Shared Memory Models

Michael L. Scott

University of Rochester

Programming Models and Requirements Session

Workshop on Communication and Middleware
for Parallel Programming Models

Held in conjunction with IPDPS

April 2002

Thanks to
Sandhya Dwarkadas, Leonidas Kontothanassis, Rob Stets,
Umit Rencuzogullari, Nicolaos Hardavellas, Galen Hunt,
Luke Chen, and Chunqiang Tang

S-DSM Motivation

- Shared memory an attractive programming model
  - » Familiar to users of SMPs and multi-threaded languages
  - » Arguably simpler than message passing, esp. for non-performance-critical code
  - » Has the potential to out-perform inspector-executor code for irregular access patterns, esp. with compiler support
- Outline
  - » Brief history
  - » Current state of the art
  - » Obstacles to better performance
  - » Wish list for communication architectures
  - » Plug for InterWeave

A Brief History of the Field

- 1986 Ivy
- 1989 Shiva
- 1990 Munin
- 1992 LRC (TreadMarks)
- 1993 Sh. Regions, Midway
- 1994 AURC (Shrimp)
- 1995 CRL
- 1996 Shasta
- 1997 Cashmere
- 1998 HLRC
- 2000 InterWeave

The original idea (Kai Li)

Relaxed memory model, optimized protocols

Software-only protocols

★ Leverage special HW (User-level messages, multiprocessor nodes)

S-DSM for SANs

- Relaxed memory model: multi-writer release consistency, data-race-free applications
- “Moderately lazy” protocol with home nodes: propagate updates and write notices at release time
- Page-size coherence granularity (false sharing not a major issue)
- VM-based write detection
- Multiprocessor nodes
- User-level network interface
- Exemplars: Cashmere, HLRC/GeNIMA

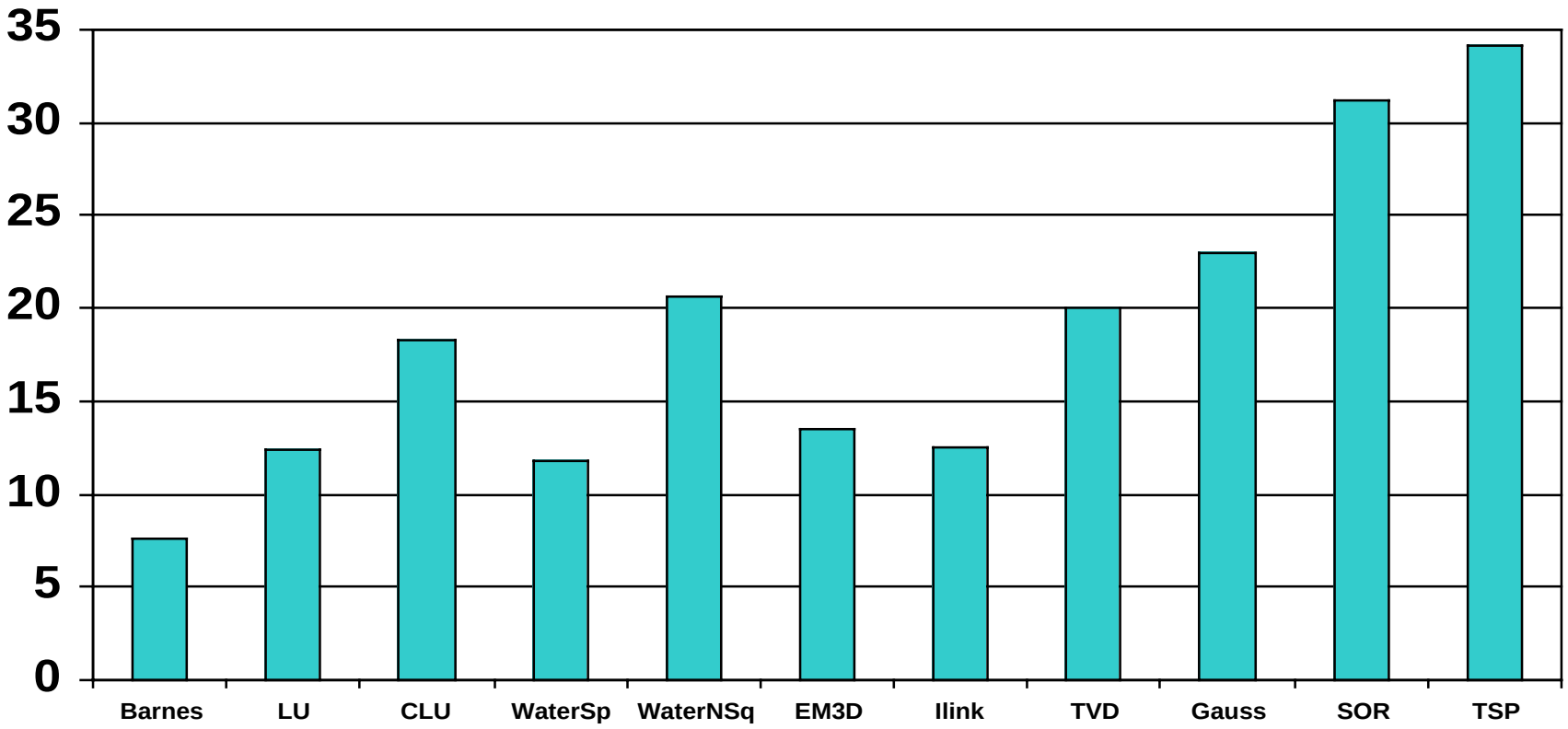
The Bad News

- The key ideas seem to have been discovered; the rate of innovation is way down
- Program committees aren't looking for papers
- No massively parallel apps
- No major OS vendor packages S-DSM (TreadMarks the only commercially-available system; optimized for high latency messages)
- High-end users (who might tolerate research code) still stick to MPI

The Good News

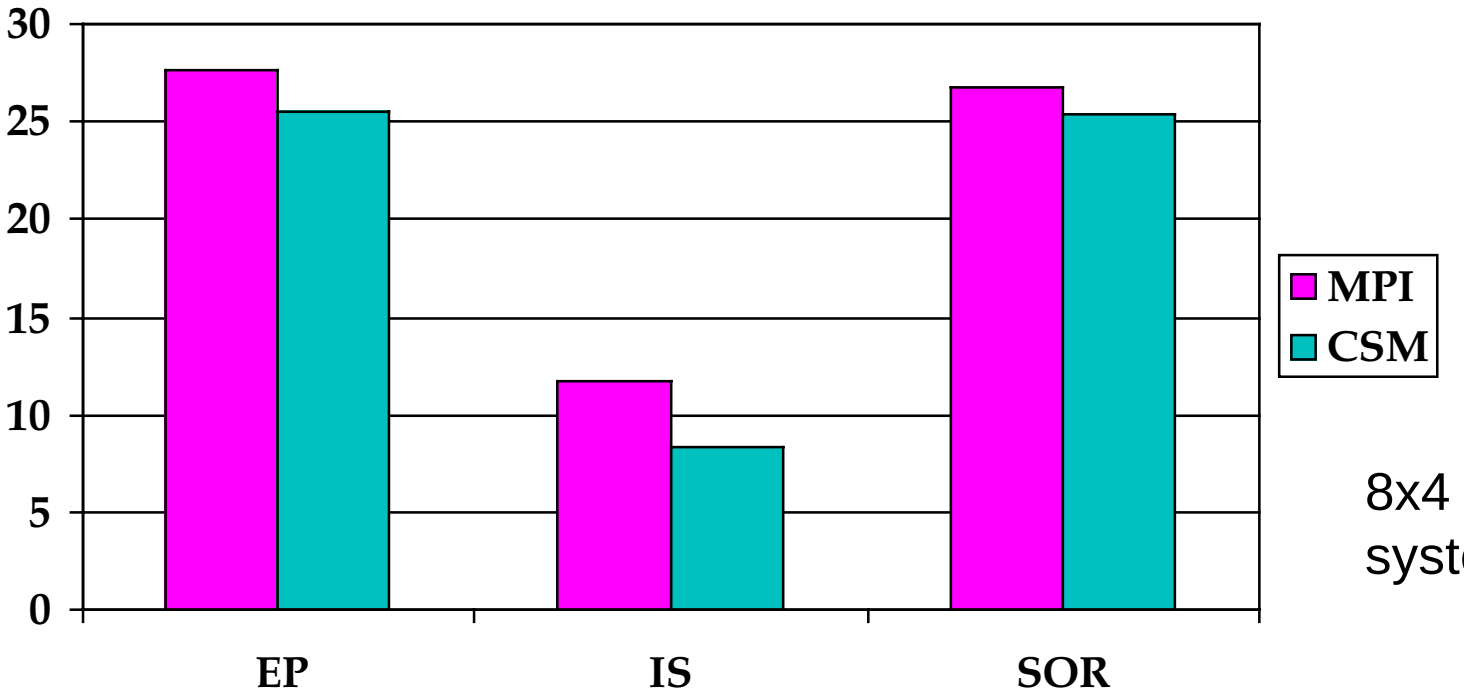
- Speedups for well-written apps are quite good
- Widespread use awaits production-quality systems
- Performance may improve with compiler support and better communication architectures
- The ideas are valuable in a wider domain; hence InterWeave (see plug at end)

Cashmere Speedups



8x4 Alpha system

Cashmere and MPI



8x4 Alpha system

- Lu et al report similar results for TreadMarks & PVM

Obstacles to Better Speedup

① False sharing (not as serious as once thought)
② Write detection overhead (twins, diffs, faults)
③ Maximum one-page message size
④ All “pull”-based — no “push” (opposite of most HW)

- 2, 3, and 4 amenable to compiler support
- 2, 4, and maybe 3 to protocol improvements
- 2 and 4 to hardware support

Communication Features

- ★ Low latency
- ★ Fast interrupts
- □ Remote writes
- □ Reliable delivery, total order
- ➡ Broadcast
- ➡ Remote reads, RMW (e.g. CAS)
- ➡ Very large address space, shadow page tables
- ▭ Fine-grain protection [UWisc], HW write detection [Bianchini], true HW coherence

★ crucial
□ useful
▭ probably impractical
➡ stay tuned

Cashmere Protocol Variants

- "Memory Channel" Cashmere
  - » shared data propagation (diffs)*
  - » meta-data propagation (directory, write notices)
  - » synchronization mechanisms (locks, barriers)

  ** Remotely-accessible shared data space is limited*

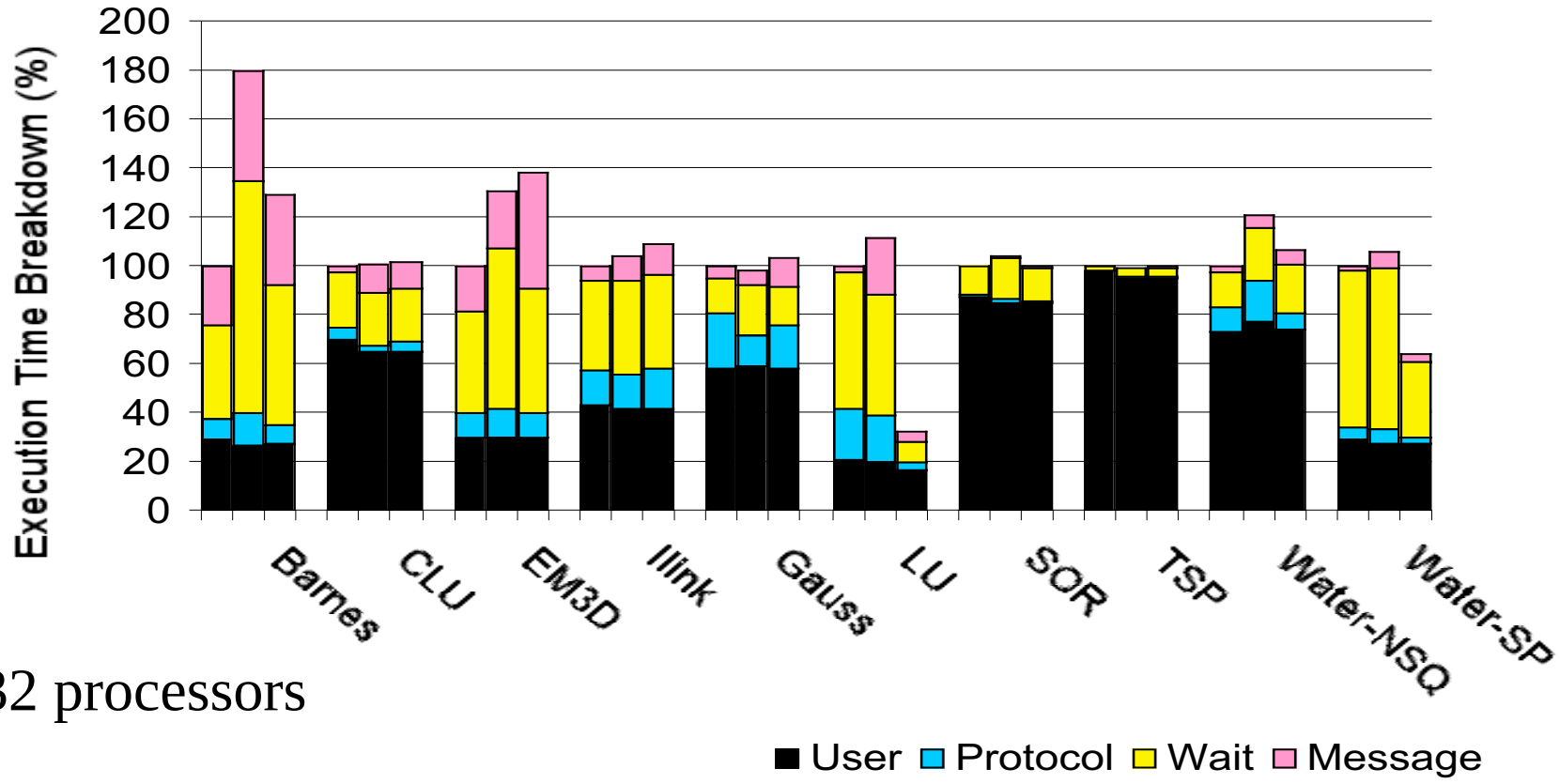
- "Explicit messages" Cashmere
  - » diffs, write notices, locks, barriers: use plain messages
  - » directory: maintain master entry only at the home
  - » hide ack latency by pipelining diffs, write notices

Free Home Node Migration

- Reduce twin/diff overhead by *migrating* home node to the active writer
  - » migration is not possible when using remote write to propagate data
- Send migration request as part of each write fault
  - » home will grant migration request if it is not actively writing the page
  - » old node will forward any incoming requests to new home
- *Migration very effectively reduces twin/diff operations*

Relative Performance

Left, middle, right bars:
MC Features, Explicit Msgs, Explicit Msgs Migration



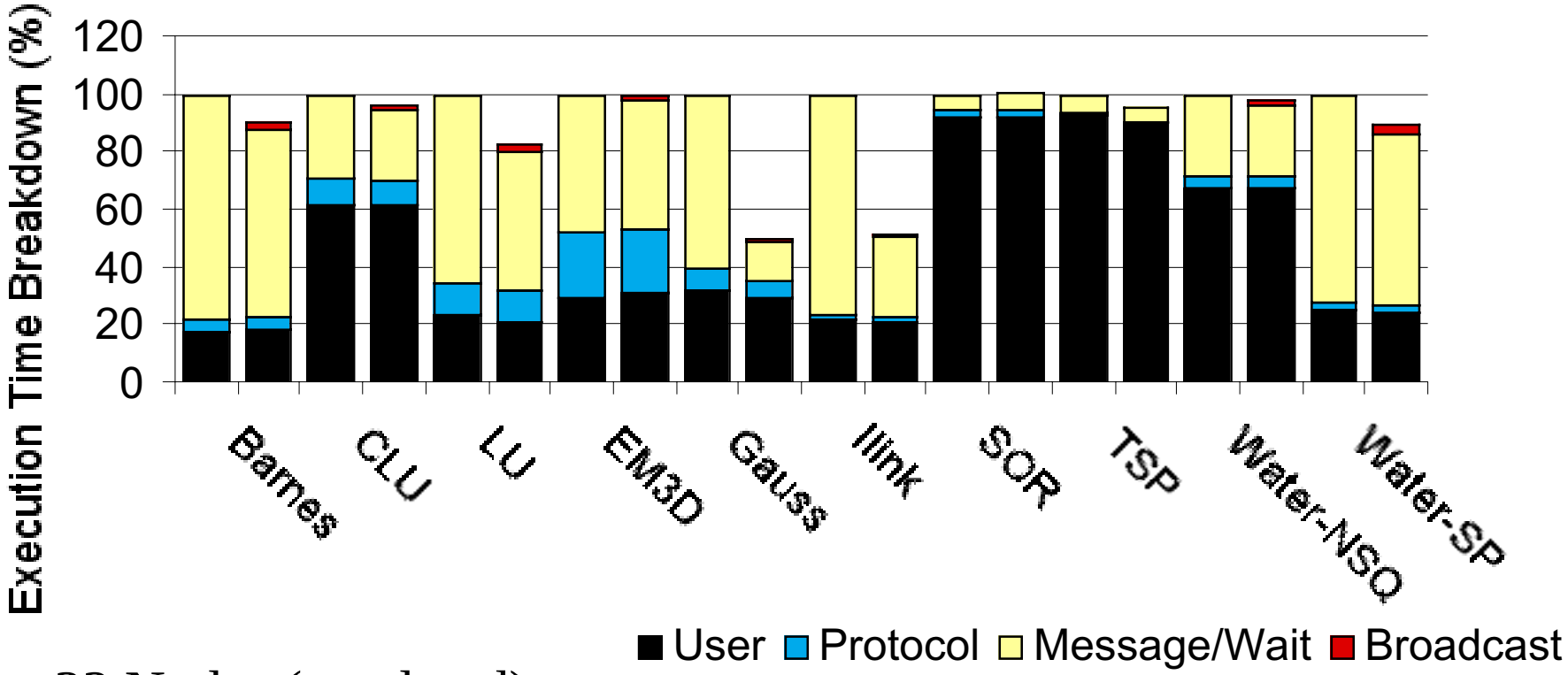
32 processors

Adaptive Broadcast

- Augment Cashmere with special broadcast buffers (don’t map shared memory directly)
- Use for widely-shared data
  - » multiple requests for page in a single synchronization interval
- Little performance improvement (<13%) on 8 nodes
- Sometimes significant improvement (up to 51%) on 32 (emulated) nodes

Relative Performance

Left, right bars: Cashmere, Cashmere-ADB



32 Nodes (emulated)

What Goes in the Network Address Space?

- Message buffers only
  - » Address space need not be huge
  - » Free home migration possible
- Application data
  - » Makes home node migration expensive
  - » Would benedit from remote reads and from very large network address space
  - » Requires pinning or dynamic virtual mappings
- Jury is still out; merits further study

Summary

- S-DSM a qualified success: familiar, simple, good for irregular apps or for SMP / CC-NUMA interoperability
- Significant opportunities for new communication architectures
  - » Fast interrupts
  - » Broadcast (used with discretion)
  - » Remote reads, atomic ops
  - » Very large network address space, dynamic virtual mappings
- Shows promise for geographically distributed systems: InterWeave

InterWeave Motivation

- Convenient support for “satellite” nodes
  - » remote visualization and steering (Astrophysics, ICF)
  - » client-server division of labor (datamining)
- ★ True distributed apps
  - » intelligent environments (AI group)
- Speedup probably not feasible; convenience the principal goal

InterWeave Overview

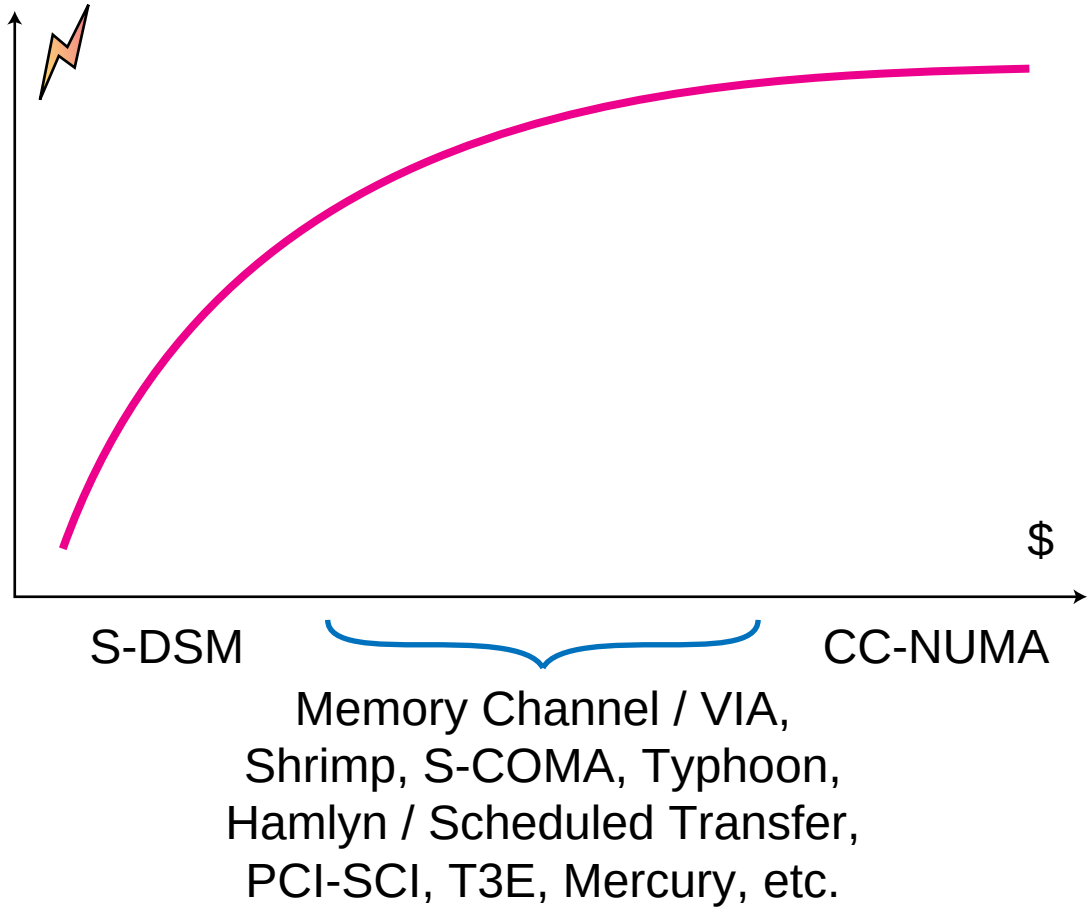
- Sharing of persistent versioned segments (heaps), named by URLs
- User-specified relaxed coherence; reader-writer locks
- Hash-based consistency
- Full support for language and machine heterogeneity, using IDL and pointer swizzling
- Cashmere functions as a single InterWeave node
- Fully compatible with RPC/RMI; supports true reference parameters

InterWeave Status

- Prototype running on Alpha/Tru64, Sun/Solaris, x86/Linux and Win2K (soon to be on Power-4/AIX);
- C, C++, Java, Fortran
- Funding from NSF & DOE/LLE
- Applications:
  - » Astroflow (stellar simulation)
  - » Barnes-Hut visualization
  - » incremental sequence mining
  - » distributed calendar
- In the works:
  - » distributed multi-player game (MazeWars)
  - » distributed object recognition (Smart Medical Home)

more info at
http://www.cs.rochester.edu/research/interweave

The Price-Performance Curve



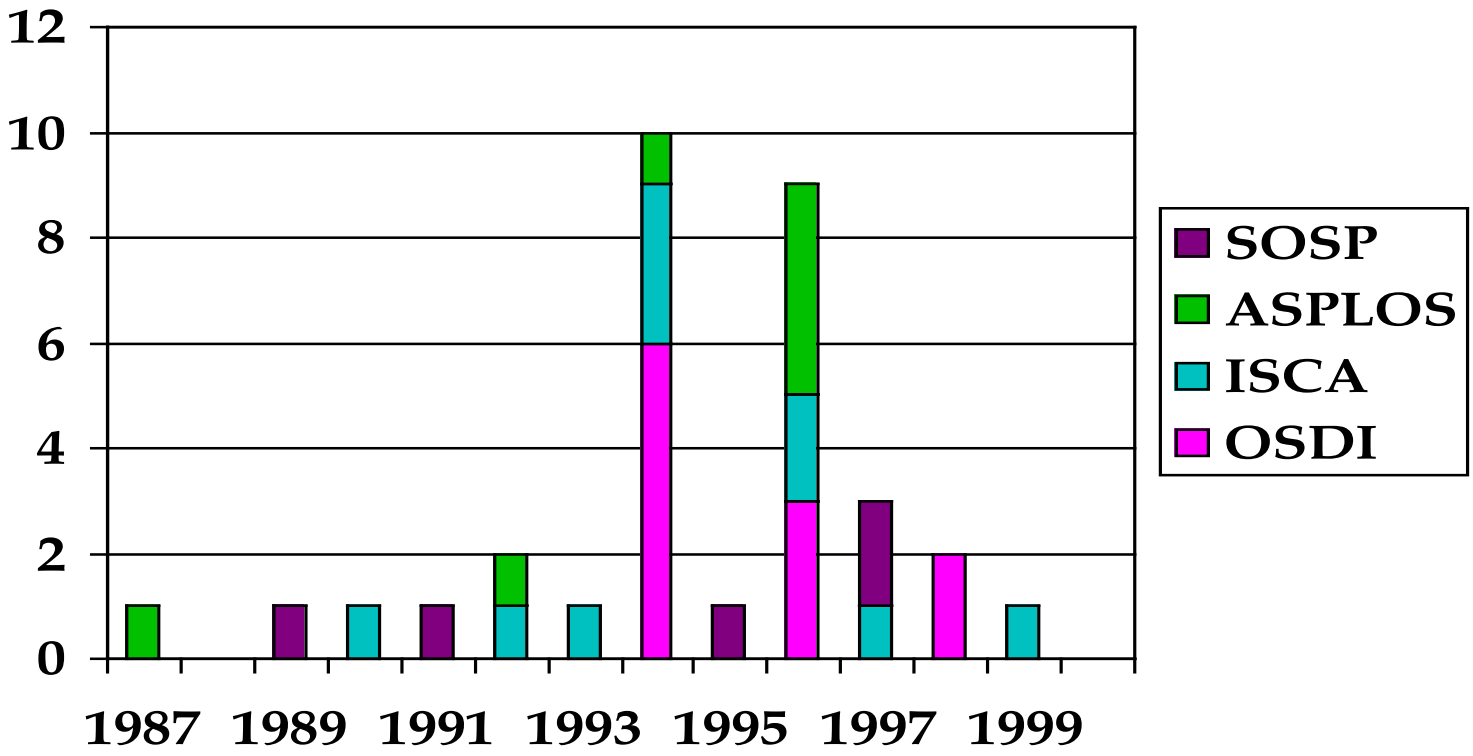
S-DSM may maximize “bang for the buck” for shared-memory parallel computing

The Fundamental Caveat

- Performance is maximized by unlimited programmer time coupled with a programming model that directly models the underlying hardware.
- S-DSM will not displace MPI at the National Labs anytime soon.

- Cf. the 1960s battle over compilation

Papers at Leading Conferences



Anybody see a trend?

Microbenchmarks

- AlphaServer 4100 cluster: 32 procs, 8 nodes
  - » Alpha 21164A 600 MHz
  - » Memory Channel II SAN
- Round-trip null message latency: 15 μsecs

	MC features (μ_{secs})	Explicit messages (μ_{secs})
Diff	31-129	70-245
Lock Acquire	10	33
Barrier (32 procs)	29	53

Segment Creation

```
IW_handle_t h = IW_create_segment (URL);
IW_wl_acquire (h);
my_type* p = (my_type *) IW_malloc (h, my_type_desc);
*p = ...
IW_wl_release (h);
```

- Communicates with server to create segment (checking access rights)
- Allocates local cached copy (not necessarily contiguous)
- Can follow pointers to other segments

Coherence

- Relaxed reader-writer locks
  - » Writer grabs current version (does *not* exclude readers)
  - » Reader checks to see if current cached copy (if any) is "recent enough"
- Multiple notions of "recent enough"
  - » e.g. immediate, polled, temporal, delta, diff-based
  - » from Beehive [Singla97], InterAct [LCR '98]
- Cache whole segments; server need only keep most recent version, in machine-independent form
- Diff-based updates (both ways) based on block time stamps

Consistency

- Need to respect happens-before
- Invalidate cached version of A that is older than version on which newly-obtained version of B depends
- Ideally want to know entire object history
- Can approximate using hashing
  - » slot *i* of vector contains timestamp of most recent antecedent hashing to *i*
  - » invalidate A if B.vec[hash(A)] is newer

InterWeave Related Work

- Distributed objects
  - » Language-specific (Emerald/Amber/VDOM, Argus, Ada, ORCA, numerous Java systems)
  - » Language-neutral (PerDiS, Legion, Globe, DCOM, CORBA, Fresco)
- Distributed state (Khazana, Active Harmony, Linda)
- Metacomputing (GLOBUS/GRID, Legion, WebOS)
- Heterogeneity, swizzling (RPC, LOOM, Java pickling)
- Multiple consistency models (Friedman et al., Agrawal et al., Alonso et al., Ramachandran et al., web caching work)
- Transactions, persistence (Feeley et al., Thor)