IMPLEMENTATION OF CASHMERE

Michael L. Scott
Computer Science Department
University of Rochester
Workshop on Scalable Shared-Memory Multiprocessors
October 1996

Abstract

Outline

- Motivation
- Key Ideas
- Memory Channel Overview
- Implementation Overview
- Current Work
- Future Plans

Cashmere People

- Michael Scott
  - Leonidas Kontothanassis (DEC CRL)
  - Galen Hunt
  - Rob Stets
  - Maged Michael
  - Natassa Ailamaki
- Sandhya Dwarkadas
  - Nikolaos Hardavellas
  - Sotirios Ioannidis
- Wei Li
  - Michal Cierniak
  - Srinivasan Parthasarathy
  - Mohammed Zaki

- Alex Poulos
- Wagner Meira

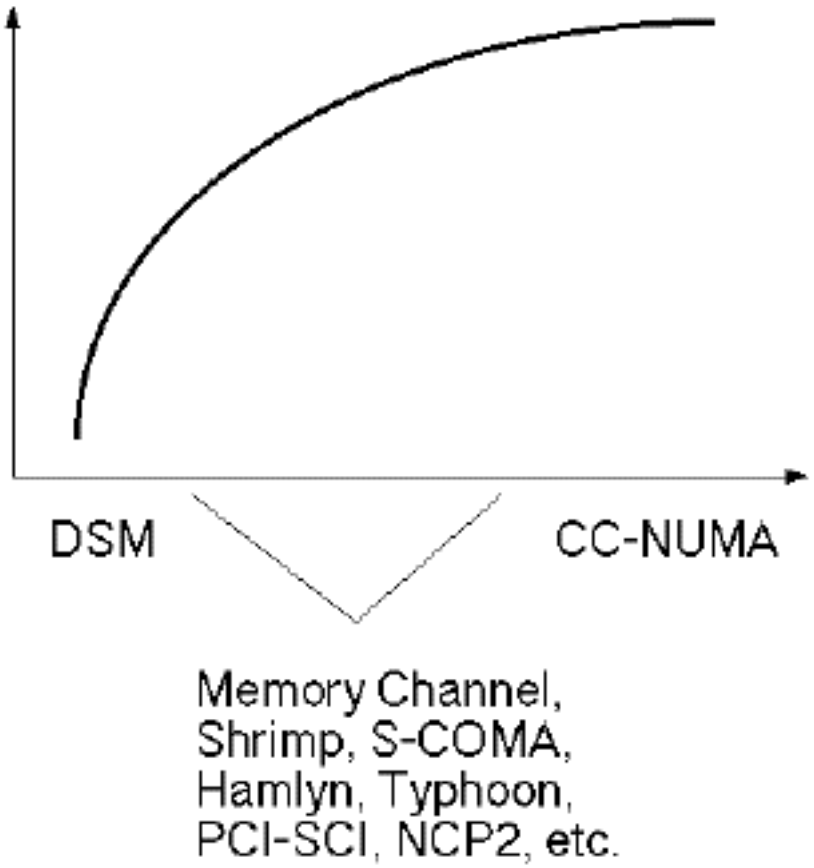
Motivation

- Shared memory programming model
- Wide hardware spectrum
  - DSM/SVM
  - full hardware coherence
  - options in-between

Hardware is faster, but software

- is cheaper
- can be built faster (sooner to market, faster processors)
- can use more complex protocols
- is easier to tune/fix/enhance
- is easier to customize

The Price/Performance Curve

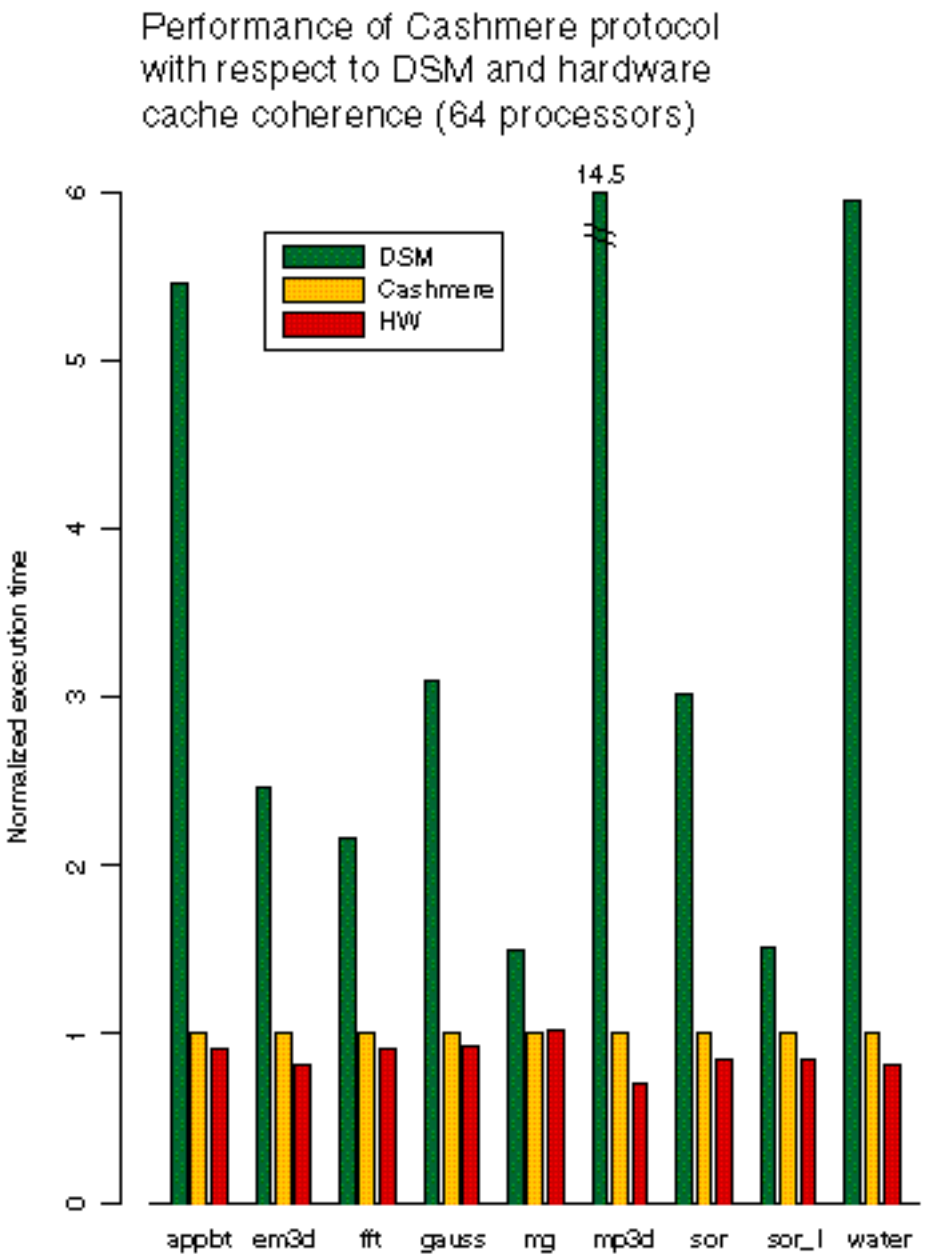


Q: How should coherence work given very low latency user-level messages?

Key Ideas in Cashmere

- multi-writer release-consistent
  - lazy, but not LRC
- write-through to home copy
  - no twins and diffs (cf. AURC)
  - dynamic choice of home node (first touch after initialization)
- directories
  - no intervals and timestamps
  - Each node has:
    - sharing set for pages for which it is home
    - write notices for remote pages that are currently mapped
- Wait for write-through at release
- send write notices
- Invalidate as necessary at acquire
- Re-map on page fault

Works very well in simulation:



BUT:

This was simulation, on a network that doesn't match any current commercial hardware.

--> Implementation based on Digital's *Memory Channel* for PCI

- 4x8 processor testbed
- remote-write API (no reads)
- I/O space (uncached)
- no inter-node coherence, but NI coherent with local processors
- global address space; both ends mapped
- VM protection

- 4 us user-user latency
- ~30 MB/sec per-link bandwidth (~130 MB/sec aggregate)

The Good News

- reasonable per-link bandwidth
- very low latency
- low cost (for the network anyway :-)
- hardware quite reliable
- multicast capability

The Bad News

- no remote reads
- low aggregate bandwidth (at present)
- high page fault and signal overhead in OSF Unix
- some resource recovery problems in early kernel software

Cashmere MC Highlights

As in simulation, except:

- copy to local on page fault (HPCA 96)
    - receive mapping on home node; transmit mapping elsewhere
- ``doubled writes" in software (not stack or global refs)
    - 4 instruction overhead
- replicated directory
    - read locally
        - no lock needed -- entries small
    - broadcast updates (HW support)

- requires lock (13 ms)

How Should We Read Pages?

- interrupts (ouch)
- v. protocol processor (current)
- v. SW polling (future?) (cf. Shasta)

A protocol processor can also:

- batch-execute directory modifications (reduction of sharing set) on behalf of remote acquirer
- do most of the work at local release
  - propagation of write notices
  - updates of directory entries
  - compute processor needn't stall
- track access patterns and prefetch?

Current Priorities

- performance tuning (and bug fixes!)
- comparison of
  - TreadMarks (intervals and diffs)
  - Cashmere (directories and write-through)

  Other options:
  - AURC (intervals and write-through)
  - directories and diffs
- multi-level coherence protocol
  - exploit HW coherence within each node
    - problem: how to reconcile remote changes with copy in use by another local processor
  - message-based on larger net

Other Future Work

- compiler integration
  - prescriptive (e.g. to aggregate messages)
  - descriptive (hints)
- fault tolerance (good results for TreadMarks)
- heterogeneity
- threads
- network interface design
- OS issues (scheduling, placement, naming, etc.)

Link to main Cashmere project page

Last Change: 7 November 1996 / scott@cs.rochester.edu

Implementation Of Cashmere

Michael L. Scott (speaker)
Wei Li
Sandhya Dwarkadas
Leonidas Kontothanassis*
Galen Hunt
Maged Michael
Robert Stets
Nikolaos Hardavellas
Wagner Meira
Alexandros Poulos
Michal Cierniak
Srinivasan Parthasarathy
Mohammed Zaki

University of Rochester
Computer Science Department
Rochester, NY 14627-0226
scott@cs.rochester.edu

* Digital Equipment Corporation
Cambridge Research Laboratory

The Cashmere project attempts to capture the ``knee of the curve'' in price-performance for shared-memory parallel computing: it exploits recent advances in local-area networks that provide low-latency, user-level access to remote locations in hardware, but implements coherence in software. The project has recently moved from simulation to implementation, using a 32-processor Alpha-server cluster (eight 4-processor nodes) on DEC's Memory Channel network. This talk will focus on the Cashmere implementation and on early experience as a Memory Channel field test site.

The Cashmere coherence protocol is characterized by

1. multi-writer lazy release consistency,
2. directories to keep track of who is sharing pages, and
3. update merging via write-through to a unique main-memory copy of each page.

Like many software coherent systems, Cashmere uses virtual memory protection at the granularity of pages to catch changes to the set of sharing processors, and to perform invalidations at synchronization release points. We have adapted the protocol to several different hardware variants. For the Memory Channel, which does not support loads from remote locations, we replicate pages to all nodes in which a

processor needs access.

Simulation studies, reported at *HPCA-1*, *Supercomputing '95*, and *HPCA-2*, indicate that on an idealized remote-access network, the Cashmere protocol can achieve dramatic performance improvements over twin-and-diff-based distributed shared memory, and can in fact approach the performance of full hardware coherence. Two practical issues limit the extent to which we can duplicate these results in our prototype implementation. First, cross-sectional bandwidth in the first-generation Memory Channel, while impressive at approx. 100MB/sec, is far short of what can be achieved in tightly-coupled multiprocessors such as the T3E or Paragon. Second, OS overheads, which could in theory be relatively modest, are substantial in OSF-1. In particular, the need to create a separate ``memory object'' for every remote-mapped page places stresses on the VM system unanticipated by its designers.

Work on Cashmere is proceeding on several fronts:

1. We are extending the coherence protocol to encompass a three-level system consisting of hardware coherence within SMP nodes, hybrid hardware/software coherence within LANs with remote memory access, and software-only coherence on message-passing networks. Longer-term, we expect to encompass heterogeneous processor architectures as well.
2. We have developed a low-overhead fault-tolerance mechanism for twin-and-diff-based coherence, and are attempting to adapt it to Cashmere.
3. We are integrating Cashmere with Rochester's Carnival system for performance prediction and analysis. We plan to use feedback from performance measurements to tune the coherence protocol, automatically, to the needs of individual applications.
4. We are integrating run-time, ``behavior-driven'' coherence with compile-time static analysis. We expect this integration to be increasingly important in our work.

This research is supported by NSF grants numbers CCR--93--19445, CCR--94--09120, and CDA--94--01142; by ARPA contract number F19628--94--C--0057, subcontract 3531427; and by an external research grant from Digital Equipment Corporation.

Back to slides Link to main Cashmere project page

Last Change: 12 November 1996 / scott@cs.rochester.edu