Distributed Shared Memory: Treadmarks

CS 739
Spring 2012
Lecture 14

Questions from reviews

• Static # of nodes?
• Naming of addresses/address spaces?
• Failures?
• Finding pages/diffs?
• Scalability?
• Security?

Origins of the idea

• N-way Multiprocessors are more expensive than
  N uniprocessors
  – expensive interconnect networks
  – smaller base to amortize costs
• Shared memory programming is easier than
  message passing
  – Up for debate now
• Hypothesis: we run parallel programs on a
  network of workstations more cheaply than on
  an SMP

Technology Motivation

• Low-latency networks (ATM) became available
• High performance workstations (Sparc, Alpha)
  also available
• Non-cache-coherent supercomputers could
  also do coherence in software
  – Intel Paragon
  – Thinking Machines CM-5

Origins of the Idea

• Kai Li, 1986: Ivy
  – Page-based DSM – uses MMU
  – Sequential consistency – same as SMPs
  – Looked at page assignment
    • do pages have a “home”, and where
  – Page lookup
    • How do you find an up-to-date copy of a page?

Key concept

• SMP has shared physical memory
  – All processors can access the same DRAM
• DSM is shared virtual memory
  – OS coordinate access to provide illusion of shared
    physical memory
  – Can look like DRAM is a L-(2,3,4) cache of a larger
    address space
**Shared memory programming**

- Big idea is to program a distributed system using shared memory instead of messages
  - Is this easier?
  - When is it easier?
- Observation: most single-node parallel programs use shared memory instead of messages
- Observation: perhaps better when data structures involved

**Other alternatives**

- Remote reference
  - Provide commands to load/store to a remote machine
  - No cache coherence
- QUESTION: could you use a parallel programming language on Treadmarks?
  - If it is shared memory, probably yes, and if granularity matches

**Advantages of DSM**

- Normal shared memory programming techniques can be used
- Easily scalable, compared to traditional bus-connected shared memory multiprocessors
- Message passing is hidden from the user
- Can handle complex and large data bases without replication or sending the data to processes

**Disadvantages of DMS**

- Lower performance than true shared memory multiprocessor systems
- Must provide for protection against simultaneous access to shared data
  - Locks, etc.
- Little programmer control over actual messages being generated
- Incur performance penalties when compared to message passing routines on a cluster

**Follow-on work**

- Many DSM systems
  - Munin
  - Midway
  - Blizzard/Typhoon (UW)

**Page Based DSM System**
Issues in DSM

- **Granularity**
  - What is unit of coherence?
  - Page
  - Cache line
  - Word
  - Object

- **Consistency**
  - When is value of a write visible?
  - Immediately
  - After lock is acquired
  - After lock is released
  - After lock is released & acquired

- **Location**
  - How do I find data?
  - Using directory
  - Using home nodes

- **Protocol**
  - How do you keep the number of messages low?

- **Implementation**
  - Virtual memory – page faults
  - Binary instrumentation – edit instructions to perform access checks

Coherence methods

- Basic protocol: invalidation
  - Make sure only one copy of a piece of data is writable by taking “ownership” before writing

- Other option: update
  - Make changes locally and then send to other nodes
  - Benefit: avoids misses to fetch data after invalidation

- Challenge: when are the updates then visible to others?
  - Immediately: broadcast new data
  - Release: when lock released

Page ownership

- **Locating owners:**
  - Centralized: single node tracks owner of all pages
  - Distributed: ownership of different pages is tracked by different nodes
    - Fixed: mapping of addresses to directory is fixed
    - Dynamic: mapping of addresses to a directory is dynamic

- **Proposition**
  - How does keep the number of messages low
  - Implementations
    - Virtual memory range facility
    - Library instrumentation or add instructions to perform access checks

TreadMarks approach

- **User-mode only software**
  - No kernel modifications

- **Byte/word granularity**
  - No dependence on language-level objects (e.g. structs, arrays)

- **Uses VM hardware to detect reference to shared data**
  - mprotect() pages to invalid, read-only, or read/write

TreadMarks Implementation (I)

- **Three kinds of variables:**
  1. **Ordinary variables**: can only be accessed by the process that created them
  2. **Shared data variables**: should always be accessed from within critical regions
  3. **Synchronization variables**
    - locks, barriers or condition variables
    - must be accessed through special library procedures.

TreadMarks Implementation (II)

- When a processor modifies shared data inside a critical region, all update messages are buffered and delayed until the processor leaves the critical region

- Processes accessing shared data variables outside critical regions do it at their own risks
  - Same as with shared memory model
  - Risk is higher
Basic

- Allocate shared memory using Tmk_malloc
- On access, check if local page is valid
  - If not, contact remote machines to get page or diffs to apply to local page to make it valid
- Question:
  - What if tmk_malloc() data points to local data?

Consistency in Treadmarks

- Consider two threads accessing shared data
  - Thread 1: lock(m); write(x); unlock(m)
  - Thread 2: lock(m); does it need locks?
    - Answer: for correct synchronization, it does
  - Thread 3: lock(m); read(x); unlock(m)
  - Thread 4: read(local-z)
- When does the write to x need to be visible?
  - Immediately?
  - When lock m is released?
  - When lock m is acquired?
- To whom is the write to x visible
  - Everybody?
  - Thread 1? Thread 2? Thread 3?

Lazy release consistency

- Observation: correctly synchronized programs don’t have data races
  - All access to shared state are ordered with Lamport’s happens-before and synchronization instructions
    - Locks, barriers
  - Two conflicting accesses to a variable by different threads must have a sync operation between the to specify the order
- Conflicts = 2 accesses, one is a write

Is LR-C a good idea

- Is it important to support buggy programs?
  - The alternative approaches (sequential consistency) were many times slower
  - Benign races?

LR-C

- Updates are only “Tracked” while holding a lock
  - assumes no shared data written without lock
- Updates made while holding one lock are propagated to the next holder of the lock
  - Not known until lock acquired, so …
- Updates are propagated from releaser to acquirer of a lock and acquire time

Eager Vs. Lazy RC
### Example

- Thread 1: lock(m); write(x); unlock(m);
- Thread 2: lock(m); read(x); unlock(m);
  - Thread 1 remembers the x was written
  - Invalidates X on other processors
  - When thread 2 acquires m, it must contact thread 1 to get the lock. It goes back to thread 1 to get any pages invalidated

### Problems with LR-C

- Publication
  - Write an object privately
  - Acquire lock
  - Add to list
  - Release lock
- In LR-C, writes to object occur without lock, are never propagated

### Multiple Writers

- What if two language-objects reside on the same page?
  - If page-based coherence, have false sharing:
    - access to one will invalidate access to the other
    - pages ping-pong back and forth between processors

### WRITE-SHARED PROTOCOL (I)

- Designed to fight false sharing
- Uses a copy-on-write mechanism
- Whenever a process is granted access to write-shared data, the page containing these data is marked copy-on-write
- First attempt to modify the contents of the page will result in the creation of a copy of the page modified (the twin).

### WRITE-SHARED PROTOCOL (II)

- At release time, the DSM will perform a word by word comparison of the page and its twin, store the diff in the space used by the twin page and notify all processors having a copy of the shared data of the update
- A runtime switch can be set to check for conflicting updates to write-shared data.

---

**Example**

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1</td>
<td>x = 3</td>
</tr>
<tr>
<td>y = 2</td>
<td>y = 2</td>
</tr>
</tbody>
</table>

**First write access**

<table>
<thead>
<tr>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 1</td>
<td>x = 1</td>
</tr>
<tr>
<td>y = 2</td>
<td>y = 2</td>
</tr>
</tbody>
</table>

**Compare with twin**

<table>
<thead>
<tr>
<th>New value of x is 3</th>
<th>twin</th>
</tr>
</thead>
</table>
Treadmarks solution

- Assume program is correctly synchronized
  - 2 threads holding different locks will not update the same range of bytes
- Track byte-level modification to pages
  - On write, create a twin page. On release, diff original page and twin to create a list of bytes that changed
  - On fault of invalid page, get diffs from all nodes that wrote to it

Multiple-writers example

- `int x,y; // on same page`
- `Thread 1: lock(m1); write(x); unlock(m1);`
  - Invalidates other copies
- `Thread 2: lock(m2); write(y); unlock(m2);`
  - Concurrent; QUESTION: How get a page here?
    - non-blocking: copies page from thread 1
- `Thread 3: lock(m1); read(x); unlock(m1);`
  - Gets diffs from thread 1, thread 2

Persistent Challenges

- Hot pages cause a lot of coherence traffic
  - NOTE: Same is true within a machine
  - QUESTION: what can be done?
    - ANSWER: rewrite application (data partitioning)
- Fine-grained vs coarse grained
  - Fine grained may work on reliable, fast network (e.g. TM CM-5)
  - Coarse grained only possibility for workstations
- Fault tolerance
  - People have combined with STM
- SMP nodes
  - only a load balance problem, but system still works (as it operates on VM)

Kai Li’s take on DSM

- As a product/feature, it went nowhere
  - Hard to reason about failure
  - Works best for coarse-grained programs, which aren’t that hard to write in other ways
  - Overheads are pretty high
- As a test bed, it was useful
  - Developed novel consistency semantics (lazy release consistency)
  - Scalable coherence protocols

Willy Zaenepoel’s Take

- DSM and P2P (and probably TM) are cousins
  - High implementation complexity leads to lots of papers
  - Research drove towards fine-grained DSM (see Shasta), but fine-grained inherently performs poorly on a cluster
    - More problems with fine grained, so more solutions and more papers
- Reality: DSM only works for coarse grained data, large chunks of contiguous data