Outline

- Last two lectures: Messaging layers
  - low level: “RISC” approach to communication
    - Active Messages, Fast Messages
  - higher level: support for protection, user-level TLB, etc.
    - VMMC-2, U-Net/MM, VIA

- This lecture: Virtual shared memory
  - rationale: programmability
  - background: memory consistency models
    - what are they? why they matter?
  - background: software shared memory
    - issues: access control, protocol processing
    - page-based vs. object-based
  - case studies:
    - TreadMarks [ Fangzhe ]
    - AURC [ Yuanyuan ]
Supporting Shared Memory

- Rationale
  - Programmability
    - global address space permits uniform access to local, remote data
  - Simplifies implementation of higher-level programming languages
    - data-parallel (HPF), concurrent object-oriented languages (Java), others ...

- Challenges
  - architecture is not scalable
  - remote accesses cost much more than local
  - caches help, but how to keep them consistent?

Memory Consistency Models

- What value can a read return?

- Memory consistency model: A formal specification of how the memory system will appear to the programmer
  - places restrictions on the values that can be returned by a read
    - clarifies meaning of "read should return the value of the last write"
  - Cache coherence: Mechanism for implementing a consistency model
Sequential Consistency (Lamport’79)

[A multiprocessor system is \textit{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 appear in this sequence in the order specified by its program.

- Two aspects
  - program order among operations from individual processors
  - single sequential order among operations from all processors

\begin{align*}
\text{P1} & \quad \text{P2} \\
\text{Flag1} = 1 & \quad \text{Flag2} = 1 \\
\text{if } (\text{Flag2} == 0) & \quad \text{if } (\text{Flag1} == 0) \\
\text{critical section} & \quad \text{critical section} \\
\text{P1} & \quad \text{P2} \quad \text{P3} \\
A = 1 & \quad \text{if } (A == 1) & \quad \text{if } (B == 1) \\
B = 1 & \quad C = A
\end{align*}

Consistency Models: Weaker Variations

- Sequential consistency requires
  - \textit{program order}: completion of previous memory operations
  - \textit{write atomicity}: serialization of writes to the same location

<table>
<thead>
<tr>
<th>Processor Consistency (Goodman’89)</th>
<th>Weak Consistency (Dubois’90)</th>
<th>Release Consistency (Gharachorloo’90)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load</td>
<td>Acquire A</td>
<td>Acquire A</td>
</tr>
<tr>
<td>Load</td>
<td>LD/ST</td>
<td>LD/ST</td>
</tr>
<tr>
<td>Load</td>
<td>Release A</td>
<td>Release A</td>
</tr>
<tr>
<td>Store</td>
<td>LD/ST</td>
<td>LD/ST</td>
</tr>
<tr>
<td>Store</td>
<td>Release A</td>
<td>Release A</td>
</tr>
<tr>
<td>Load</td>
<td>Acquire B</td>
<td>Acquire B</td>
</tr>
<tr>
<td>Store</td>
<td>LD/ST</td>
<td>LD/ST</td>
</tr>
<tr>
<td>Load</td>
<td>Acquire B</td>
<td>LD/ST</td>
</tr>
<tr>
<td>Store</td>
<td>Release B</td>
<td>Release B</td>
</tr>
</tbody>
</table>

stores need not be performed in the same order on all processors

loads and stores between synchronization operations can be reordered
Consistency Models: Implementation Freedom

- **Sequential consistency**
  - reads stall for pending writes
  - writes considered complete on being written to write buffer

- **Processor consistency**
  - reads can bypass pending writes

- **Weak consistency**
  - reads can bypass writes, writes can be reordered in the write buffer
  - acquires stall for pending writes and releases
  - releases stall for pending writes
    - first read after release waits for release to perform

- **Release consistency**
  - reads can bypass writes *and pending releases*
  - processor issues acquire and stalls for acquire to perform
  - releases stall for pending writes and releases

---

Consistency Models: Performance Implications

[Figures 3 and 7 from paper]

![Performance Gain over BASE](image)

Weaker consistency models improve performance by overlapping write latency (SC), having read bypass writes (PC), and pipelining writes (WC, RC).

However, taking full advantage of weaker consistency models increases implementation complexity: *so the benefits must outweigh costs!*
Software Shared Memory

- Goal is to support a shared address space without incurring any of the hardware costs
  - cache remote data in main memory
  - check access privileges on each access
    - if sufficient, allow operation to proceed
    - else, perform coherence action to upgrade privileges
      - implemented using software corouting and messages

- Performance issues
  - overheads
    - access control
    - protocol processing
  - communication latency and bandwidth
  - interrupting the processor vs. polling
    - overhead vs. responsiveness

Software Shared Memory: Two Camps

<table>
<thead>
<tr>
<th>Page-based</th>
<th>Object-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granularity of sharing is a virtual-memory page (IVY: 1984)</td>
<td>Granularity of sharing is an user-defined object</td>
</tr>
<tr>
<td>Access control</td>
<td>Access control</td>
</tr>
<tr>
<td>- virtual-memory protection support</td>
<td>- fine-grained checks</td>
</tr>
<tr>
<td>- shared page in one of INV, RD_ONLY, or RD_WR modes</td>
<td>- binary rewriting (Blizzard, Shasta)</td>
</tr>
<tr>
<td>- page-fault handlers invoked on access violation</td>
<td>- aggregate multiple operations under a single check (Midway, CRL)</td>
</tr>
<tr>
<td>Primary sources of overhead</td>
<td>Primary sources of overhead</td>
</tr>
<tr>
<td>- false sharing</td>
<td>- access control</td>
</tr>
<tr>
<td>- network bandwidth</td>
<td>- fine-grained sharing</td>
</tr>
<tr>
<td>Solutions</td>
<td>- protocol processing overheads</td>
</tr>
<tr>
<td>- weaker consistency (TreadMarks)</td>
<td>- processor unresponsiveness</td>
</tr>
<tr>
<td>- hybrid HW/SW schemes (AURC)</td>
<td></td>
</tr>
</tbody>
</table>

Next lecture (3/25/1998)
Lecture Summary

- Memory consistency models
  - contract between the application and underlying system
    - clarifies what is meant by “reads return the value of the last write”
  - examples: sequential consistency, processor consistency, weak consistency, release consistency
    - weaker models overlap operations, reduce coherence traffic
    - however, introduce additional implementation complexity

- Page-based software shared memory
  - TreadMarks
    - aggressive implementation of lazy release consistency (LRC)
    - designed for loosely-coupled networks of workstations
      - trades off computation for communication
        - diffs, multiple-writer protocols, vector timestamps
  - AURC
    - hardware support (automatic update) simplifies LRC implementation
    - trades off network bandwidth for protocol complexity

Next Lecture

- Shared object memory and custom protocols
  - CRL: High-Performance All-Software Distributed Shared Memory, Johnson et al.
  - Towards Transparent and Efficient Software Distributed Shared Memory, Scales and Gharachorloo
  - Implementation and Performance of Munin, Carter et al.