RC: Properly-Labeled Programs

- shared
  - special
    - synch
      - acq
      - rel
  - nsynch
    - ordinary

Two ways of labelling
- Parallelizing compilers
- Programming methodology
Conditions for Release Consistency

- Before an ordinary read or write access is allowed to perform with respect to any other processor, all previous *acquire* accesses must be performed and

- Before a *release* access is allowed to perform with respect to any other processor, all previous ordinary read and writes accesses must be performed, and

- Special accesses are sequential consistent with respect to one another
Comparison of the four models

Sequential Consistency

Processor Consistency

LOADs and STOREs can perform in any order as long as local data and control dependencies are observed.

Ordering among Ordinary and Special Accesses

Weak Consistency (WCsc)

Ordering between Ordinary and Special Accesses

Release Consistency (RCpc)
Performance Potential

Release Consistency (RCpc)

Accesses Serially

Weak Consistency

Release Consistency

Overlap in processing
Lazy release consistency for Software DSM

Pipelining Remote Memory Accesses in

Merging of Remote Memory Updates in
Eager Release Consistency

Repeated Updates of Cached Copies in Eager
Lazy Release Consistency

Message Traffic in LRC.
Lazy release consistency

- “happened-before-1” partial order
- Write notice propagation
- Multiple writer protocols
  - Modify different parts of a page concurrently
  - False sharing
  - Reduces the amount of messages
- Invalidate vs. update (on the acquiring processor)
Lazy versus Eager

Chooseky Messages.

Chooseky Dat
Entry Consistency

- Memory consistency requirements can be relaxed even more:
  - Parallel programs define their own higher level consistency requirements
    - Synchronization object (locks, barriers)
    - Critical section
    - Shared data accessed in critical section
  - A processor’s view of the shared memory becomes consistent with the most recent updates only when it enters a critical section (Bershad & al)
Entry Consistency

- All synchronization objects should be explicitly declared as instances of one of the synchronization data types provided (by Midway runtime): locks, barriers.

- All shared data must be explicitly labeled with the keyword `shared` which is understood by the compiler.

- All shared data must be explicitly associated with at least one synchronization object. This is made by calls to the runtime system, is dynamic and may change.
## Entry Consistency performance

<table>
<thead>
<tr>
<th>Matrix Multiply</th>
<th>#procs</th>
<th>Elapsed</th>
<th>Speedup</th>
<th>Data Transfered (MB)</th>
<th>#Msgs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry Cons</td>
<td>1</td>
<td>164</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>83.5</td>
<td>1.96</td>
<td>2.14</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>43.3</td>
<td>3.79</td>
<td>4.81</td>
<td>72</td>
</tr>
<tr>
<td>Release Cons</td>
<td>1</td>
<td>164</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>86.8</td>
<td>1.89</td>
<td>2.17</td>
<td>1802</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>48.4</td>
<td>3.39</td>
<td>4.97</td>
<td>5106</td>
</tr>
</tbody>
</table>
Scope Consistency
“A bridge between RC and EC”

- Scope Consistency offers most of the potential performance advantages of Entry Consistency.
- Doesn’t require explicit binding of data to synchronization variables
- “Consistency Scope” dynamic establish the binding of data to synchronization variables
- Most of the programs that follow LRC will be correct under scope consistency(ScC)
Consistency Scope

- “Consistency Scope” - modification to data within that scope are guaranteed to be visible within that scope:
  - Ex All critical sections guarded by the same lock
  - Session – the interval in which a consistency scope is open at a given processor.
  - Any modification during a consistency scope session become visible to processes that enter sessions of that scope.
Scope Consistency Rules

- Before a new session of a consistency scope is allowed to open at processor P, any write previously performed with respect to that consistency scope must be performed with respect to P.

- A memory access issued by processor P is allowed to perform only after all consistency scope sessions previously opened by P have been successfully opened.

(L.Iftode & al.)
Scope versus LRC

$P_0$

$X = 1$

*acquire* ($L_1$)

$Y = 1$

*release* ($L_1$)

$P_1$

*ScC propagates Y*

$LRC$ propagates $X$ and $Y$

*acquire* ($L_1$)

$a = Y$

$b = X$

Scope Consistency versus Lazy Release Consis
Conclusions

- The most important memory consistency models were presented.
- Every new model presented introduced a new relaxation.
- The more information a processor have the more optimizations can be performed.
- Some of them are suitable for hardware some for software distributed shared memory.
Bibliography


2. Lazy release consistency for software distributed shared memory; Pete Keleher, Alan Cox, Willy Zwaenepoel

3. The Midway distributed shared memory; Brian Bershad & al.

4. Scope Consistency: A bridge between release consistency and entry consistency; L. Iftode , J.P. Singh, K. Li

5. Parallel computer architecture(chapter 5 and 9); David Culler, J.P. Singh, A. Gupta
Delayed Consistency

Figure 1: System structure with Store buffers and coherence update buffers
Delayed protocols
Figure 5: Cache state diagram for the Receive Delayed protocol.
Update based cache protocols

Figure 2: A two-level cache hierarchy with two separate write buffers and a lockup-free second-level cache.