CS 258
Parallel Computer Architecture

LimitLESS Directories: A Scalable Cache Coherence Scheme

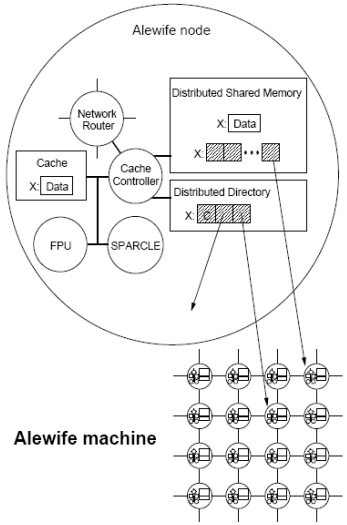
David Chaiken, John Kubiatowicz,
and Anant Agarwal

Presented:
March 19, 2008
Ankit Jain

The Background & Problems

- Bus-Based Protocols
  - Do not scale because broadcasts are slow and limit parallelism
- Traditional Directory-Based Protocols
  - Monolithic Directories
    - » Implicitly serialize all memory requests
  - Directory Accesses consume a disproportionately large fraction of available network bandwidth
  - Full Directories are Large
    - » Full Map Size: Total Memory Size * Number of Processors
  - Limited Directory Protocols
    - » Allowing a limited number of simultaneous cached copies of any block of data
    - » Pro: Size of directory is smaller
    - » Con: Potential Thrashing since eviction and reassignment when more simultaneous copies needed
    - » Previous studies show small set of pointers is sufficient to capture *worker-set* of processors

Alewife Architecture



- Cost Effective Mesh Network
  - Pro: Scales in terms of hardware
  - Pro: Exploits Locality
- Directory Distributed along with main memory
  - Bandwidth scales with number of processors
- Con: Non-Uniform Latencies of Communication
  - Have to manage the mapping of processes/threads onto processors due
  - Alewife employs techniques for latency minimization and latency tolerance so programmer does not have to manage
- Context Switch in 11 cycles between processes on remote memory request which has to incur communication network latency
- Cache Controller holds tags and implements the coherence protocol

LimitLESS Protocol + Requirements

- *Limit*ed Directory that is *L*ocally *E*xtended through *S*oftware *S*upport
- Handle the common case (small worker set) in hardware and the exceptional case (overflow) in software
- Processor with rapid trap handling (executes trap code within 5-10 cycles of initiation)
- State Shared
  - Processor needs complete access to coherence related controller state in the hardware directories
  - Directory Controller can invoke processor trap handlers
- Machine needs an interface to the network that allows the processor to launch and intercept coherence protocol packets

The Protocol

Type	Symbol	Name	Data?
Cache to Memory	RREQ	Read Request	
	WREQ	Write Request	
	REPM	Replace Modified	√
	UPDATE	Update	√
	ACKC	Invalidate Ack.	
Memory to Cache	RDATA	Read Data	√
	WDATA	Write Data	√
	INV	Invalidate	
	BUSY	Busy Signal	

Table 1: Protocol messages for hardware coherence.

Component	Name	Meaning
Memory	Read Only	Some number of caches have read only copies of the data.
	Read Write	Exactly one cache has a read write copy of the data.
	Read Transaction	Holding read request, update is in progress.
	Write Transaction	Holding write request, invalidation is in progress.
Cache	Invalid	Cache block may not be read or written.
	Read Only	Cache block may be read, but not written.
	Read Write	Cache block may be read or written.

Table 2: Directory states.

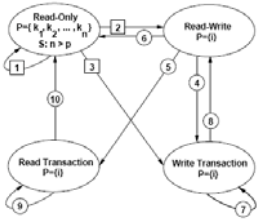


Figure 2: Directory state transition diagram.

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$j \rightarrow$ WREQ	$P = \{i\}$	$P = \{j\}$; AckCtr = 1	INV $\rightarrow i$
8	$j \rightarrow$ ACKC	AckCtr = 1, $P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$

Note: In the Read-Only State, the notation S: n>p indicates that the outputs from the state are handled through a software interrupt handler if the size of the pointer set (n) is greater than the size of the limited directory (p).

An Example

- Proc *i* has data block *D* from Proc *d* in Read-Write State
- Proc *j* wants to write a value to data block *D*

Processor i

Data Block	State
d	Read-Write

Processor j

Data Block	State
d	Invalid

Processor d Directory Entry

Data Block	State	AckCtr	Owning Processors
d	Read-Write	0	i

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$j \rightarrow$ WREQ	$P = \{i\}$	$P = \{j\}$; AckCtr = 1	INV $\rightarrow i$
8	$j \rightarrow$ ACKC	AckCtr = 1, $P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$

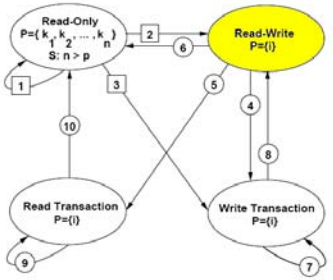


Figure 2: Directory state transition diagram.

An Example

- Proc *i* has data block *D* from Proc *d* in Read-Write State
- Proc *j* wants to write a value to data block *D*

Processor i

Data Block	State
d	Read-Write

Processor j

Data Block	State
d	Invalid

Processor d Directory Entry

Data Block	State	AckCtr	Owning Processors
d	Read-Write	0	i

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$j \rightarrow$ WREQ	$P = \{i\}$	$P = \{j\}$; AckCtr = 1	INV $\rightarrow i$
8	$j \rightarrow$ ACKC	AckCtr = 1, $P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$

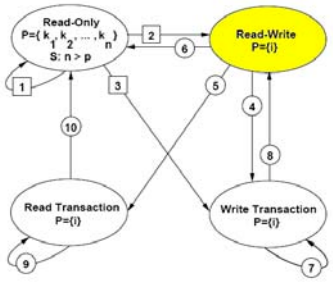


Figure 2: Directory state transition diagram.

j → WREQ

Precondition: P = { / }

INV → i

An Example

- Proc *i* has data block *D* from Proc *d* in Read-Write State
- Proc *j* wants to write a value to data block *D*

Processor i

Data Block	State
d	Invalid

Processor j

Data Block	State
d	Invalid

Processor d Directory Entry

Data Block	State	AckCtr	Owning Processors
d	Read-Write	1	j

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$j \rightarrow$ WREQ	$P = \{i\}$	$P = \{j\}$; AckCtr = 1	INV $\rightarrow i$
8	$j \rightarrow$ ACKC	AckCtr = 1, $P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{i\}$	AckCtr = 0	WDATA $\rightarrow i$

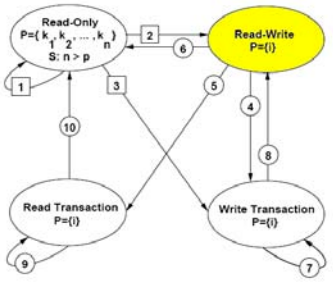


Figure 2: Directory state transition diagram.

An Example

- Proc *i* has data block *D* from Proc *d* in Read-Write State
- Proc *j* wants to write a value to data block *D*

Processor i

Data Block	State
d	Invalid

Processor j

Data Block	State
d	Invalid

Processor d Directory Entry

Data Block	State	AckCtr	Owning Processors
d	Read-Write	1	j

AckCtr = 1, P = { j }

i → ACKC

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$j \rightarrow$ WREQ	$P = \{z\}$	$P = \{j\}$; AckCtr = 1	INV $\rightarrow z$
8	$j \rightarrow$ ACKC	AckCtr = 1, $P = \{z\}$	AckCtr = 0	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{z\}$	AckCtr = 0	WDATA $\rightarrow i$

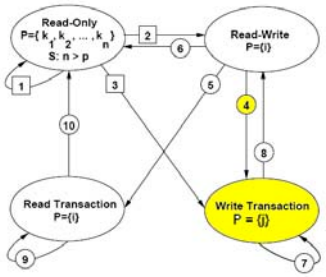


Figure 2: Directory state transition diagram.

An Example

- Proc *i* has data block *D* from Proc *d* in Read-Write State
- Proc *j* wants to write a value to data block *D*

Processor i

Data Block	State
d	Invalid

Processor j

Data Block	State
d	Read-Write

Processor d Directory Entry

Data Block	State	AckCtr	Owning Processors
d	Read-Write	0	j

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$j \rightarrow$ WREQ	$P = \{z\}$	$P = \{j\}$; AckCtr = 1	INV $\rightarrow z$
8	$j \rightarrow$ ACKC	AckCtr = 1, $P = \{z\}$	AckCtr = 0	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{z\}$	AckCtr = 0	WDATA $\rightarrow i$

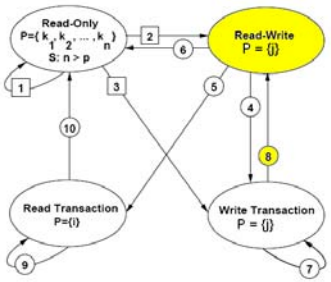
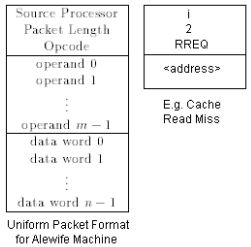


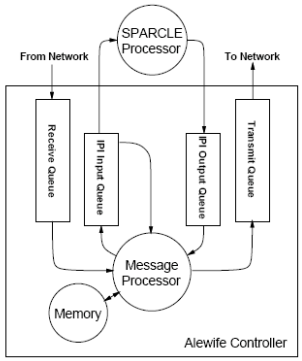
Figure 2: Directory state transition diagram.

Interprocessor-Interrupt (1/2)



Uniform Packet Format for Alewife Machine

E.g. Cache Read Miss



- Trap routine can either discard packet or store it to memory
- Store-back capability permits message-passing and block transfers
- Potential Deadlock Scenario with Processor Stalled and waiting for a remote cache-fill
  - Solution: Synchronous Trap (stored in local memory) to empty input queue

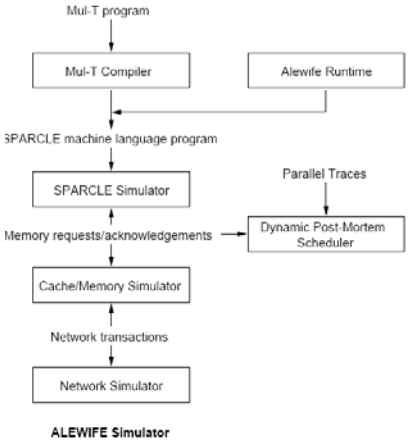
Interprocessor-Interrupt (2/2)

Meta State	Description
Normal	Directory handled by hardware.
Trans-In-Progress	Interlock, software processing.
Trap-On-Write	Trap: WREQ, UPDATE, REPM.
Trap-Always	Trap: all incoming packets.

Table 4: Directory meta states for LimitLESS scheme.

- Overflow Trap Scenario
  - First Instance: Full-Map bit-vector allocated in local memory and hardware pointers emptied into this and vector entered into hash table
  - Otherwise: Empty hardware pointers into bit vector
  - Meta-State Set to "Trap-On-Write"
  - While emptying hardware pointers, Meta-State: "Trans-In-Progress"
- Incoming Write Request Scenario
  - Empty hardware pointers to memory
  - Set AckCtr to number of bits that are set in bit-vector
  - Send invalidations to all caches except possibly requesting one
  - Free vector in memory
  - Upon invalidate acknowledgement (AckCtr == 0), send Write-Permission and set Memory State to "Read-Write"

Performance Technique



Application	Dir_4NB	LimitLESS$_4$	Full-Map
Multigrid	0.729	0.704	0.665
SIMPLE	3.579	2.902	2.553
Matexpr	1.296	0.317	0.171
Weather	1.356	0.654	0.621

Table 5: Performance for three coherence schemes, in terms of millions of cycles.

Notes:

- **Multigrid: Small worker sets → limited directories perform as well as full map**
- **SIMPLE implemented barrier synchronization with single lock**
- **Matexpr has worker sets up to 16 processors**
- **Weather has one variable initialized by one processor and then read by all the other processors**

Results (1/3)

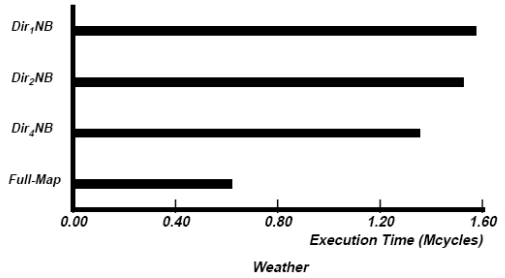


Figure 7: Limited and full-map directories.

Results (2/3)

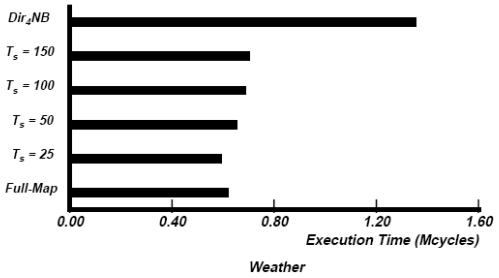


Figure 8: LimitLESS, 25 to 150 cycle emulation latencies.

Results (3/3)

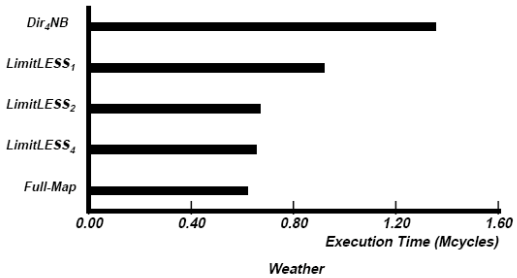


Figure 9: LimitLESS with 1, 2, and 4 hardware pointers.

Summary

- LimitLESS directories can closely emulate Full-Map Directories while saving hardware resources
- LimitLESS is not as sensitive to tuning parameters as the Limited Directory approach
- The protocol is general enough to apply to other coherence techniques
- In the future, it can be extended to give feedback to programmers/compilers about hot-spots, etc

Full Memory State Transition Diagram

Transition Label	Input Message	Precondition	Directory Entry Change	Output Message(s)
1	$i \rightarrow$ RREQ	—	$P = P \cup \{i\}$	RDATA $\rightarrow i$
2	$i \rightarrow$ WREQ	$P = \{i\}$	—	WDATA $\rightarrow i$
	$i \rightarrow$ WREQ	$P = \{\}$	$P = \{i\}$	WDATA $\rightarrow i$
3	$i \rightarrow$ WREQ	$P = \{k_1, \ldots, k_n\} \wedge i \notin P$	$P = \{i\}$, AckCtr $= n$	$\forall k_j$ INV $\rightarrow k_j$
	$i \rightarrow$ WREQ	$P = \{k_1, \ldots, k_n\} \wedge i \in P$	$P = \{i\}$, AckCtr $= n - 1$	$\forall k_j \neq i$ INV $\rightarrow k_j$
4	$j \rightarrow$ WREQ	$P = \{i\}$	$P = \{j\}$, AckCtr $= 1$	INV $\rightarrow i$
5	$j \rightarrow$ RREQ	$P = \{i\}$	$P = \{j\}$, AckCtr $= 1$	INV $\rightarrow i$
6	$i \rightarrow$ REPM	$P = \{i\}$	$P = \{\}$	—
7	$j \rightarrow$ RREQ	—	—	BUSY $\rightarrow j$
	$j \rightarrow$ WREQ	—	—	BUSY $\rightarrow j$
	$j \rightarrow$ ACKC	AckCtr $\neq 1$	AckCtr $=$ AckCtr $- 1$	—
	$j \rightarrow$ REPM	—	—	—
8	$j \rightarrow$ ACKC	AckCtr $= 1$, $P = \{i\}$	AckCtr $= 0$	WDATA $\rightarrow i$
	$j \rightarrow$ UPDATE	$P = \{i\}$	AckCtr $= 0$	WDATA $\rightarrow i$
9	$j \rightarrow$ RREQ	—	—	BUSY $\rightarrow j$
	$j \rightarrow$ WREQ	—	—	BUSY $\rightarrow j$
	$j \rightarrow$ REPM	—	—	—
10	$j \rightarrow$ UPDATE	$P = \{i\}$	AckCtr $= 0$	RDATA $\rightarrow i$
	$j \rightarrow$ ACKC	$P = \{i\}$	AckCtr $= 0$	RDATA $\rightarrow i$

Table 3: Annotation of the state transition diagram.