Transparent Shared Memory

on

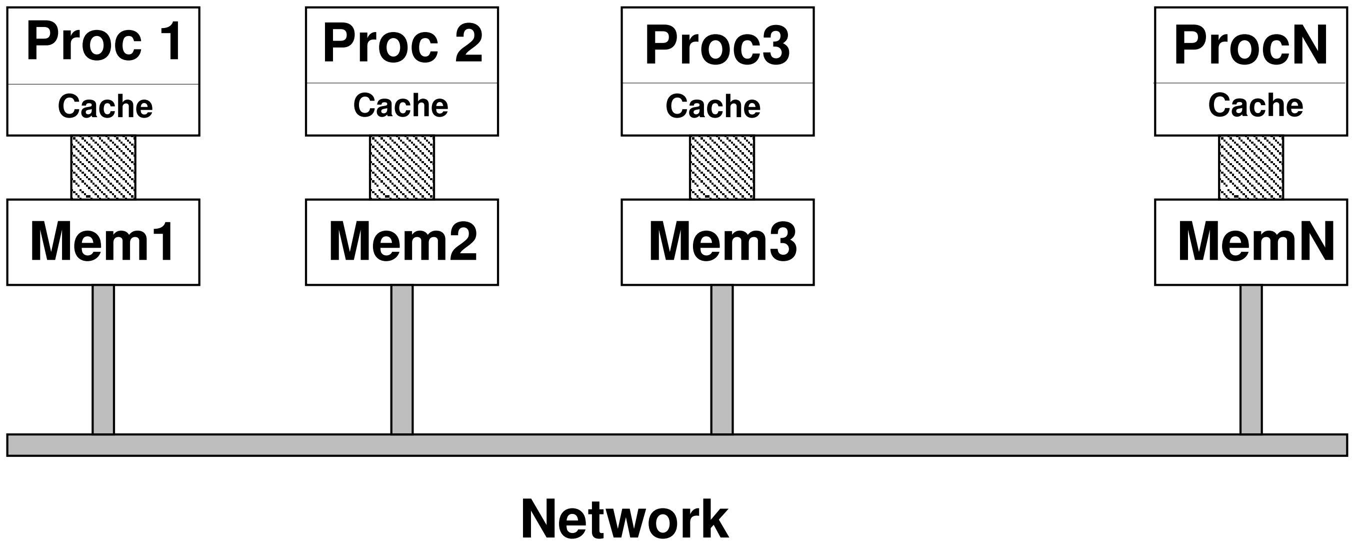
Clusters of Workstations

Sandhya Dwarkadas
Department of Computer Science
University of Rochester

Joint work with Michael L. Scott,
Robert Stets, Galen Hunt, Srinivasan Parthasarathy, Umit Rencuzogullari, Rajeev Balasubramonian, DeQing Chen, and Grigorios Magklis, University of Rochester
Nikolaos Hardavellas and Leonidas Kontothanassis (DEC)

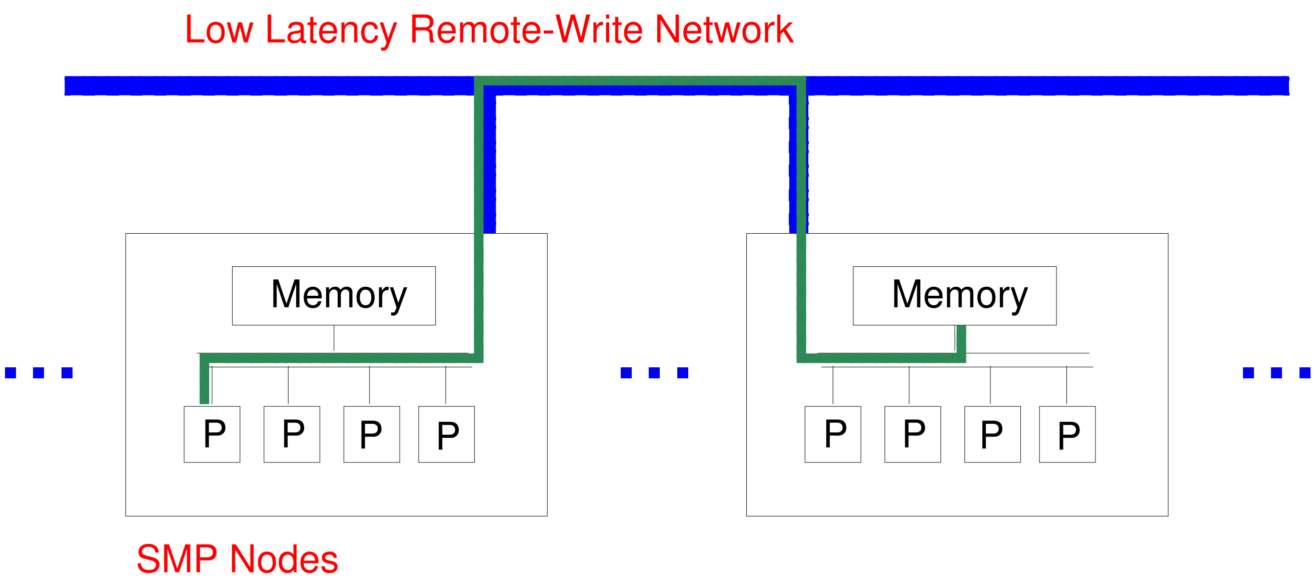
Distributed Memory Hardware (DM)

Parallel computing on clusters of workstations



All commodity technology (including network), thus cheap

Cost-Effective Parallel Systems



Motivation: Need for a uniform programming model

Shared memory easier to program and exploits hardware coherence within node

Goals

Maximize use of hardware coherence within nodes

Decrease software protocol overhead by exploiting network's capabilities

Techniques

Reduce protocol synchrony

- Reduce inter-processor synchronization
- Minimize global locks inside protocol

Maximize use of network capabilities
- broadcast, remote-write, and total order features for

- shared data
- protocol meta-data
- application synchronization

Overview

Programming model: shared memory

Remote-write networks: Memory Channel

Protocol description: Cashmere

SMP optimizations: Cashmere-2L

Performance results

Conclusions and ongoing work

Shared Memory Programming Model

Shared address space:

- Processes
- Shared memory allocation
- Synchronization
  - Locks
  - Barriers
  - Flags

Why Shared Memory?

Easier to go sequential → parallel

Easier to utilize multi-paradigm platform

Sequential =
single thread + single address space

Shared memory =
multiple threads + single address space

Message passing =
multiple threads + multiple address spaces

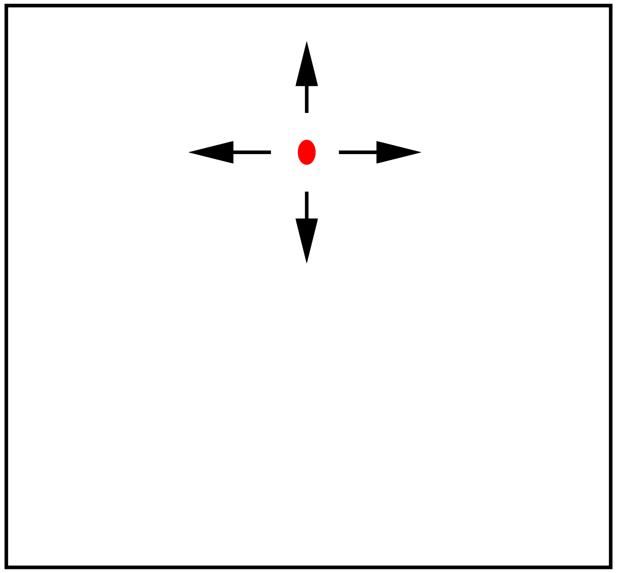
Example Program - Jacobi

```
do k = 1, 100
    do j = 1, M
        do i = 2, M-1
            a(i,j) = (b(i-1,j)+b(i+1,j)
                      +b(i,j-1)+b(i,j+1))/4
        enddo
    enddo

    do j = 1, M
        do i = 1, M
            b(i,j) = a(i,j)
        enddo
    enddo
enddo
```

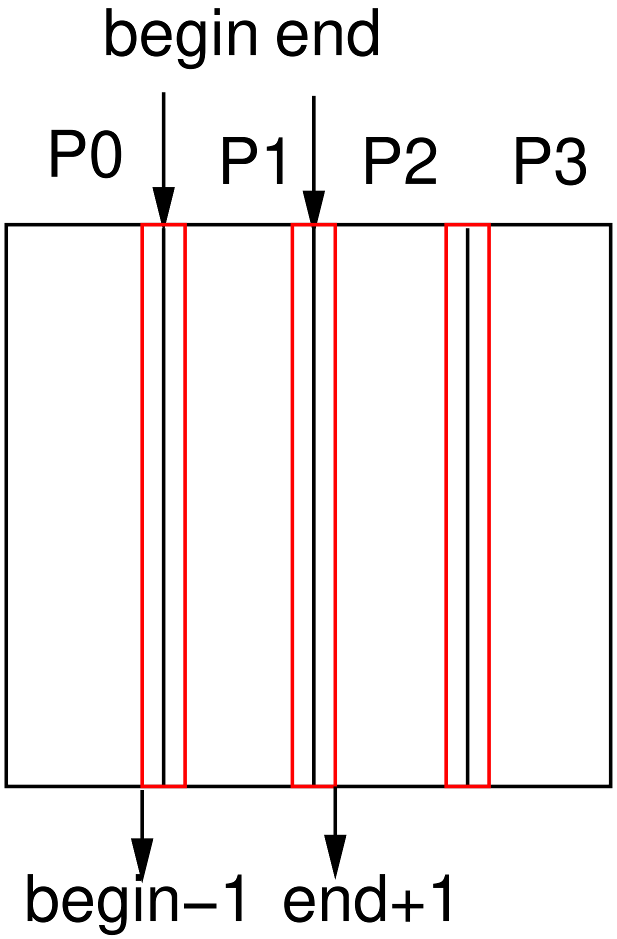
Jacobi Parallelization

Computation



$$a(i,j) = \mathbf{F}(b(i-1,j), b(i+1,j), b(i,j-1), b(i,j+1))$$

Parallelization



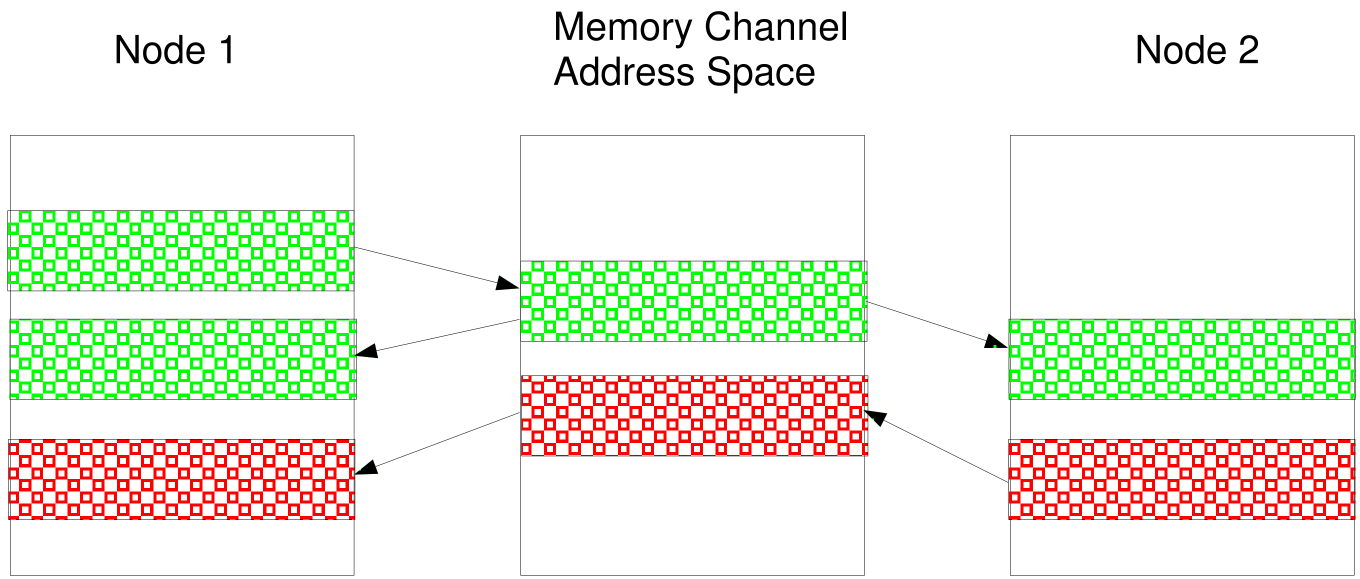
Shared Memory Version

```
do k = 1, 100
    do j = begin, end
        do i = 2, M-1
            a(i,j) = (b(i-1,j)+b(i+1,j)
                       +b(i,j-1)+b(i,j+1))/4
        enddo
    enddo
    call barrier()
    do j = begin, end
        do i = 1, M
            b(i,j) = a(i,j)
        enddo
    enddo
    call barrier()
enddo
```

Message Passing Version of Jacobi

```
do k = 1, 100
    do j = begin, end
        do i = 2, M-1
            a(i,j) = (b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))/4
        enddo
    enddo
    do j = begin, end
        do i = 2, M-1
            b(i,j) = a(i,j)
        enddo
    enddo
    if (p .neq. (nprocs - 1)) then
        send(p+1, b(end-1))
        recv(p+1, b(end))
    endif
    if (p .neq. 0) then
        send(p-1, b(begin))
        recv(p-1, b(begin-1))
    endif
enddo
```

Memory Channel



Create transmit and receive segments with user-level access

Write to transmit segment sent to all recv segments with same id

Total ordering of writes - allows simple implementation of synchronization

Cashmere - Design Principles

Virtual memory based software distributed shared memory (SDSM)

Moderately lazy release consistency

Multiple writer protocol

Distributed directory-based coherence

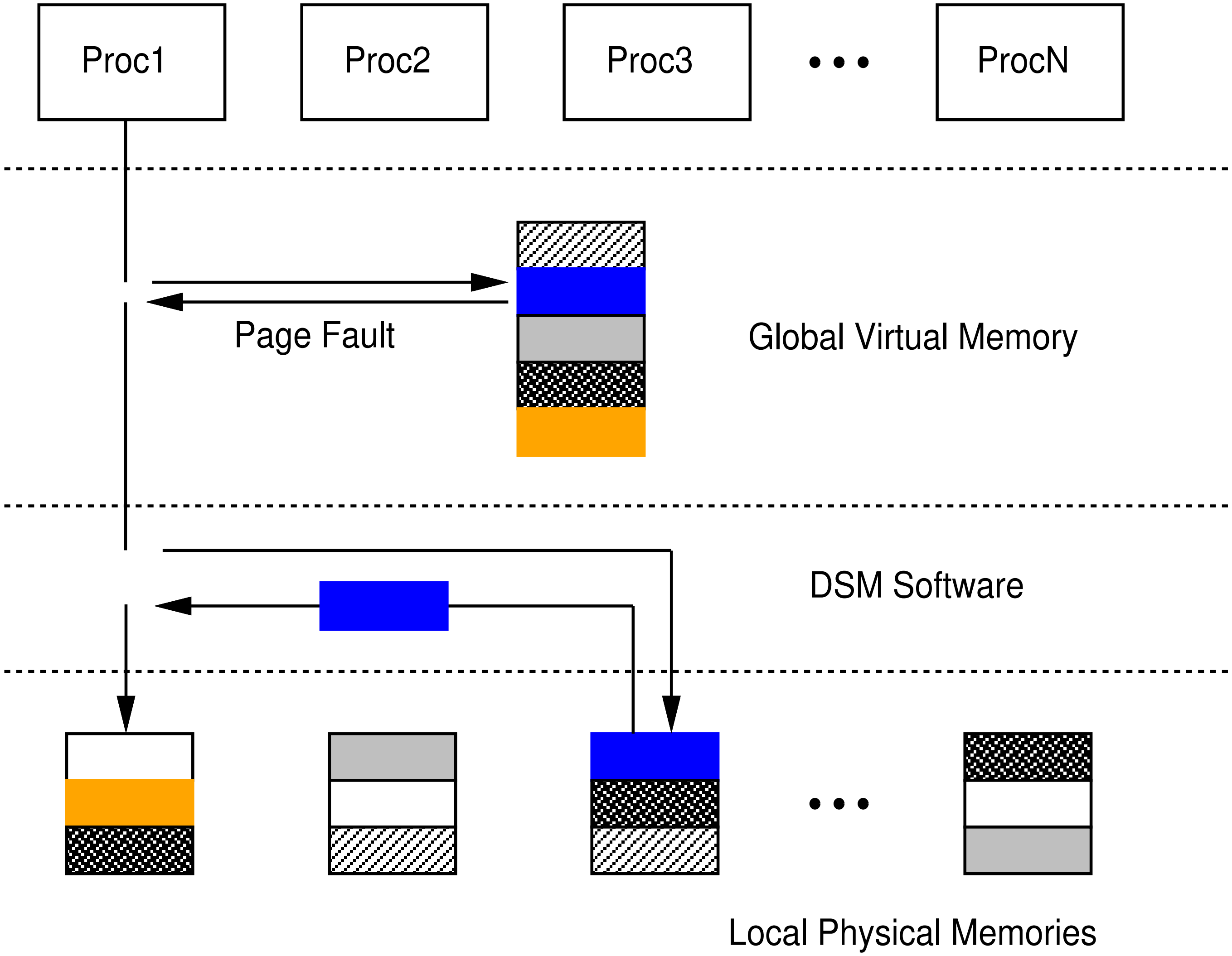
Home nodes

Page-based Coherence

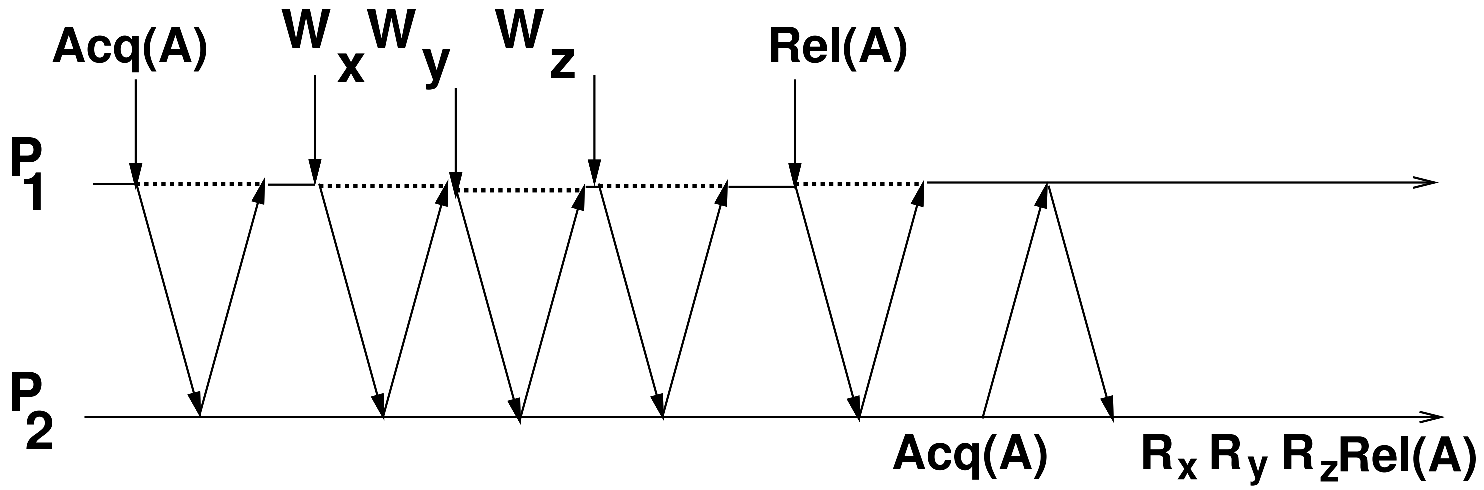
Utilizes underlying virtual memory hardware to detect shared memory modifications

Advantages: No software overheads in the absence of sharing

Conventional Page-based SDSM [Li 86]

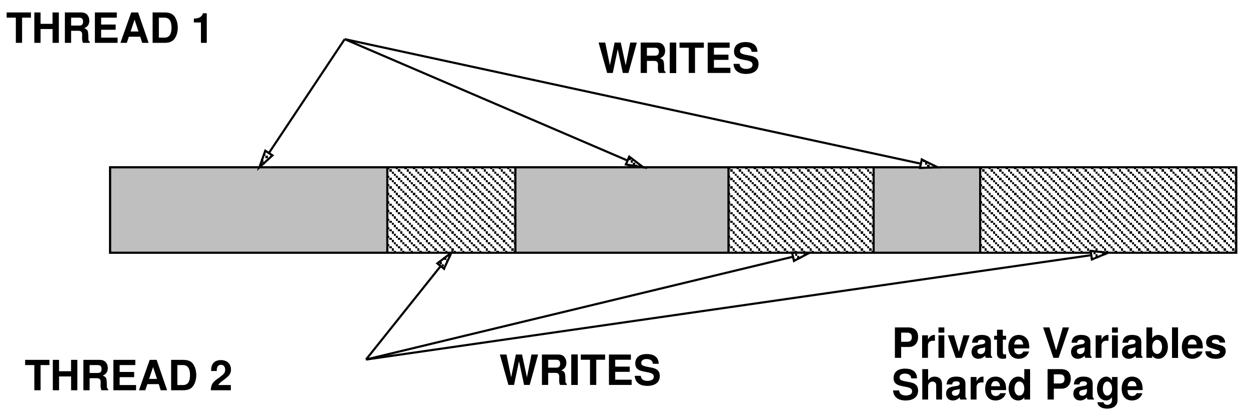


Sequential Consistency



Problem: Unnecessary communication

False Sharing



Leads to "ping-pong" effect

Moderately Lazy Release Consistency

Release Consistency (RC) [Ghara. et al. 90]

- Distinguish ordinary accesses and synchronization (acquire and release)

Need to guarantee consistency only after synchronization

$\leadsto$ Enforce consistency only at synchronization points

Push modifications at release

Apply modifications at acquire

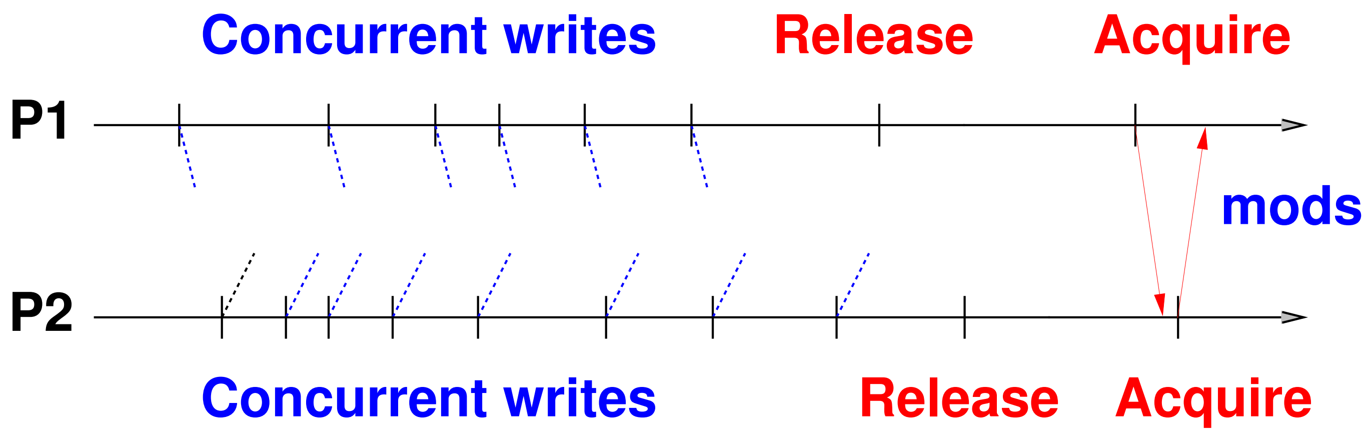
Multiple Writer Protocol

Reduces false sharing overhead

Buffer writes until release

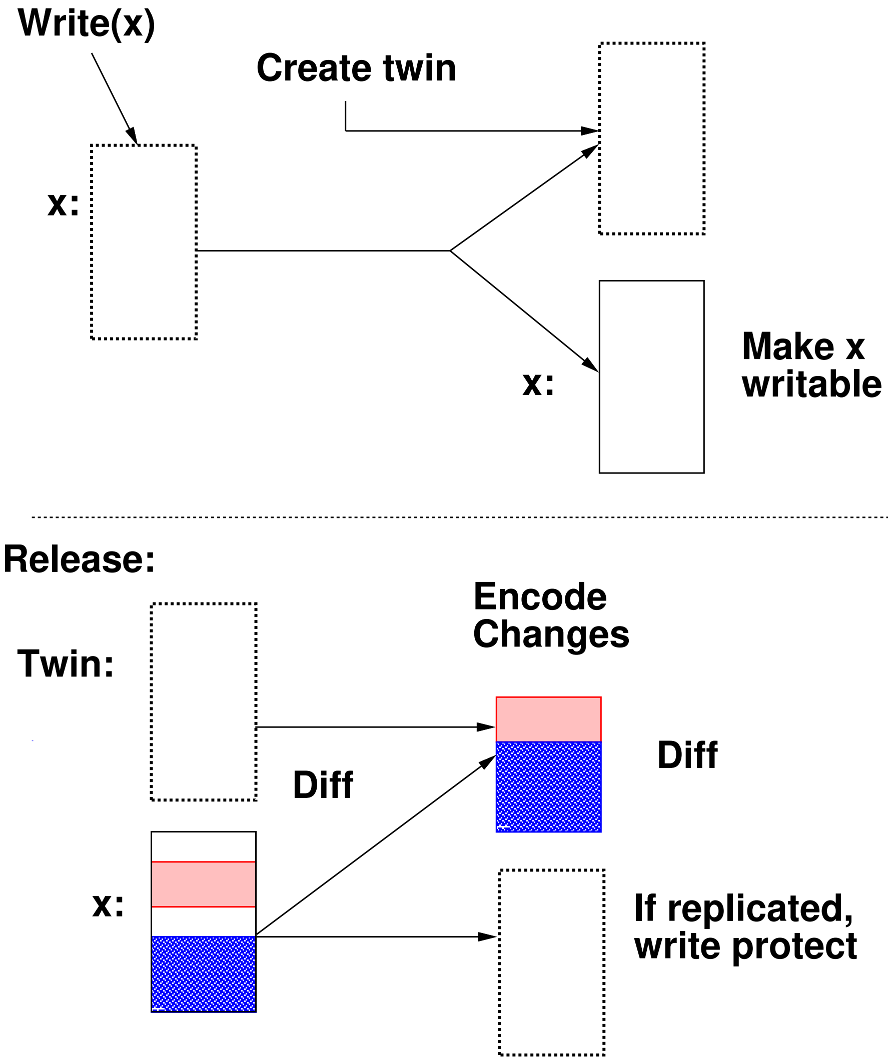
Create diffs and propagate to home node at release

Acquire lock $\rightarrow$ apply modifications



Outgoing Diffs

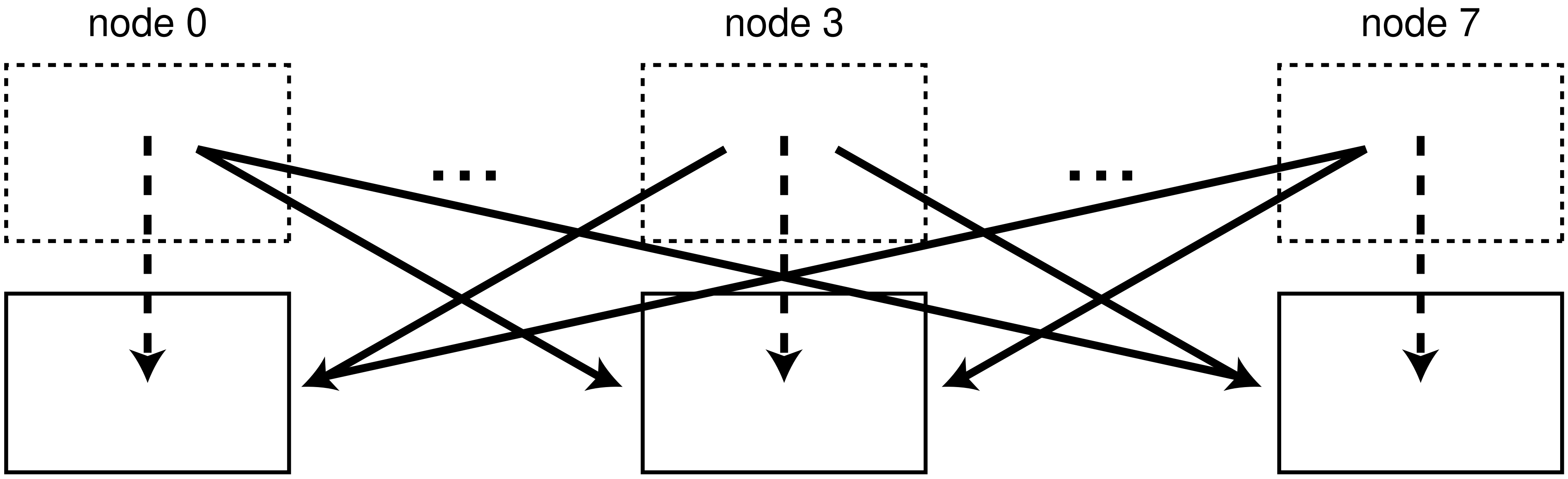
[Munin, TreadMarks, Home-based LRC]



Cashmere-2L Address Space Layout

Directory-Based Coherence

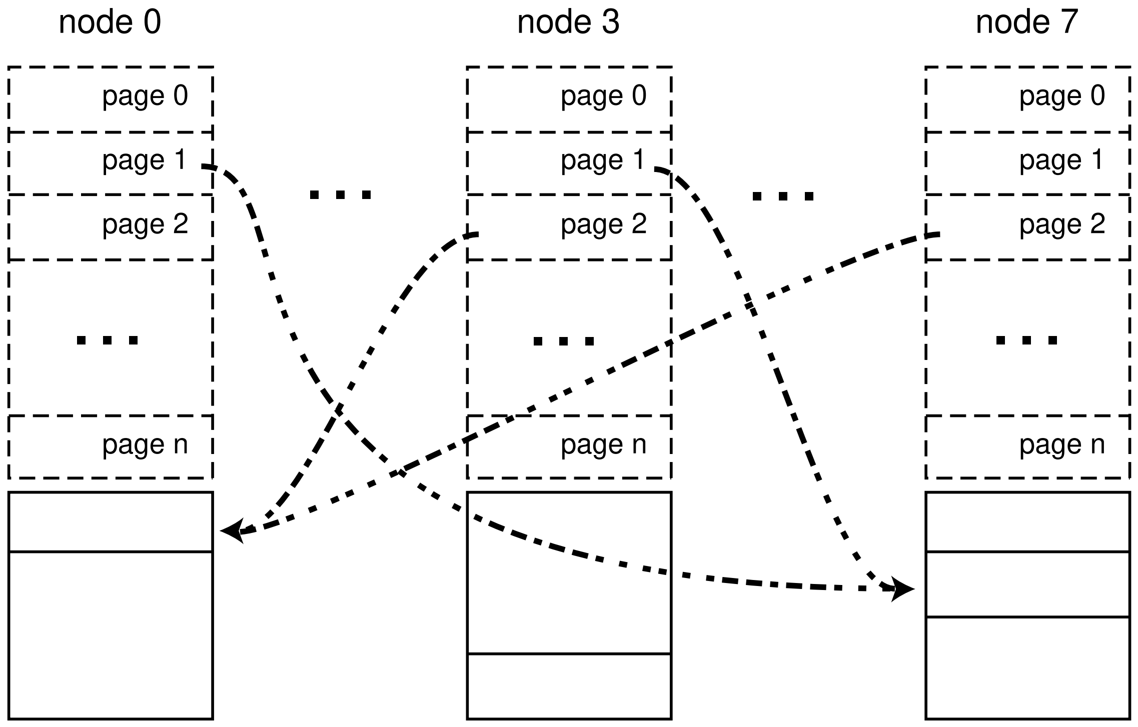
Global directory tracks per-SMP accesses to pages



Synchronization objects also employ broadcast

Home-Node Based Coherence

Data kept up-to-date at the home node



Exclusive and Migratory mode to avoid unnecessary communication

Protocol Actions

Page Faults

- Update global page state information
- Obtain current data on a read fault, create twin on a write fault

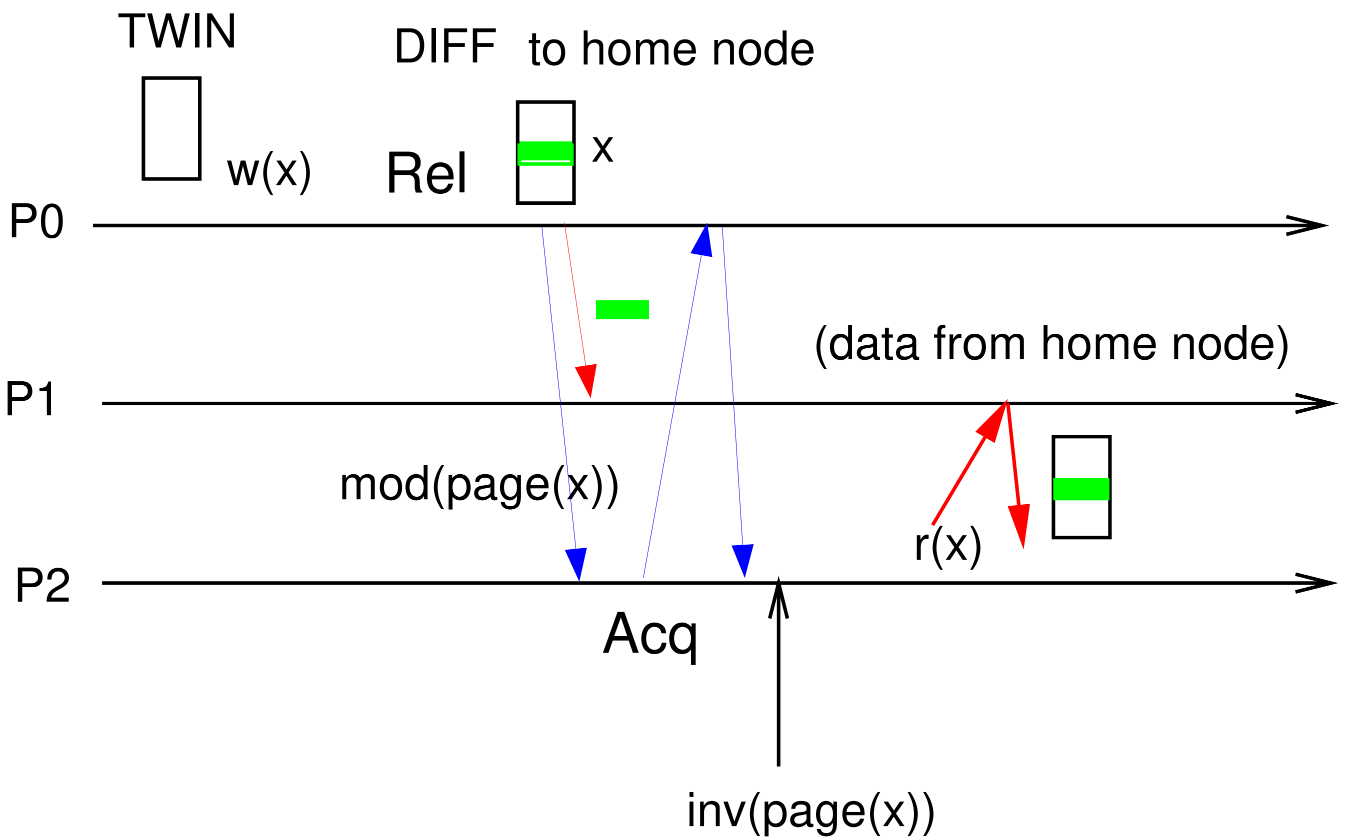
Releases

- Use outgoing diffs to send modifications to the home node
- Send write notices to sharing processors

Acquires

- Invalidate pages named by write notices
- Update global page state information

Protocol Actions for Cashmere-2L



Exploitation of Network Capabilities

Shared data -

- Use remote write to propagate modifications to the home node - avoids interrupts

Protocol meta-data -

- Broadcast directory modifications using remote-writes
- Rely on total network order for lock-free data structures
- Deliver write notices using remote writes without interrupts
- Use remote-writes for polling-based messaging

Application synchronization -

- Use remote writes for fast application locks and barriers

SMP Design Optimizations

Each processor in a node shares same page frame

Coalesce global meta-data operations

Avoid redundant data fetches and home-node updates

Design protocol for asynchronous operation of processes

Hardware/Software Interaction

Software coherence operations performed for entire node

- Coalesce fetches of data to validate pages
- Coalesce updates of modified data to home node

Per-page timestamps within SMP to track remote-SMP events

- Last write notice received
- Last home-node update
- Last data fetch

Avoiding Redundant Updates

Reducing Synchronization Requirements within a Node

Need to provide mutually exclusive update capabilities with multiple writers on a node

Possible solutions

- Shootdown TLB entries of all concurrent copies
- Cashmere-2L - Incoming diffs

Incoming Diffs

Current Platform

32 600 MHz 21164 processors (8 AlphaServer 4100 4/600 SMPs)

- 8KB icache, 8KB dcache, 96KB combined, on-board
- 8MByte board-level cache
- 2GByte primary memory per SMP
- 31GB disk per SMP
- 18.8 SpecInt95 rating, 29.2 SpecFP95 rating

DEC Memory Channel II network

- One-way latency: 3.35μsecs
- Bandwidth - 76 MBytes/sec per link

Target Parallel Applications

Computational Biology/Physics

- Genetic linkage analysis [HH94, CBR95]
- DNA chip data analysis
- Computational fluid dynamics

Vision Applications

- The Correspondence problem [BrazilianCAHPP97]
- Object recognition
- Volumetric 3-D reconstruction from 2-D images
- Cone-beam tomography
- Neurology simulations

Experimental Platform

32 233 MHz 21064A processors (8 AlphaServer 2100 4/233 SMPs)

- 16KByte icache, 16KByte dcache on-chip caches
- 1MByte board-level cache
- 256MBytes primary memory per SMP

DEC Memory Channel I network

- One-way latency: 5.2μsecs
- Bandwidth - 29 MBytes/sec per link, 60 MBytes/sec aggregate

Experiments

Comparison (of Cashmere-1L) with TreadMarks (state-of-the-art system developed for high-latency networks)

Performance of Cashmere-2L relative to Cashmere-1L

Effects of network capabilities on performance

Effects of SMP optimizations

Comparison to non-VM-based system

TreadMarks

State-of-the-art Software Distributed Shared Memory System

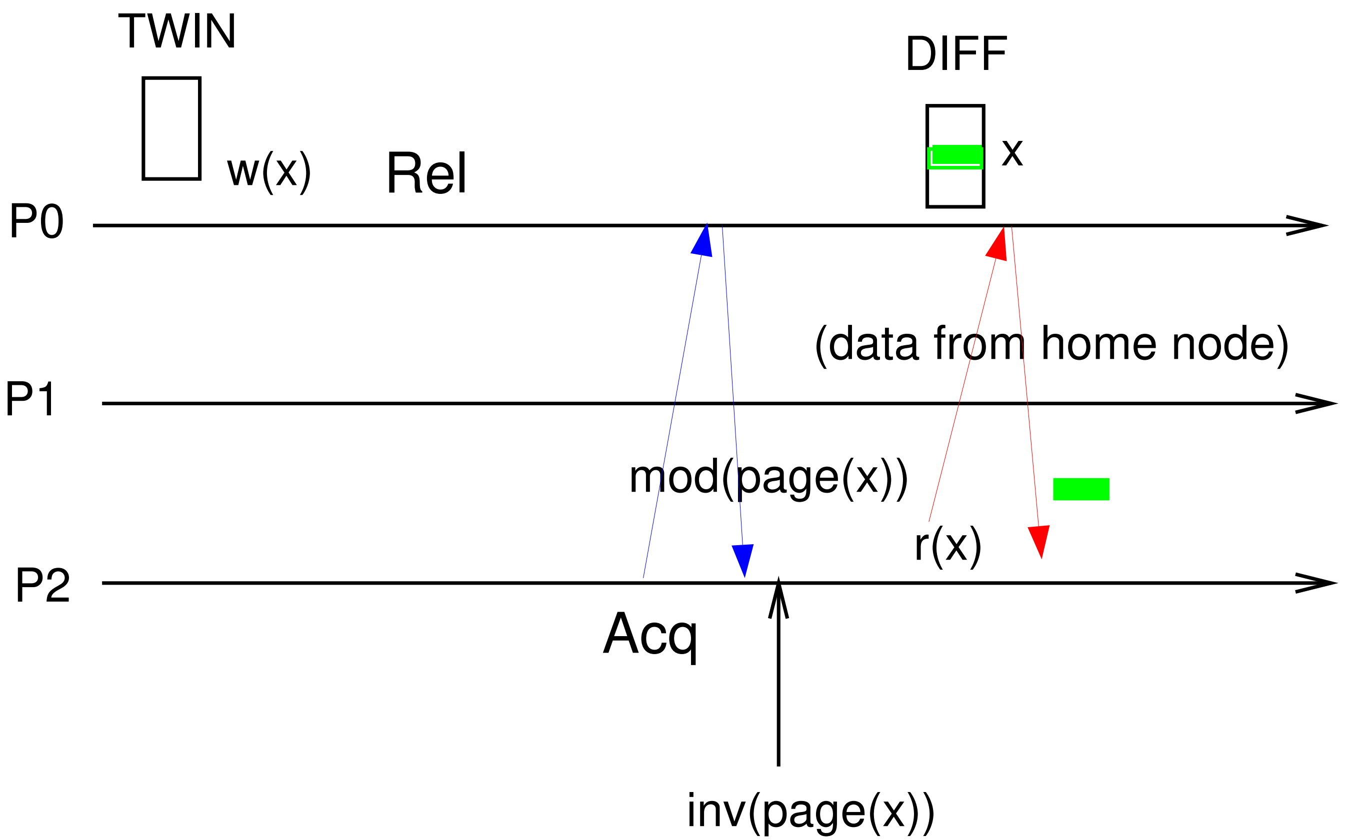
Page-based

Distributed vector timestamps

Lazy release consistency (LRC - [Keleher et al. 92])

Multiple writer protocol [Carter et al. 91]

Protocol Actions for TreadMarks



⇝ Must communicate all write notices to all processors

MicroBenchmarks

Operation	Time (μs)		
	CSM-2L	CSM-1L	TMK
Lock Acquire	19	11	79
Barrier	58 (321)	41 (364)	75 (1213)
Page Transfer (Local)	-	230	230
Page Transfer (Remote)	520	500	520

Incoming diffs - $\sim 500\mu secs$

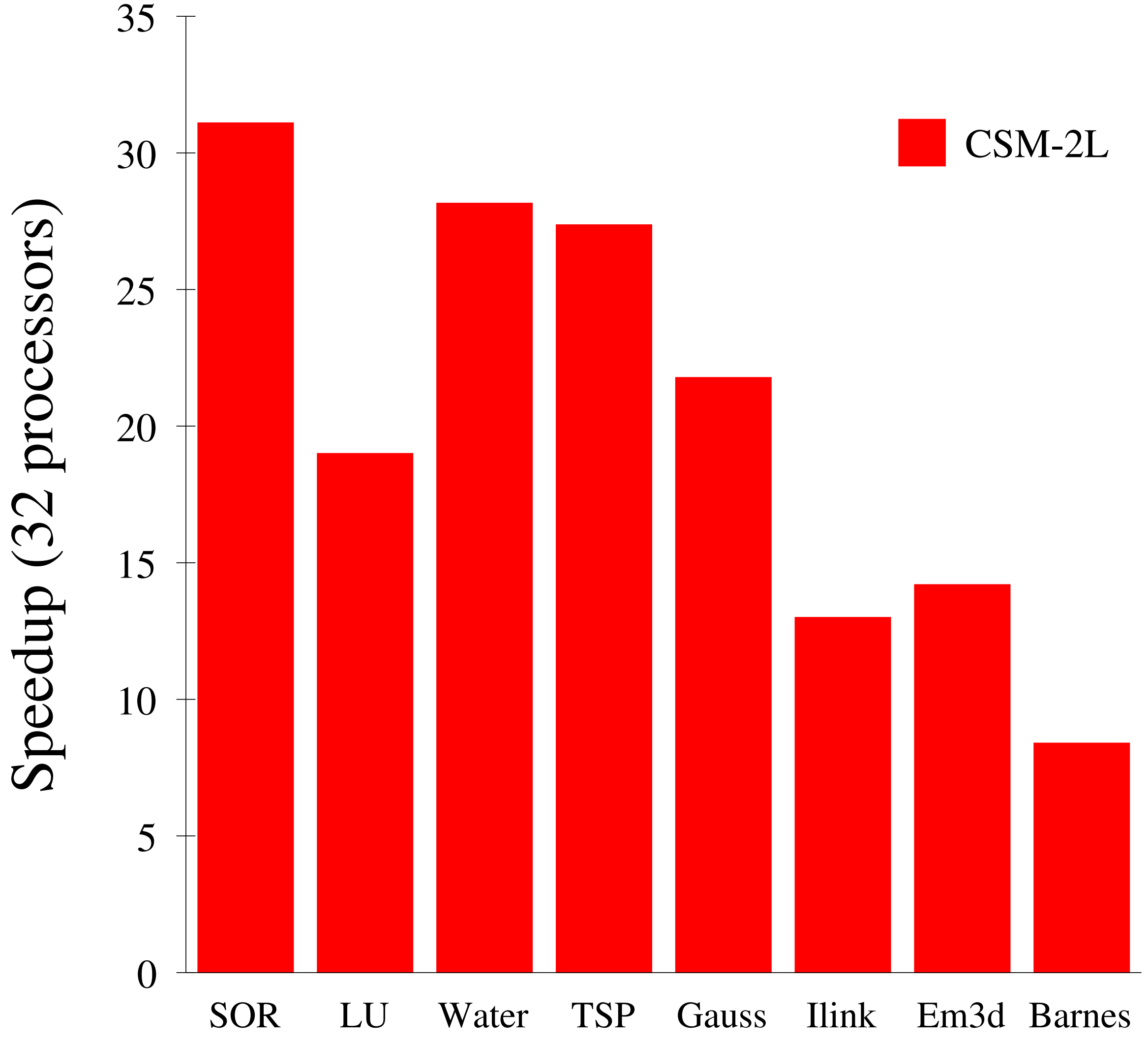
Outgoing diffs - $\sim 350\mu secs$

Twin creation - $\sim 200\mu secs$

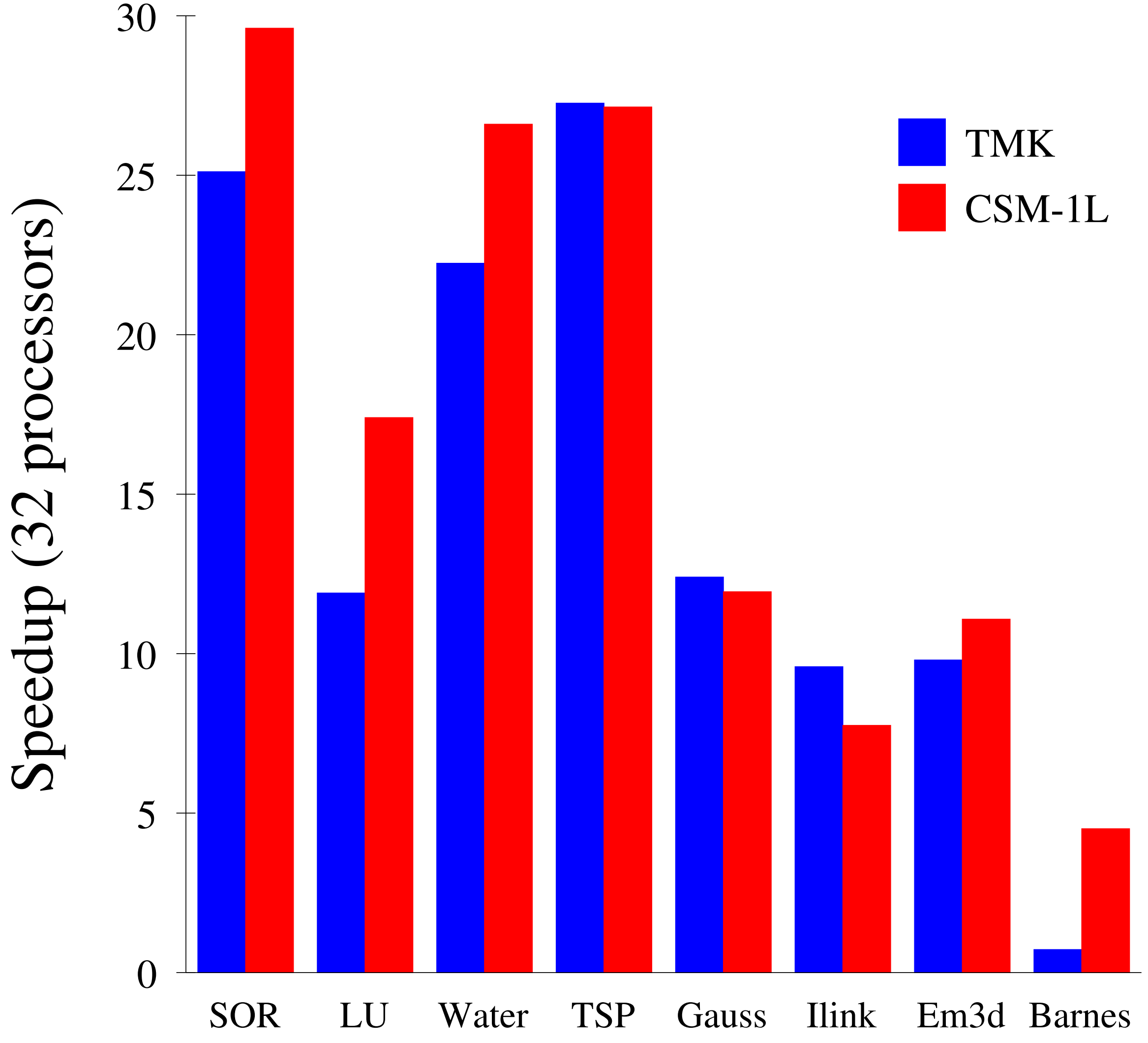
Benchmarks

Program	Problem Size	Time (sec.)
SOR	3072x4096 (50Mbytes)	195.0
LU	2046x2046 (33Mbytes)	254.8
Water	4096 mols. (4Mbytes)	1847.6
TSP	17 cities (1Mbyte)	4029.0
Gauss	2046x2046 (33Mbytes)	953.7
Ilink	CLP (15Mbytes)	899.0
Em3d	60106 nodes (49Mbytes)	161.4
Barnes	128K bodies (26Mbytes)	469.4

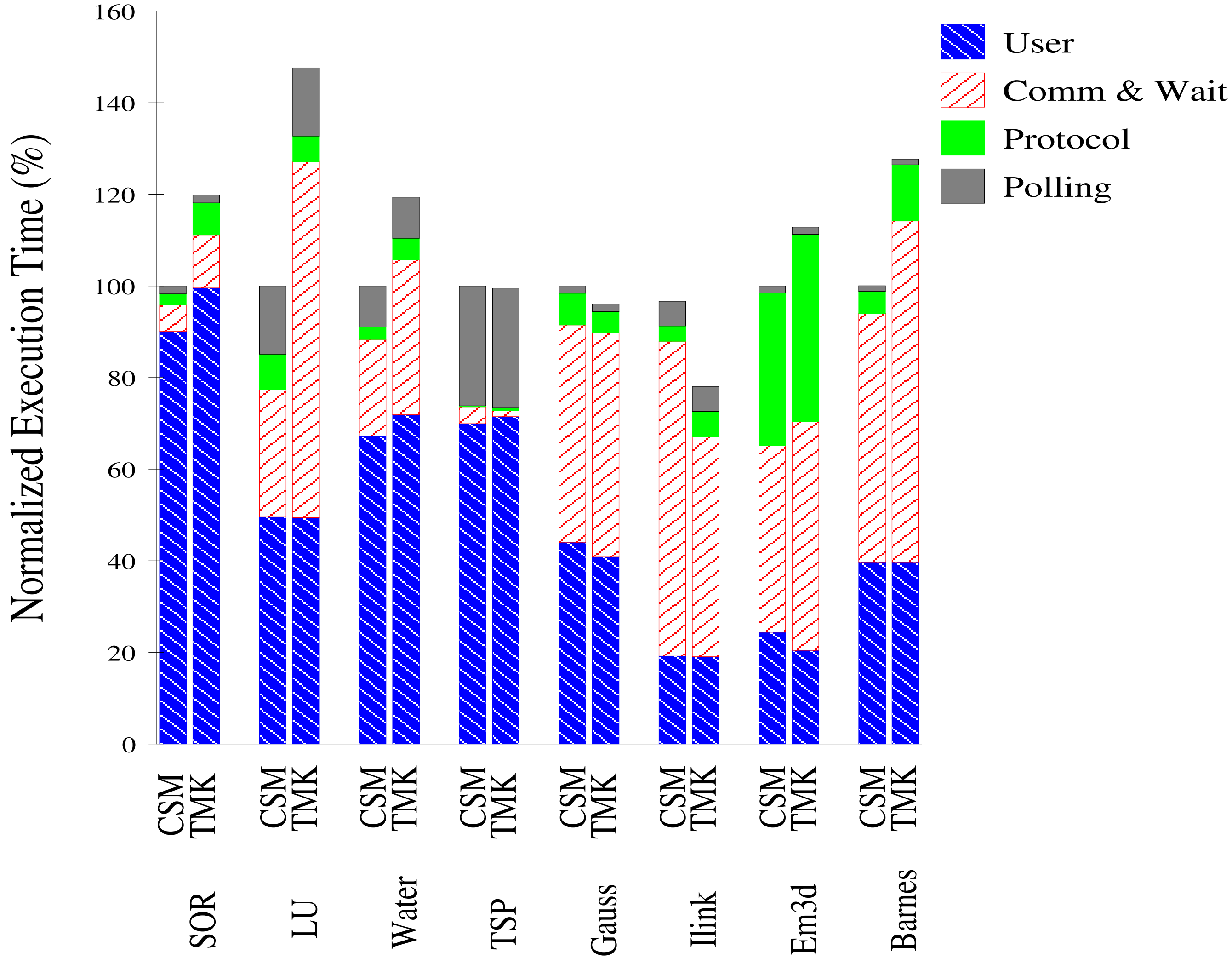
Performance: Cashmere-2L



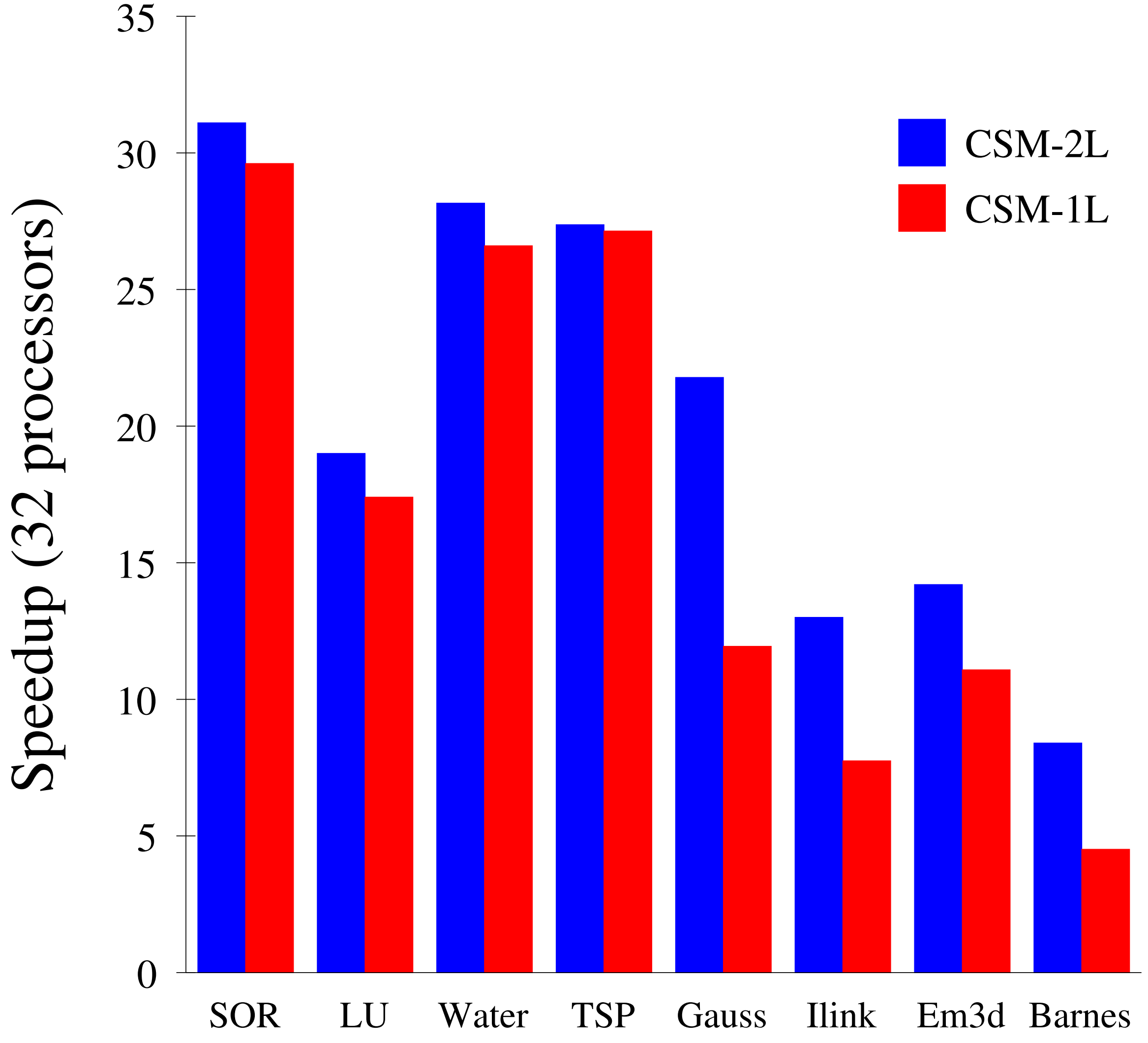
Performance: TreadMarks Vs. Cashmere-1L



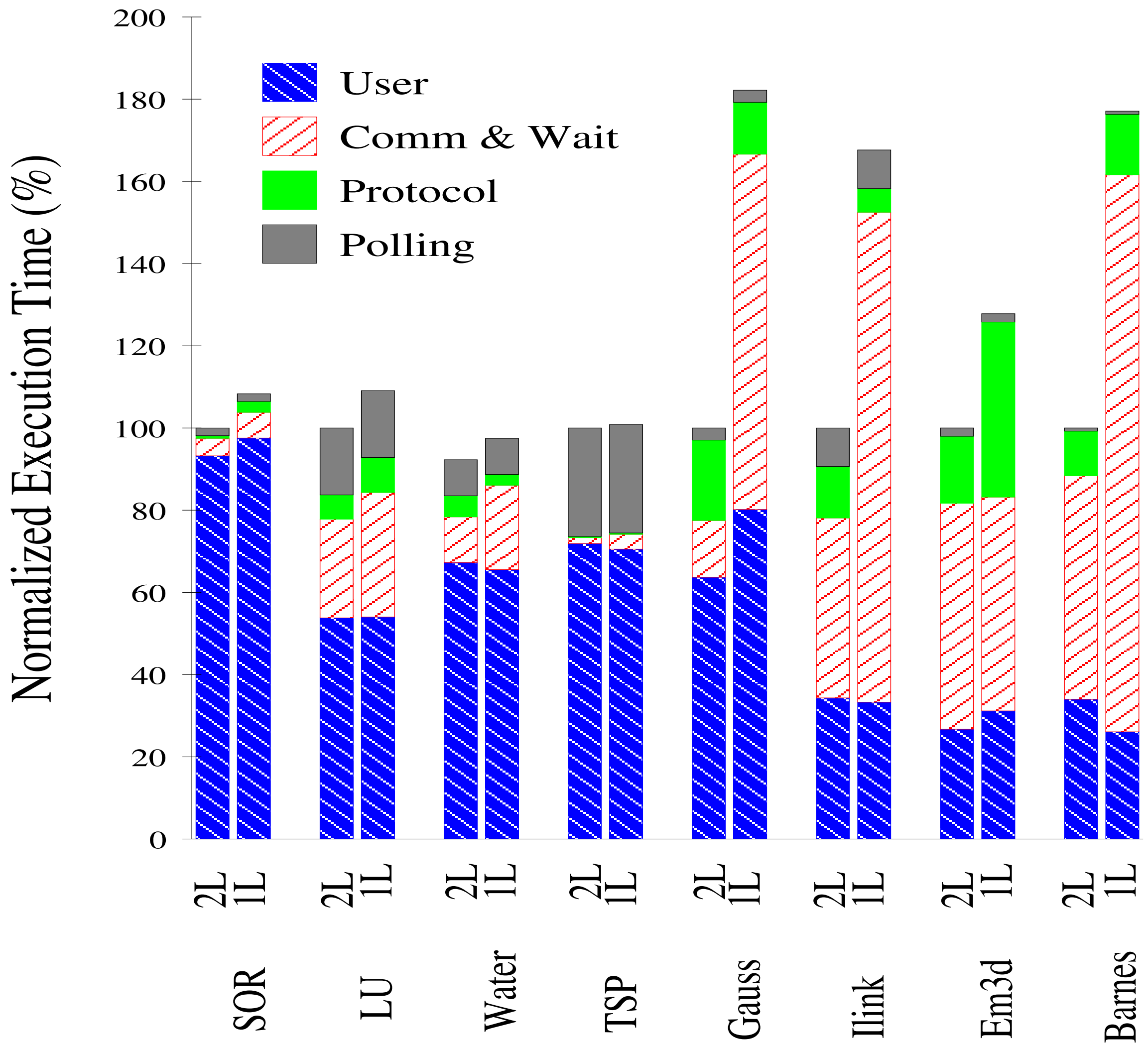
Execution Breakdown: TreadMarks Vs. Cashmere-1L



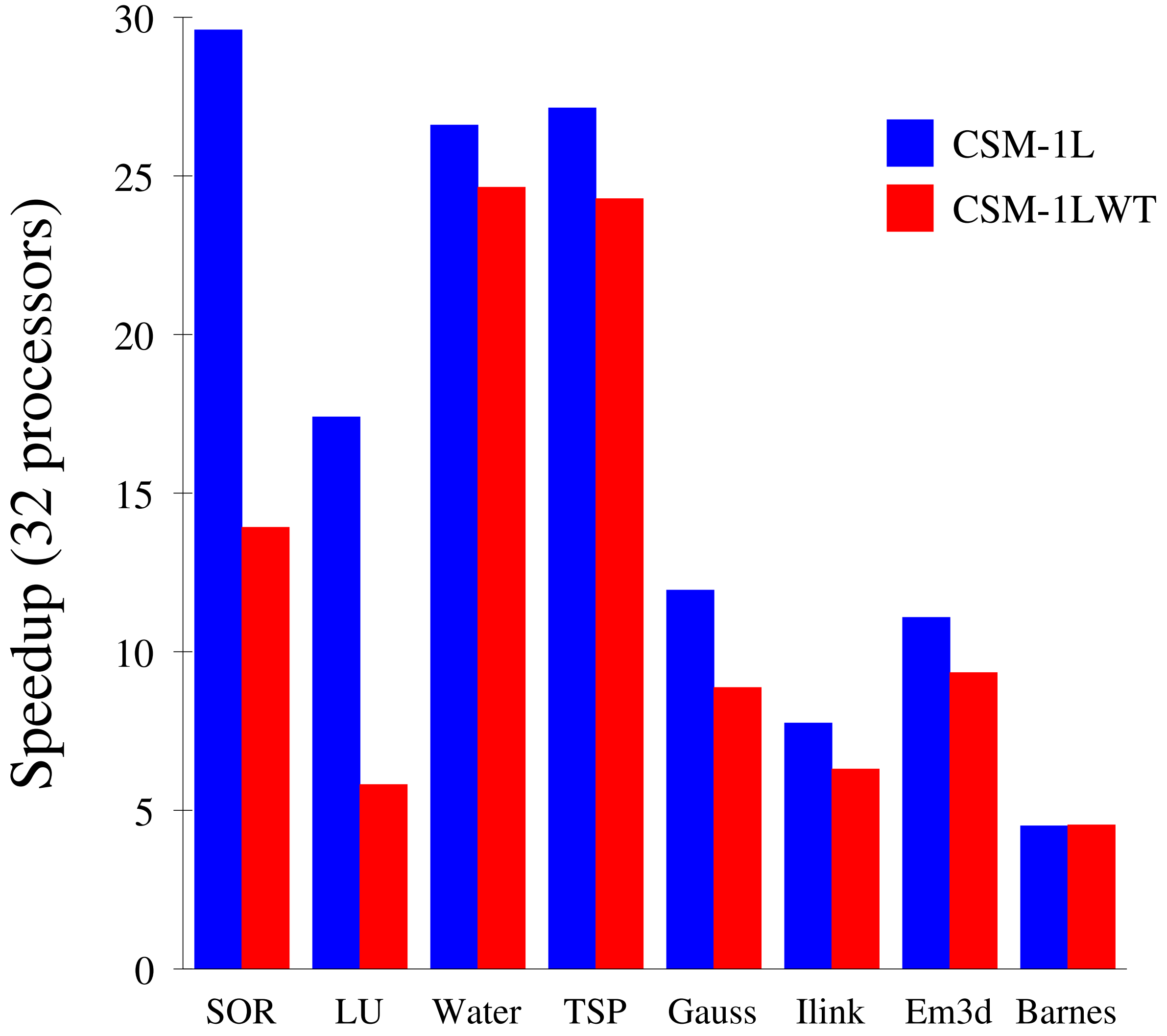
Performance: Cashmere-2L Vs. Cashmere-1L



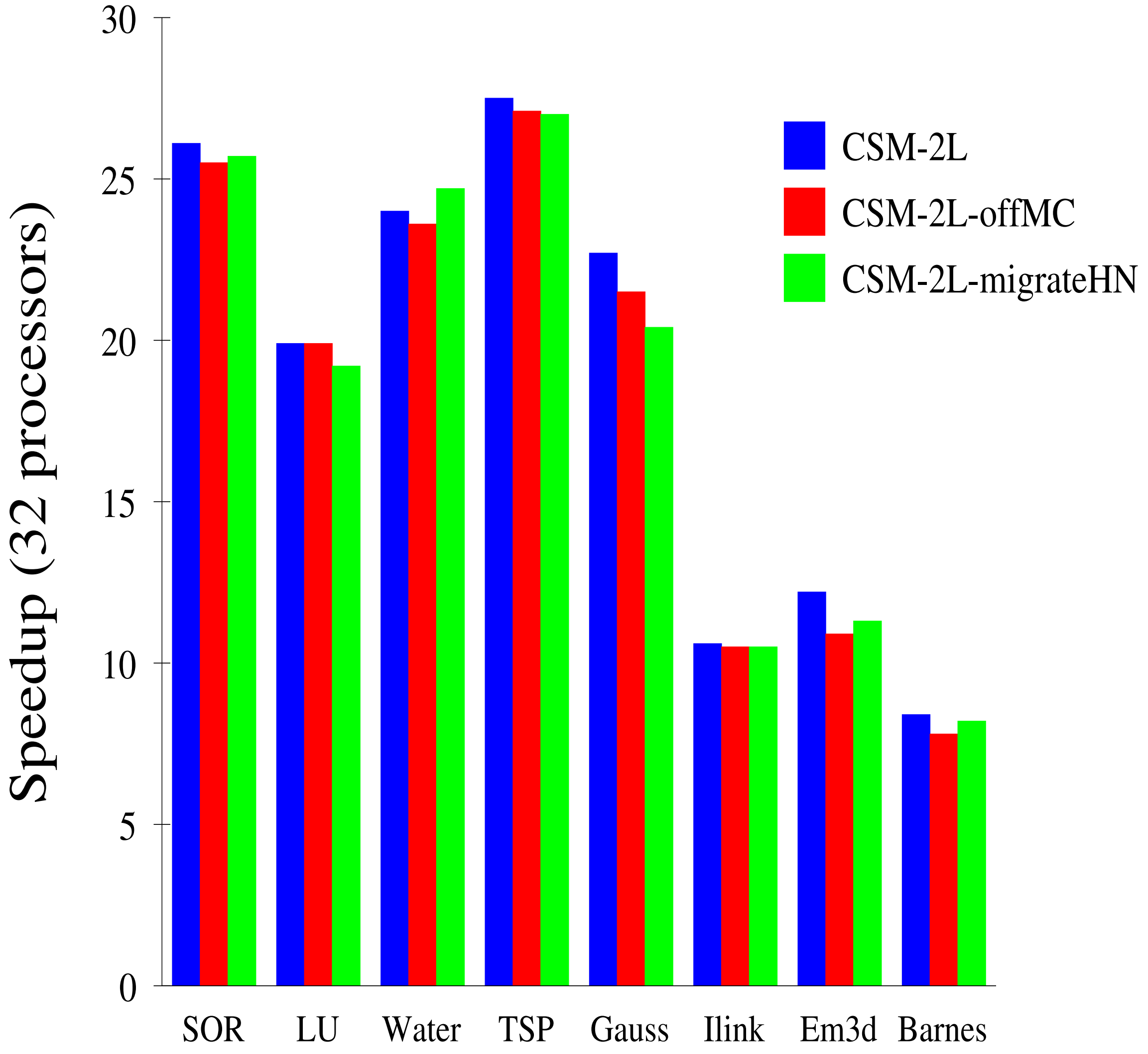
Execution Breakdown: 2L Vs. 1L



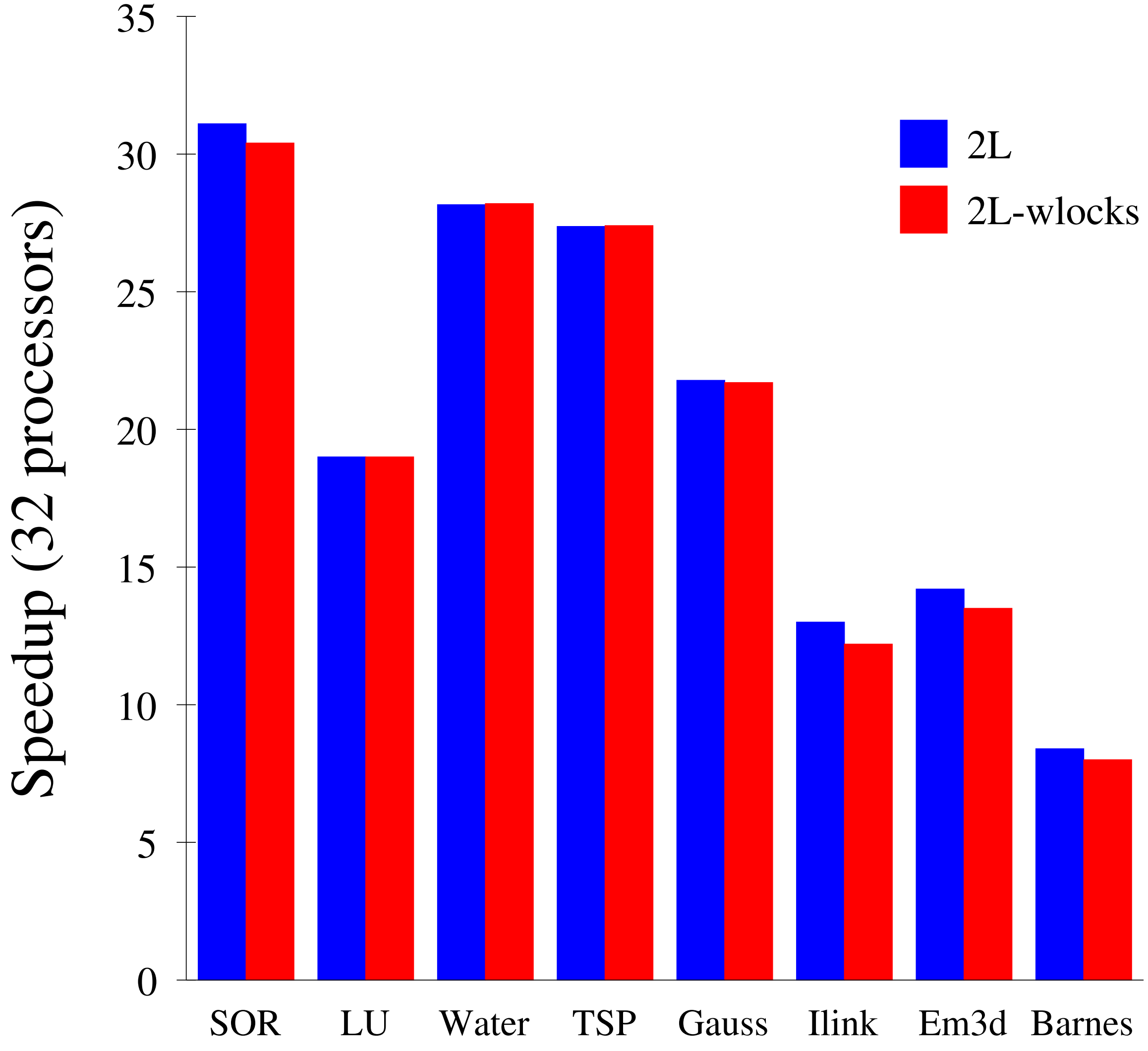
Performance: Cashmere-1L Using Write Through



Impact of Placing Data on the Memory Channel



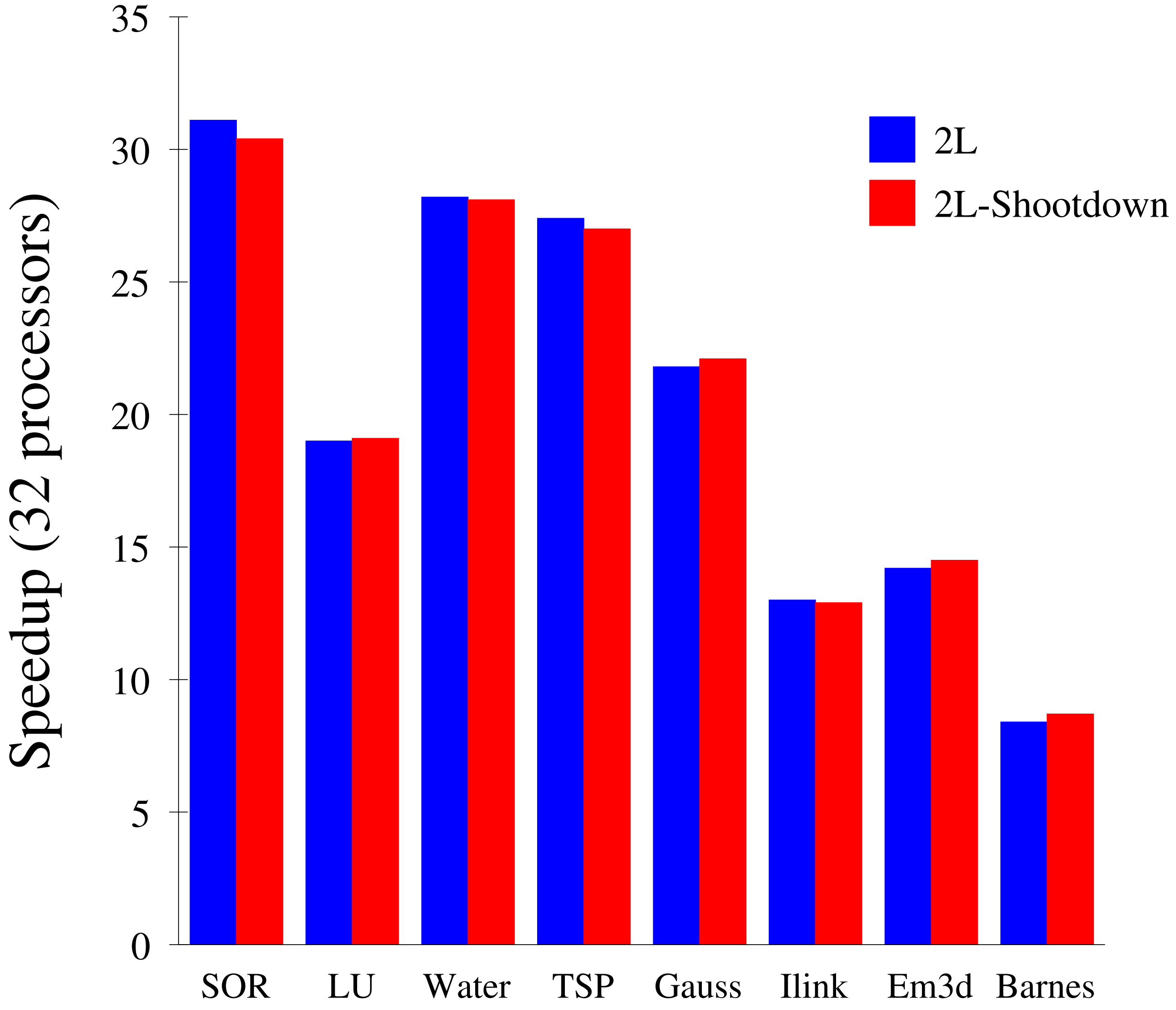
Performance: Lock-free Directories



Lock-free Directories

- Improvement seems negligible
  - Ilink - 7%, Barnes - 5%
- Not a significant amount of contention for directory entries

Performance: Incoming Diffs



Incoming Diffs Vs. TLB Shootdown

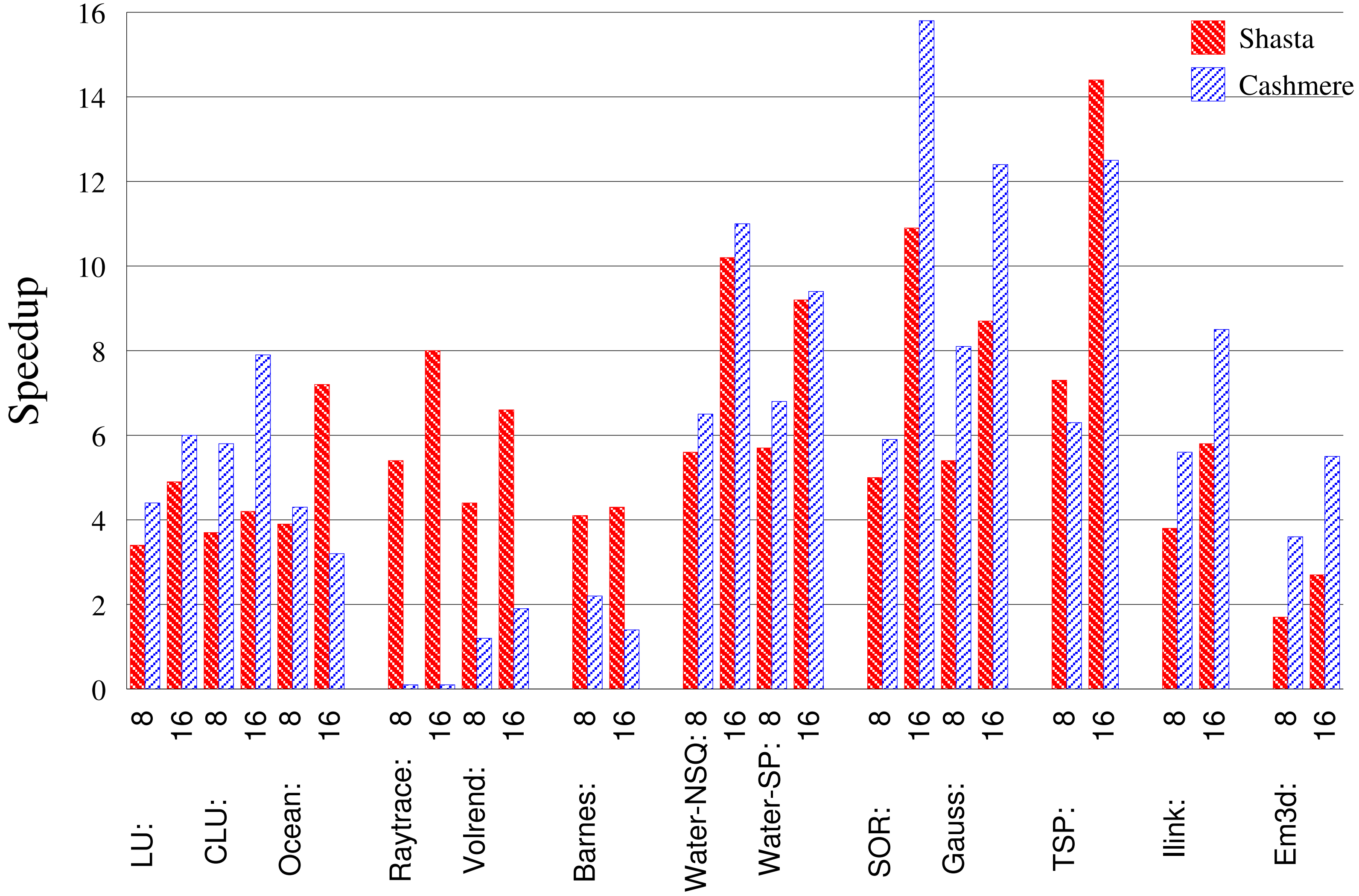
No significant performance difference!

- Seems to contradict SoftFLASH [ASPLOS96] results

Reasons due to important protocol design decisions

- multiple-writer protocol (vs. SoftFLASH's single writer protocol)
- separate page tables (vs. SoftFLASH's shared page tables)

VM-Based Vs. Variable (Fine) Grain Coherence



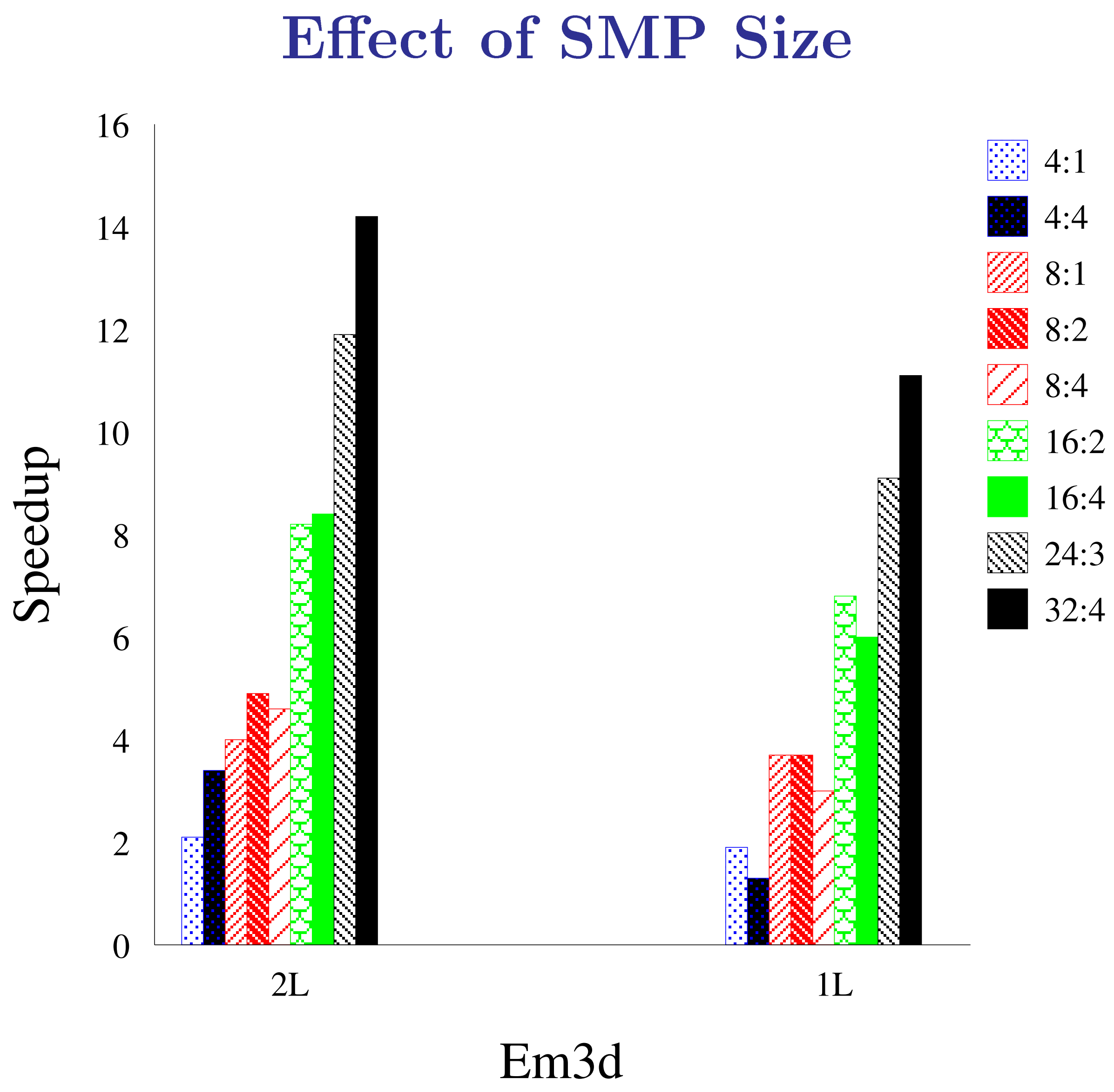
VM-Based Vs. Variable (Fine) Grain SDSM

VM-Based (coarse grain)

- VM-based - avoids instrumentation overhead
- Higher performane with coarse-grain synchronization
- Must pay attention to computation granularity when using work queues, reductions, atomic counters

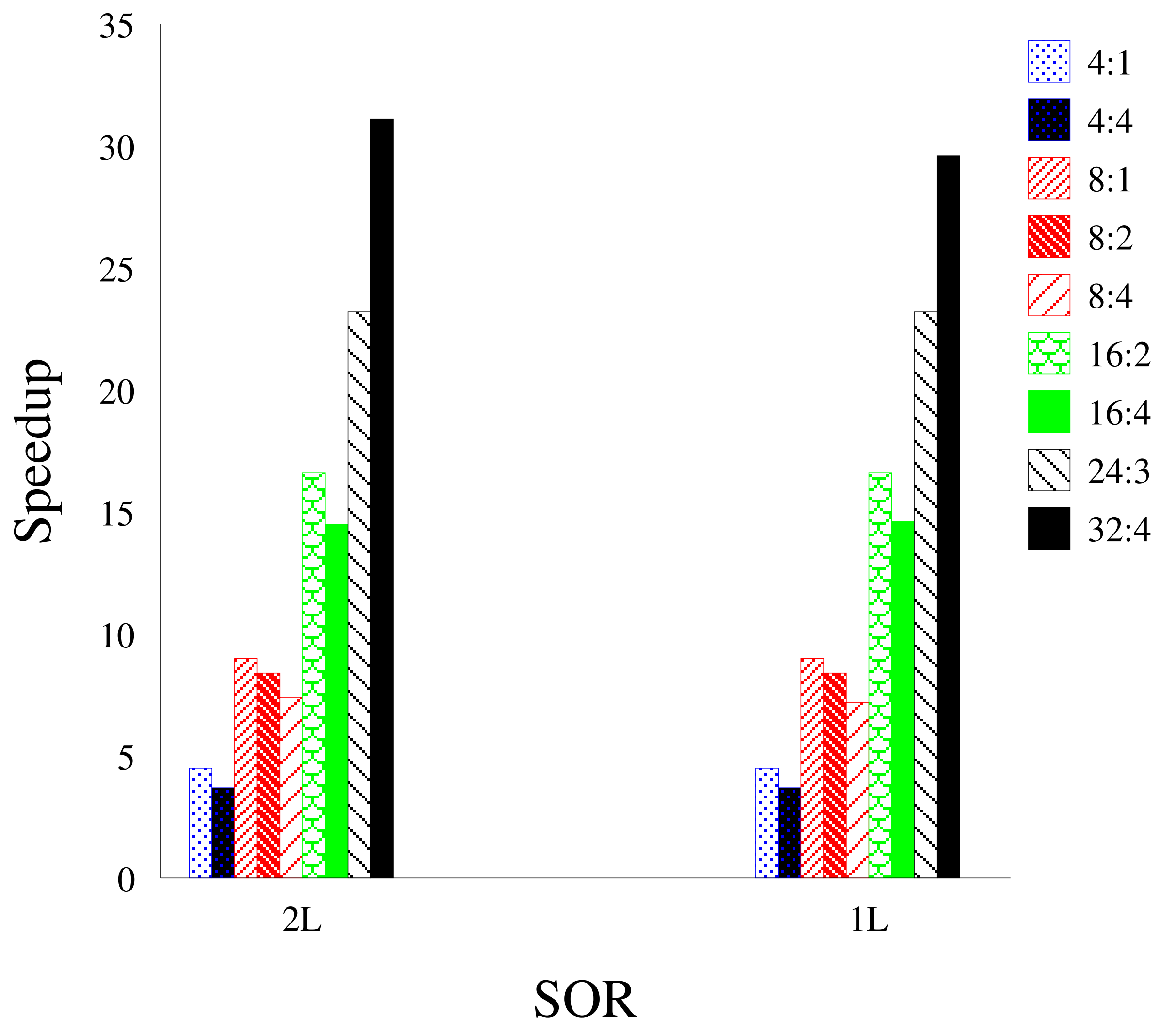
Variable (fine) grain

- Instrumentation-based - allows variable (fine) granularity
- Incurs fixed cost even in the absence of sharing
- Higher performance and robustness in the presence of fine-grain synchronization
- Must pay attention to coherence granularity for coarse-grain synchronization

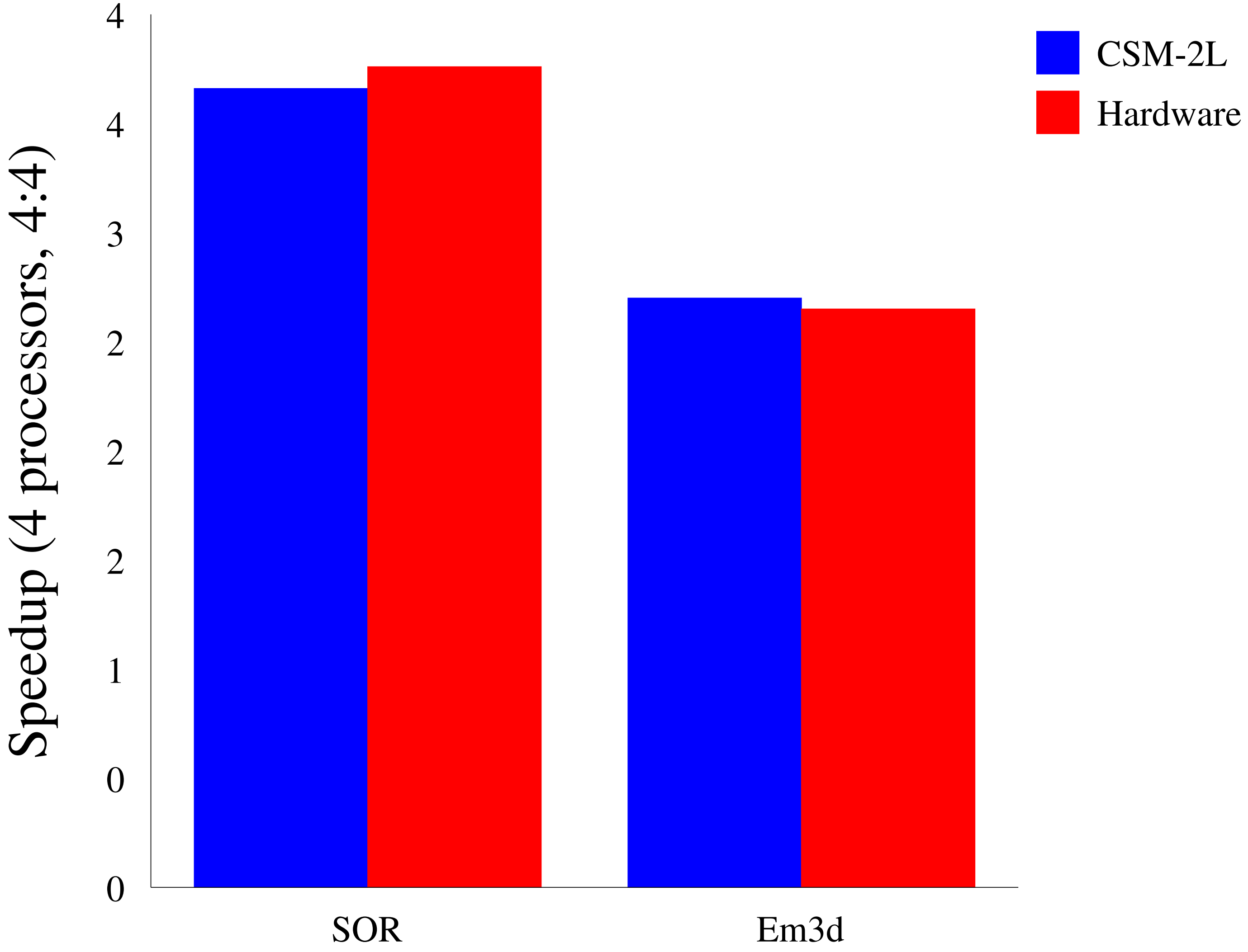


Effect of SMP Size
Speedup
16
14
12
10
8
6
4
2
0
2L
1L
Em3d
4:1
4:4
8:1
8:2
8:4
16:2
16:4
24:3
32:4

Effect of SMP Size



Cashmere-2L Vs. Hardware



Conclusions

Low-latency networks make directories a viable alternative for SDSM

Remote-write capability mainly useful for fast messaging and polling

Two-level design exploits hardware coherence within SMP

- Sharing within SMP uses hardware coherence
- Operations due to sharing across SMPs coalesced

Multiple-writer protocols reduce the need for synchronization within SMP

Ongoing Work

ARCH: Compiler-Runtime integration for

- load balancing
- optimal communication

with the goal of making SDSM a uniform platform-independent programming model

InterAct: Sharing support for interactive client-server applications

Applications