Coherence Optimization in Multiprocessors

Work by Kaxiras and Goodman (HPCA-5)

Distributed shared-memory multiprocessors

Directory-based cache coherence protocol

Sharing pattern optimization:

Migratory, Wide, Producer-Consumer

Sharing pattern identification:

Dynamic identification

Previous work: address-based techniques

- examine history of data (what happens to the data?)
- adaptive protocols, coherence message prediction

Contribution

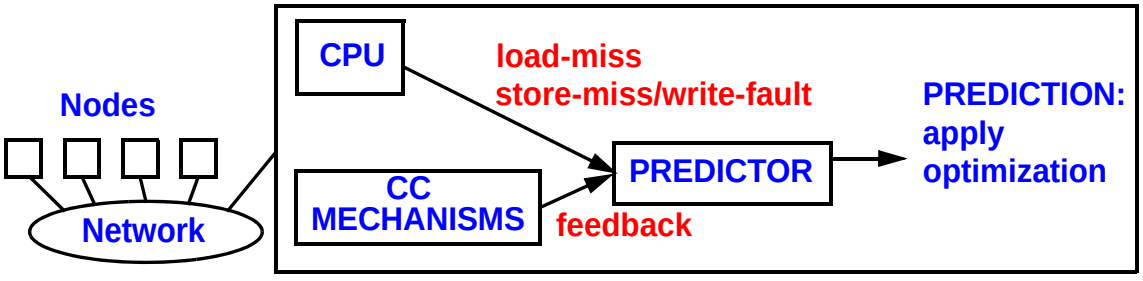
Novel approach to identify (& optimize) sharing patterns:

Instruction-based prediction

Discover what (load/store) instructions are trying to do

Benefits: few resources to capture instruction behavior

fast to adapt



Migratory Sharing

Migratory data move from processor to processor:

- Read-modify-write by a processor at a time
- Read & write latency for new processor

Optimization: migrate data in one step

- Read with write permission (& invalidate old node)
- No write latency or traffic

Previous work: adaptive protocols (address-based)

- Directory detects pattern: R_a**W_a** R_b**W_b** R_c**W_c**...
- Applies migratory optimization

Cost:

Must remember last writer for every data block

Instruction-Based Prediction

Identify instructions responsible for migratory sharing:

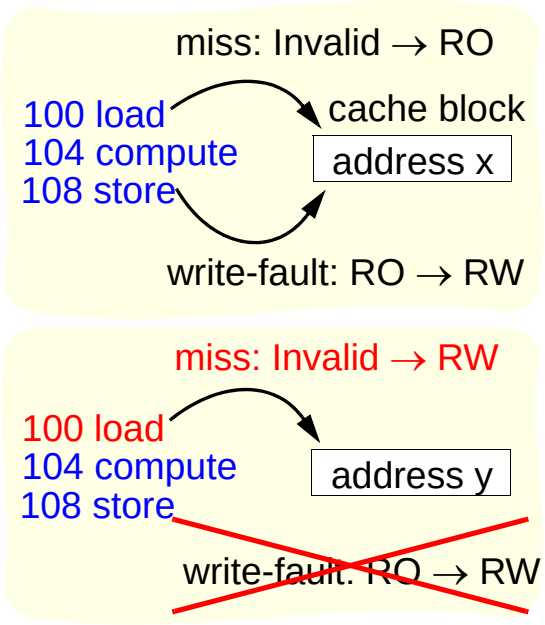
- Predict a store-write-fault follows a load-miss
- Detect Write-after-Read on cache blocks
- Typical behavior of migratory blocks
- Migratory optimization: convert read to write

Cost:

Predictor per node

+ few predictor entries for programs examined
+ only track load instructions accessing migratory data

Migratory Sharing Prediction Example

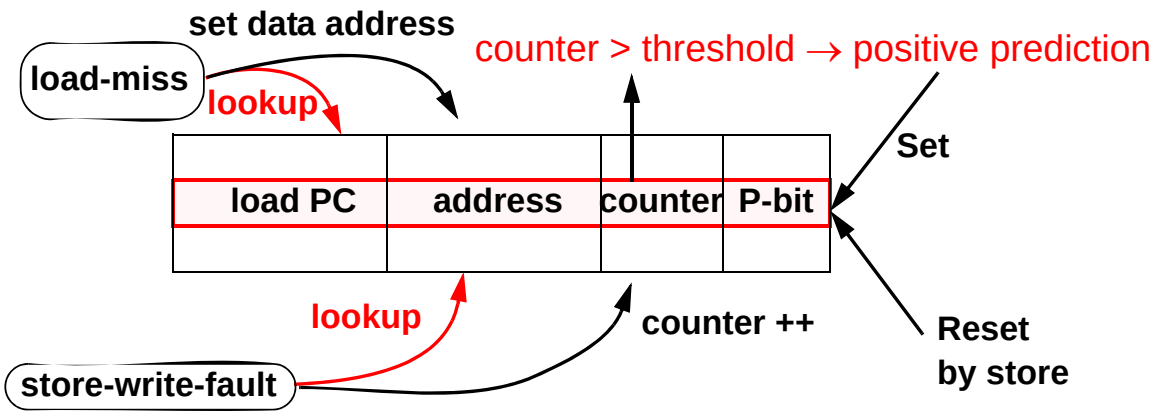


Constraint for migratory data:

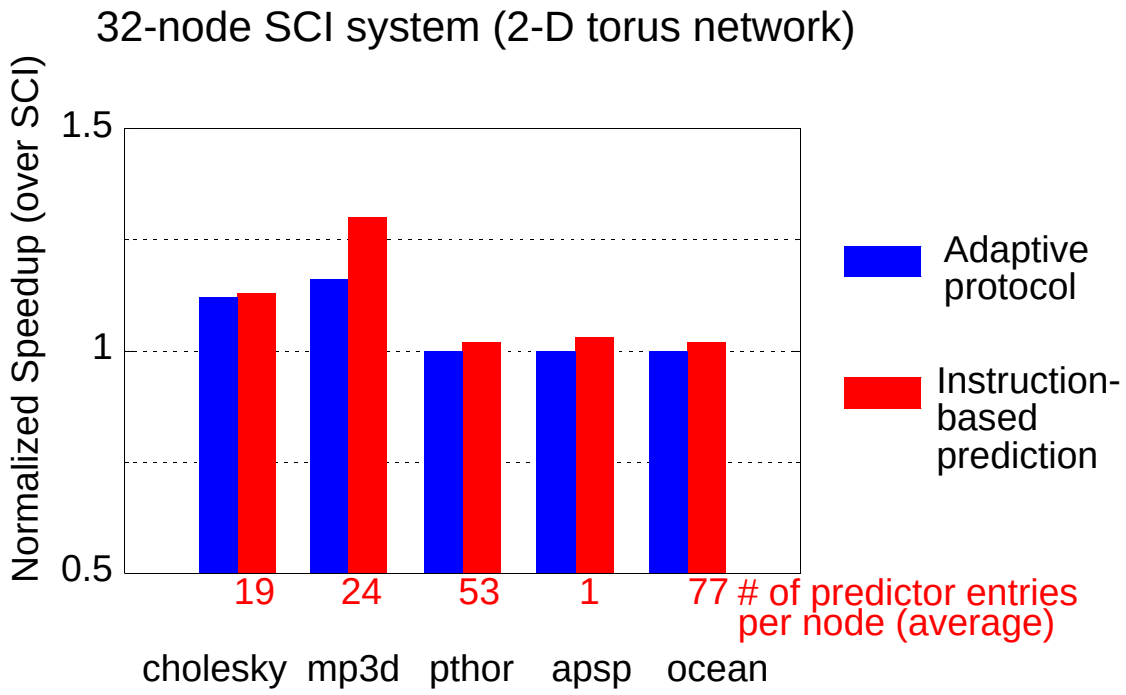
Cache-block 2nd or only copy unaffected between load-miss & store

Migratory Sharing Predictor

A simplified implementation



Results for Migratory Sharing



Wide Sharing

Widely Shared Data: read by many, written frequently

Very expensive to access (worse with system size)

Hot spot and network congestion near home node

Many invalidations waste B/W

WS Optimization

- Tree protocol (STEM, STP, TD, etc.)
- GLOW: scalable reads + scalable writes + network locality
- USE GLOW only for WSD

Dynamic WS identification:

address-based vs. instruction-based

Address-based Identification for WS

Directory Detection

- Directory discovers WSD blocks
- Counts reads between writes
- Notifies nodes about the data
- Nodes remember WSD (storage cost)
- Nodes use GLOW when accessing WSD

Slow to adapt

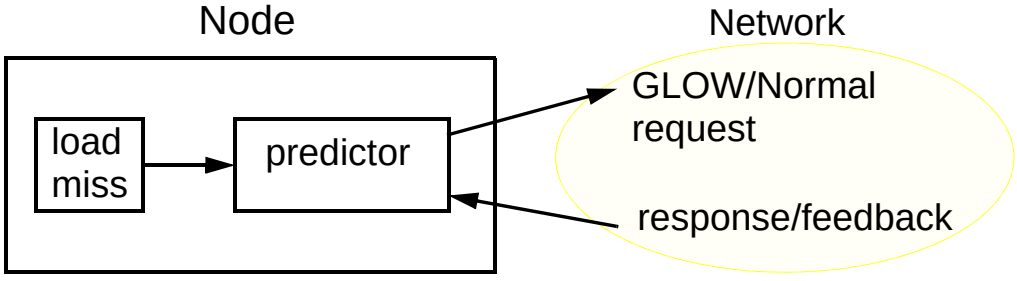
Instruction-Based Prediction for WS

Idea: discover load instructions accessing WSD

If load accessed WSD in the past, probably it will in the future

Track loads accessing WSD in a small predictor

probe predictor on load-miss
update predictor with response (feedback)



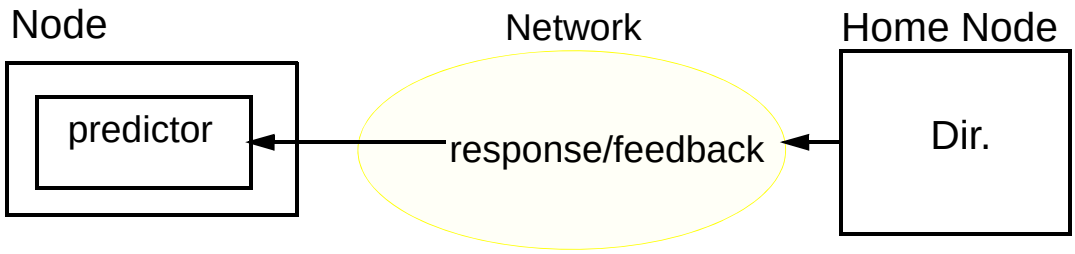
Instruction-Based Prediction for WS (Cont)

Feedback to determine if load accessed WSD:

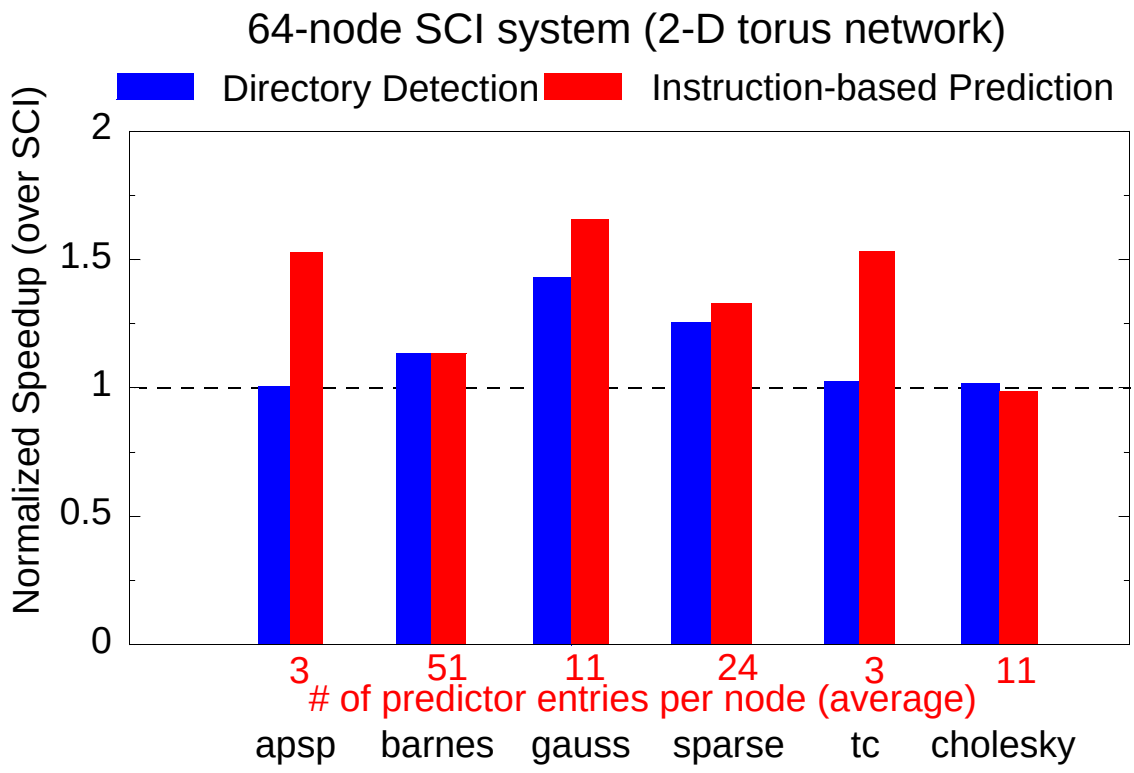
- Miss-latency (heuristic):
  Wide sharing → long latencies (congestion, hot spots)
- Directory-feedback:
  Directory informs about WSD blocks (Directory Detection)

Adapt-back

- miss-latency: delete predictor entries after a while
- directory feedback: accessing non-WSD



Results for Wide Sharing



Producer-Consumer Sharing

Producer-consumer sharing optimizations:

- Send data ASAP from producer to consumer(s)
- Update protocols, Competitive update, Data Forwarding

Dynamic Identification

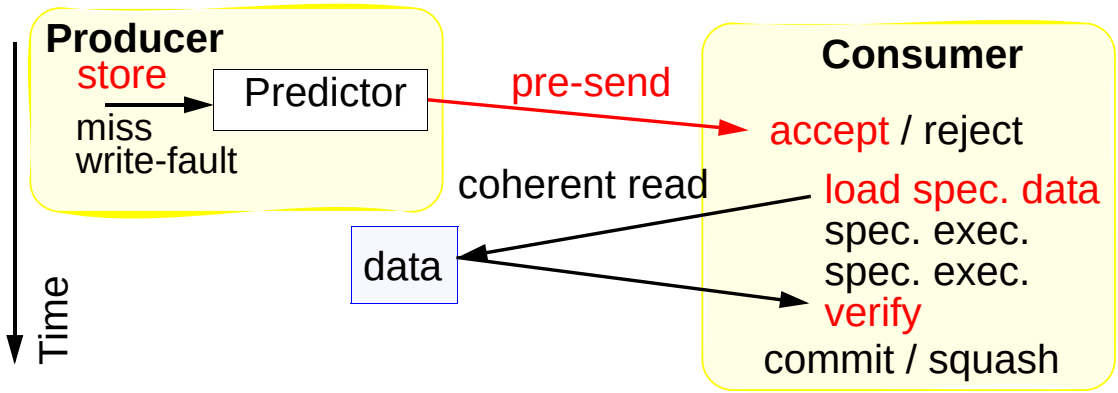
- Previous work: directory detects P-C data

Contributions:

- Instruction-based prediction:
  Predict consumers of store instructions
- Optimization: speculative pre-send

Optimization: Speculative Pre-Send

1. Predict consumer(s)
2. Send the data speculatively to consumer(s)
3. Consumer(s) can use data speculatively
   But have to verify data through normal CC-protocol



Instruction-Based Prediction for P-C

Predict set of new consumers of a store

History: set of previously invalidated nodes by the store

Last-prediction

...

store → invalidate (a_1, ... a_i) } overlap

store → invalidate (b_1, ... b_j) } overlap

store → invalidate (c_1, ... c_k)

predict (c_1, ... c_k)

Intersection-prediction

...

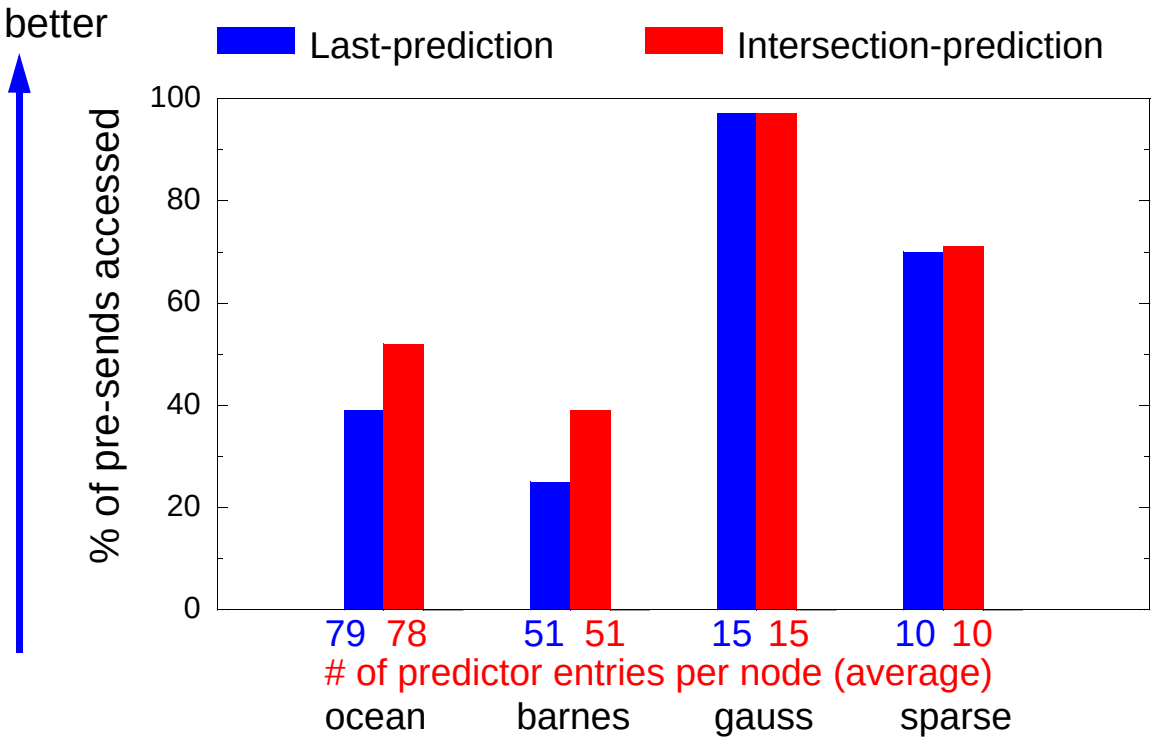
store → invalidate (a_1, ... a_i)

store → invalidate (b_1, ... b_j)

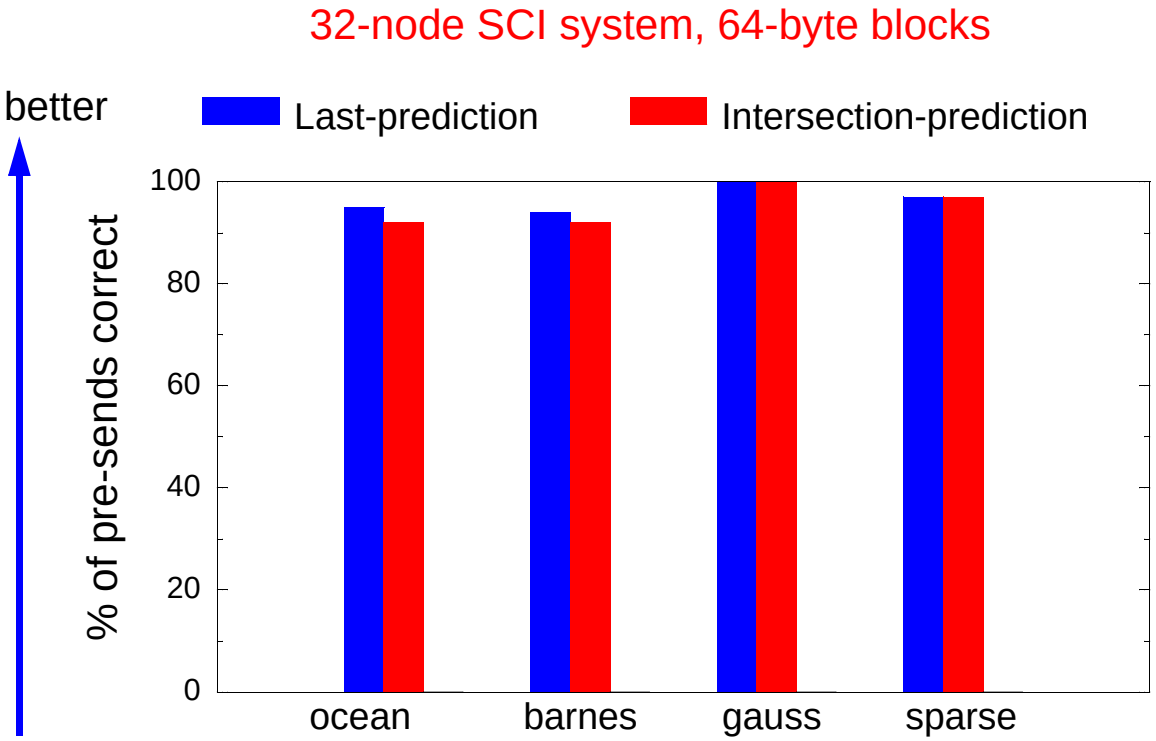
predict (a_1, ... a_i) AND (b_1, ... b_j)

Results for P-C Sharing

32-node SCI system, 64-byte blocks



Results for P-C Sharing



Summary

Instruction-based prediction:

Observe load/store history in relation to CC events

Predict future behavior

	Instruction-based Prediction	Address-based
Migratory Sharing	Few resources	Many resources
Wide Sharing	Fast to adapt	Slow to adapt

Producer-consumer + speculative pre-send

- Simple predictors (few resources/room for improvement)
- Low mis-speculation rates