Relaxed Shared Memory Consistency Models

By Steven Davis & Seth Groder

Multiple Processor Systems 053  Presented: 5/18/06
Outline

- What are Shared Memory Consistency Models
  - Strict Models
  - Sequential Models
  - Relaxed Models
- Motivation for Relaxing Models
- Requirements of a Relaxed Consistency Model
- Classes of Relaxed Model
- Popular Relaxed Models
  - Weak Ordering (WO)
  - Release Consistency (RC)
- Questions
Shared Memory Consistency Models

- Specify a contract between programmer and system, wherein the system guarantees that if the programmer follows the rules, memory will be consistent and the results of memory operations will be predictable.
- Specify the order by which shared memory access events of one process should be observed by other processes in the system.
- A consistency model affects both programming languages and compilers as well as hardware.
  - Hardware and Compilers: The model indicates how much these components can reorder accesses. This limits the amount of optimization which can be achieved
  - Programming Language: The model must supply mechanisms to introduce necessary constraints on the reordering of accesses.
## Classes of Consistency Models

**Strict Consistency**
- A read always returns with most recent write to the same location
- Implicitly assumes the existence of absolute global time

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
<th>P1: W(x)a</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: R(x)a</td>
<td>R(x)NIL</td>
</tr>
</tbody>
</table>

(a) Strict

<table>
<thead>
<tr>
<th>P1: W(x)a</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2: R(x)a</td>
</tr>
</tbody>
</table>

(b) Not Strict
Classes of Consistency Models

- **Sequential Consistency**
  - The result of any execution appears the same as the result of the interleaving of individual programs strictly in sequential program order.
  - This is the easiest consistency model to understand, since the system is preserving a model behaving in a way expected by an average programmer.
  - Sequential consistency is weaker than atomic consistency (which would demand that operations are seen in order in which they were actually issued - which is essentially impossible to secure in distributed system, where deciding global time is virtually impossible)
  - Does not rely on global time
Sequential Consistency Example

/*Assume initial values of A and B are 0*/

(1a) A = 1;
(1b) B = 2;

(2a) print B;
(2b) print A;

- Possible outcomes for (A,B): (0,0), (1,0), (1,2); impossible under SC: (0,2)
- We know 1a->1b and 2a->2b by program order
- A = 0 implies 2b->1a, which implies 2a->1b
- B = 2 implies 1b->2a, which leads to a contradiction
- BUT, actual execution 1b->1a->2b->2a is SC (A,B)=(1,2) despite not program order
  - Appears just like 1a->1b->2a->2b as visible from results.
- Actual execution 1b->2a->2b-> is not SC (A,B)=(0,2)
Classes of Consistency Models

- Relaxed Consistency
  - Improves shared memory access performance while ensuring correctness to a specified consistency model
  - Any Memory Consistency model which is weaker than Sequential Consistency
    1. All accesses to synchronization variables are seen by all processes (or nodes, processors) in the same order (sequentially) - these are synchronization operations. Accesses to critical sections are seen sequentially.
    2. All other accesses may be seen in different order on different processes (or nodes, processors).
    3. The set of both read and write operations in between different synchronization operations is the same in each process.

Therefore, there can be no access to synchronization variable if there are pending write operations. And there can not be any new read/write operation started if system is performing any synchronization operation.
Motivations for Relaxed Models

- Limitations of Strict Consistency
  - Un-optimizable, detrimental to performance
  - Highly Restrictive
  - Not well suited for uniprocessor systems
    much less multiple processor systems
Motivations for Relaxed Models

- Limitations of Sequential Consistency
  - By preserving the sufficient conditions necessary to maintain sequential consistency the following critical performance optimizations cannot be made:
    - Code Motion
    - Common-subexpression elimination
    - Software pipelining
    - Register allocation
  - Simplicity is achieved at cost of efficiency: distributed systems with sequential consistency model are, without further optimization, such as speculation, one magnitude slower than those providing weaker models
Assumptions for Benchmarking

- Based on Stanford DASH multiprocessor
- Coherent caches, directory based invalidation scheme
- Latency = 1:25:100 processor cycles
- Detailed simulation, contention modeled
- 16 processors
Performance Benchmarks

- **MP3D**: 3-dimensional particle simulator
  - 10,000 particles, 5 time steps

- **LU**: LU-decomposition of dense matrix
  - 200x200 matrix

- **PTHOR**: logic simulator
  - 11,000 two-input gates, 5 clock cycles
Performance Evaluation: Control
Sequential Consistency

- Processor issues accesses one-at-a-time and stalls for completion

Low processor utilization (17% - 42%) even with caching
Requirements of a Relaxed Consistency Model

- Programmer’s Interface
- Translation Mechanism
- System Specification
Programmer’s Interface

- A contract such that if the program follows certain high-level rules or provides enough program annotations, then any system on which the program is run will always guarantee a sequentially consistent execution.
  - This execution is guaranteed regardless of the default reorderings permitted by the system specifications it supports.
Translation Mechanism

- This translates the programmer’s annotations to the interface (specifically, the order-preserving mechanisms) exported by the system specification, so that the system may do its job.
System Specification

- A clear specification of two things
  1. What program orders among memory operations are guaranteed to be preserved in an observable sense by the system, including whether or not write atomicity is maintained.
  2. If all program orders are not guaranteed to be preserved by default, then what mechanisms will the system provide for a programmer to enforce order explicitly when desired.
The System Specification

- **Three sets of models**
  1. **Write-to-Read Relaxation**
     - Only allow a read to bypass (complete before) an earlier incomplete write in program order
  2. **Write-to-Write Relaxation**
     - Allow writes to bypass previous writes
  3. **Complete Relaxation**
     - Allow reads \textit{or} writes to bypass previous reads as well
Relaxing the Write-to-Read Program Order

- Allows the hardware to hide the latency of write operations
  - While the write miss is still in the write buffer (and not yet visible to other processors) the processor can issue and complete reads that hit in its cache (or a single read which misses)
- Despite relaxing the write-to-read ordering, the programmer’s intuition is decently preserved in most implementations – Even without any special operations
Benefits of Relaxing the Write-to-Read Program Order

- Relaxing this order results in the hiding of write latencies and the benefits can be substantial.
- Most processors are capable of implementing and taking advantage of this relaxation
Performance Evaluation

Relaxed Write-to-Read Consistency

- Write latency is fully hidden
Relaxing Write-to-Write Program Order

- Allows writes to bypass earlier writes (to different locations)
  - This allows the write buffer to merge, and even retire, writes before previous writes in the program order become visible
- Reordering of writes can violate the intuitive nature of sequential consistency semantics
Benefits of Relaxing the Write-to-Write Program Order

- Enables multiple write misses to be fully overlapped and to become visible out of program order
- Further reduces the impact of write latency on processor stall time
- May improve communication efficiency
Performance Evaluation:

Relaxed Write-to-Write Consistency:

- Overlap of writes allows for faster synchronization on critical path
Relaxing All Program Orders

- No program orders are guaranteed by default other than data and control dependencies within a process.
- These models are particularly well matched to dynamically scheduled processors whose implementation allows them to proceed past read misses to other memory references.
- The only model which allows many of the key reorderings and eliminations of accesses necessary for many compiler optimizations.
Benefits of Relaxing All Program Orders

- Multiple read requests can be outstanding simultaneously
- These requests can also be bypassed by later writes in program order
- These writes can complete out of program order

Thus read and write latencies are hidden.
Performance Evaluation
Completely Relaxed Consistency

- Latency of reads can be hidden
- Currently studying compiler scheduling to exploit R-R,R-W overlap

BR: Blocking reads (W-R and W-W)
DSn: Dynamically Scheduled, window size n
## Characteristics of Various System Specifications

<table>
<thead>
<tr>
<th>Model</th>
<th>Write-to-Read Reorder</th>
<th>Write-to-Write Reorder</th>
<th>Read-to-Read/Write Reorder</th>
<th>Read Other's Write Early</th>
<th>Read Own Write Early</th>
<th>Operations for Ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>TSO</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>MEMBAR, RMW</td>
</tr>
<tr>
<td>PC</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>yes</td>
<td>MEMBAR, RMW</td>
</tr>
<tr>
<td>PSO</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>yes</td>
<td>STBAR, RMW</td>
</tr>
<tr>
<td>WO</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
<td>SYNC</td>
</tr>
<tr>
<td>RC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
<td>REL, ACQ, RMW</td>
</tr>
<tr>
<td>RMO</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
<td>various MEMBARs</td>
</tr>
<tr>
<td>Alpha</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
<td>MB, WMB</td>
</tr>
<tr>
<td>PowerPC</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td>yes</td>
<td>SYNC</td>
</tr>
</tbody>
</table>

Sparc V9 RMO, Digital Alpha, and IBM PowerPC are all processor specific implementations.
Weak Ordering (WO)

- Developed by Dubois, Scheurich, and Briggs in 1986
- Uses synchronization to protect critical sections
- Synchronization can be achieved through locks/unlocks or flags

```
<table>
<thead>
<tr>
<th>P1, P2, ..., Pn</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . .</td>
<td>TOP: while(flag2==0);</td>
<td>TOP: while(flag1==0);</td>
</tr>
<tr>
<td>Lock(TaskQ)</td>
<td>A = 1;</td>
<td>x = A;</td>
</tr>
<tr>
<td>newTask-&gt;next = Head;</td>
<td>u = B;</td>
<td>y = D;</td>
</tr>
<tr>
<td>if (Head != NULL)</td>
<td>v = C;</td>
<td>B = 3;</td>
</tr>
<tr>
<td>Head-&gt;prev = newTask;</td>
<td>D = B * C;</td>
<td>C = D / B;</td>
</tr>
<tr>
<td>Head = newTask;</td>
<td>flag2 = 0;</td>
<td>flag1 = 0;</td>
</tr>
<tr>
<td>UnLock(TaskQ)</td>
<td>flag1 = 1;</td>
<td>flag2 = 1;</td>
</tr>
<tr>
<td>. . .</td>
<td>goto TOP;</td>
<td>goto TOP;</td>
</tr>
</tbody>
</table>
```

(a)
Release Consistency (RC)

- Developed by Gharachorloo in 1990
- Developed because WO was not relaxing enough, divides synchronizations into acquires and releases
- Uses acquire (read) to delay memory accesses after it until acquire is completed
- Uses release (write) to allow access to new values of data modified before the release is completed
**FIGURE 9.6** Comparison of the weak ordering and release consistency models. The operations in block 1 precede the first synchronization operation, which is an acquire, in program order. Block 2 occurs between the two synchronization operations, and block 3 follows the second synchronization operation, which is a release.
Conclusion

- As with any system, the ideal is the goal.
- By allowing for weaker or relaxed memory models, system utilization is allowed to approach 100%
- Programmer has to pay for increased performance by dealing with looser constraints
Questions?

References:
- Parallel Computer Architecture by David E. Culler, Jaswinder Pal Singh, and Anoop Gupta
- Dr. Shaaban’s Lecture Notes
- http://projects.seas.gwu.edu/~hpcl/upcdev/tut/sld134.htm
- http://user.it.uu.se/~arnoldp/cse3ms/consist/
- http://phase.hpcc.jp/emx/isca95/tsld014.htm