Multiprocessing Issues

James Goodman

Readings


Recommended Readings

- Jon Stokes, Understanding CPU caching and performance, http://arstechnica.com/old/content/2002/07/caching.ars
- Lecture slides from “Advanced Caching Techniques” by Christos Kozyrakis (Lecture 9, 19Mar10)

TEST

- In-class
- Date: Thursday, 6May
- Open notes (no electronic devices)
- Required reading list online: CS703 “Resources”
For Today


Significance of Memory Order

- Observation: In a shared memory multiprocessor, writing to a memory does more than simply convey a value: it also implies a notion of time.
- Memory operations contain simple notion of time:
  - Writes are “events”
  - Reads are observations of events
- The rules regarding memory ordering are a “contract” between the hardware and the software.

Sequential Consistency

Leslie Lamport (1979):

“Sequential Consistency is satisfied 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.”

Simple definition, which may seem obvious to the novice parallel programmer (but not to the advanced programmer

Example

Initially, Flag1 = Flag2 = 0

P1:
Flag1 = 1;
If (Flag2 == 0)
critical section

P2:
Flag2 = 1;
if (Flag1 == 0)
critical section

If writes or reads can get out of order, both P1 and P2 can enter critical section
**Example 2**

Initially \( A = B = 0 \)

\[
P1: \quad P2: \quad P3: \\
A = 1 \quad if \ (A == 1) \quad B = 1 \quad if \ (B == 1) \\
\ \ \ \ \ \ \ \ \ \ register1 = A
\]

If \( P3 \) set \( register1 \) to 0, this implies that \( P2 \) set \( B=1 \) before \( P1 \) set \( A=1! \)


**Example 3**

Initially \( Data = 0; \ Flag = 0; \)

\[
P1: \quad P2: \\
Data = 1; \quad if \ (Flag == 1) \\
Flag = 1; \quad \ \ register = Data;
\]

This case is critical. If \( Flag \) gets propagated before \( Data \), how can we synchronize?

**How do memory operations get out of order?**

- Caches
- Processors perform operations out of order
- Write buffers with read bypassing
- Overlapping writes
- Nonblocking reads

**Questions to be Addressed**

Assuring that writes are atomic

- What constitutes a write?
- When is a write "committed"?
- What does "the most recent write" mean?

*Compilers must be careful in rearranging code*
Relaxing memory models

- Supporting SC is expensive! Writes in an MP may take a long time to “complete”
- How to relax model?
  - Push the problem to software: no implication about the order of memory operations. If you care, insert a “memory barrier.”
- Note: We still probably want to guarantee that writes to *the same* location obey sequential consistency. This is usually referred to as cache coherence.

Memory Barrier

- A memory barrier is a point in an execution when memory operations are ordered
  - All reads and writes occurring in program order before the point of the barrier must have completed before passing the barrier
  - No reads or writes occurring in program order after the barrier may be initiated before passing the barrier
- Note a memory barrier only applies to a single processor—other processors not executing a barrier may still observe these operations out of order

Weaker Ordering

- “Weak ordering” is imprecisely defined, but the weakest implies no ordering (except through memory barriers).
- Under some circumstances, this can even occur with a single processor, i.e., a Read-after-write hazard may be violated.

Intermediate Ordering

- Processor Consistency, Total Store Order (TSO)
  - Allows reads to bypass writes to other addresses
  - Does not allow writes to bypass other writes to any address
  - Does not allow reads to bypass reads to any other address
- This restriction is easier to implement, but avoids most synchronization problems.
  - Widely used (Intel, Sun)
Release Consistency

• Two barriers, Acquire & Release
  – On Acquire, may not proceed with any operations until all previous reads and writes are fully committed.
  – On Release, may not proceed with any operations until all writes have fully committed.
• This is claimed to be easier to support, reason about. Made available on newer processors (Intel Itanium)

Which is Best?

• Mark Hill: Processors should support simple memory-consistency models, IEEE Computer, 31(8), pp. 28-34, August 1998.
• Most of lost performance can be made up by speculation.
• Great benefits accrue by making parallel programming easier!

Litmust tests that differentiate

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>w(a=1)</td>
<td>r(b=1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>w(b=1)</td>
<td>r(a=0)</td>
<td></td>
</tr>
<tr>
<td>All but the weakest models disallow this.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>w(a=1)</td>
<td>w(b=1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>r(b=0)</td>
<td>r(a=0)</td>
<td></td>
</tr>
<tr>
<td>Processor consistency allows this</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>r(b=1)</td>
<td>r(a=1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>w(a=1)</td>
<td>w(b=1)</td>
<td></td>
</tr>
<tr>
<td>This is possible if writes can pass reads</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>w(a=1)</td>
<td>w(b=1)</td>
<td>r(a=1)</td>
</tr>
<tr>
<td></td>
<td>r(a=0)</td>
<td>r(b=0)</td>
<td></td>
</tr>
<tr>
<td>Write update protocols can easily allow this.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of Four Alternatives

1. Sequential Consistency
   Unambiguous, simple to understand, intuitive
   Obvious implementations create performance problems
2. Weak Ordering
   No constraints on memory ordering, except at barriers
   Tricky to reason about
   Memory barriers are potential performance problems
3. Processor Consistency
   Handles common cases without barriers
   Tricky to reason about cases where it fails
   Good performance implementations are subtle
4. Release Consistency
   Exploits knowledge about synchronization points
   Can be tricky to reason about
   Obvious implementations give good performance
Memory Ordering

• Why do memory operations get out of order?
• Compiler optimizations!
  – Compiler needs to know about shared variables and races
• Writes can be very slow
  – Uniprocessor: write is instantaneous
  – Multiprocessor: cache coherence may require multiple messages and memory accesses to assure that old write values are hidden.

Questions in 2010

• Was Hill correct about performance and aggravation? If so, why hasn’t the world changed to SC?
• In 1998, few foresaw multicores (Hill definitely did not). Did this affect Hill’s argument? Stronger? Weaker?

Multiprocessors Should Support Simple Memory-Consistency Models

• Written in 1998, but made predictions for the next ten years.
• Claim: Relaxing consistency buys some additional performance, but not enough for the aggravation