Administrivia

Problem Set 1, due Wednesday at the *beginning* of class

Reading for Wednesday:

- Falsafi et al.: Customized Shared-Memory Protocols
- von Eicken et al.: Active Messages on ATM

Outline

Deadlock Review

Typhoon

SHRIMP

Coprocessor Comparison

Fetch Deadlock

Request & *reply* networks (CM-5)

- Key: replies can always be handled (don't use memory)
- Request depend only on replies ⇒ no deadlock

Problems:

- Must poll two locations (one for each network)
- Network complexity... networks must be independent!
- Requests can't generate requests

Shared-Memory Deadlock

Shared-memory protocols have all the same problems

Deadlock 1: What about request-request deadlock?

- FLASH: ⇒ send NACK to original sender, to cause retry
- Alewife: Drain deadlocked messages to main memory
- CS294-5: send new request back to sender (via reply)

Deadlock 2: Handlers can't access remote memory!

- Handler blocks on outstanding remote read
- New source of deadlock (draining won't help)

Deadlock due to Integration

Deadlock 3:

- ◆ Processor A issues a load, which ties up the bus
- ◆ The network input queue contains a message
- ◆ The data for the load arrives, but is enqueued after the message
- ◆ ⇒ Can't complete load until message is handled

FLASH solution:

- ◆ MAGIC replaces bus (this is required!)
- ◆ MAGIC has *two* ports ⇒ can handle incoming traffic while load ties up the other port

Alewife solution: High-availability interrupts

- ◆ Interrupt the load to handle the messages
- ◆ Executing handlers may invalidate the load!

Alewife Summary

Network Interface:

- ◆ Unified messages for shared-memory and message passing
  ⇒ software can generate arbitrary protocol messages
- ◆ Integration of DMA with active messages
- ◆ Automatic draining to avoid deadlock ⇒ fewer restrictions on handlers
- ◆ 2D mesh with dimension-order routing
- ◆ User-level message passing

No support for virtual memory

Biggest advantage: already exists with solid development environment

FLASH Summary

Commodity processor

MAGIC chip controls all memory access

Handlers execute on MAGIC:

- ◆ No user-level handlers
- ◆ Handlers obey conventions to avoid deadlock

MAGIC performs block transfers by cache line:

- ◆ ⇒ can get current version before sending
- ◆ Transfers are automatically interleaved:
  After k cache lines, transfer goes to the end of the queue
- ◆ Real virtual memory

Tempest & Typhoon

Similar to FLASH, but user-level handlers/protocols

Their claim: user-level protocols lead to significant performance improvements (35%)

Problem: (with their example)

- ◆ They tweak the protocols to act like application-specific message passing
- ◆ *Then* claim shared memory is simpler than message passing...

Typhoon Issues

Deadlock: Users' problem... just like the CM-5

- Request-Reply networks
- Use gang scheduling
- Drain the network at context switches (isolate users)

Block transfer:

- NP packetizes blocks into 64-byte packets
- Gather/Scatter can be implemented (kind of)
- Packets are interleaved with other traffic

Page faults on coprocessor:

- Prohibited: pages must be pinned
- $\Rightarrow$ difficult to implement scatter (covers too many pages)

Typhoon: Fine-Grain Access Control

2 bits per 32-byte block (memory line)

- States: ReadWrite, ReadOnly, Invalid, Busy

Key implication:

Can interrupt on any *access* to a location

- $\Rightarrow$ arbitrary code to handle that access
  - Add full/empty bits using different storage
  - Implement atomic operations (fetch-and-*op*)
- Also useful for debugging: tracing a variable

Extends virtual memory tricks to memory lines:

- Block-access handler replace page-fault handler
- Examples: stack overflow, migrated objects

Shared-Memory vs. Message Passing

General claims for shared memory:

- Simpler than message passing
- Avoids complicated planning and data layout
- Faster for fine-grain access (one instruction)

When is shared-memory simpler?

When is it faster?

Producer-Consumer Sharing

A produces a value, v, used by B

Message passing: send one active message to B

Shared memory: (common case)

- B reads v before it is ready
- A writes $v \Rightarrow$ invalidate B, wait for ack
- B reads v again $\Rightarrow$ second round trip

How does B know when to read v?

What about prefetching?

One solution: *update* protocols

Update Protocols

Idea: A writes v in place at B

B always has the local copy

A always writes through to B
 ⇒ Useful to minimize writes via a temp. variable

Problems:

- Still can't tell when data reaches B
  Requires separate synchronization (e.g. barrier)
- Update protocols can lead to weaker consistency
  (but not for common cases)
- Want most writes to invalidated remote copies
  ⇒ two kinds of write instructions

Typhoon's Protocol

Idea: new user-level protocol for EM3D

Advantages:

- Uses explicit update commands
  ⇒ Exactly 1 write from A to B
- Uses presence counters at receive for synchronization
  (just like counters for active-message block transfers)

Nearly identical to message passing code, except:

- Message passing code is simpler to implement
- SM requires 10x more memory for caching
- MP is slightly faster (with better cache behavior!)
- MP requires preprocessing step to perform reasonable data layout, but EM3D already requires such a step

SHRIMP

Goal: user-level communication with virtual memory

Two nice properties:

- Cleanly handles pinning problem
- Writes to "remote" pages are automatically converted to packets

Key idea: set up maps of local pages to remote pages

- Segments (again)

Two problems with segments:

- Can't build active messages (no N-to-1 enqueue)
- Have to pin the receiving pages...

Handling Unpinned Pages

Receiver can unpin an incoming segment:

- First, tell sender page is unpinned (1 round trip)
- Sender marks segment as read only
- Write causes page fault
- Page fault causes round trip to repin the page
- Page fault handler then retries the write

This is heavyweight, but it works

- Comparable cost to regular page eviction

Relationship to Network RAM?

Complexity

What did they leverage?

Project	Leveraged
Alewife	Modified Sparc core, routers, new CMMU, new OS
FLASH	MIPS T5, new bus arbitration, new coprocessor, new OS
Tempest	SuperSparc processor, CM-5 network, new Sparc-based NI, new OS
SHRIMP	Pentium, Paragon backplane, modified OS, new NI
NOW	Workstations, modified OS, networks, new NI?

Multiple Users

Project	Sharing
Alewife	Single user, no virtual memory
FLASH	Fine-grain protection, virtual memory, multiple simultaneous users
Tempest	Fine-grain protection, virtual memory, gang scheduling
SHRIMP	Page-level protection, virtual memory, multiple simultaneous users
NOW	Page-level protection + software fault isolation, virtual memory, gang scheduling

The Real Question

What should a coprocessor do?