Predicting Coherence Communication by Tracking Synchronization Points at Run Time

Socrates Demetriades
and Sangyeun Cho
Coherence Communication

The result of data sharing between threads, when those run on a shared memory multiprocessor with coherent private caches.
Coherence Communication

- Block A exclusive to T0
- T13: Request to share A
- T13 “communicates” with T0.
- Block A is copied to T13.

[Shared Memory Model / Write-Invalidate Coherence Protocol]
Coherence Communication

- Block A is shared.
- T13: Request for exclusive ownership.
- T13 “communicates” with T0 & T6.
- Invalidate copies.

*Communicating Misses*: all request that must communicate with at least one other core.

[Shared Memory Model / Write-Invalid Coherence Protocol]
Communication Overheads

Directory-based Coherence Protocol

Indirect Miss to the Directory
=> Increase Miss Latency

Snoop-based Coherence Protocol

Broadcast to all
=> Increase traffic
Communication Prediction

Trade-Off
Accuracy vs Extra traffic
Traditional Prediction Approaches

1. Simple temporal-based prediction.
   - Locality between consecutive misses.
2. ADDRESS-based prediction.
   - Locality based on the address of the request.
3. INSTRUCTION-based prediction.
   - Locality based on the static store/load instr.
Contributions of this Work

Synchronization Point based Prediction (SP-prediction)

Inter-thread communication caused by coherence transactions is tightly related with the synchronization points in parallel execution.

• **Main Idea:** Associate the communication behavior with synchronization points and utilize this association to predict the destination of misses.

• **Main Advantage:** Has very low storage cost, yet delivers relatively high performance.
Outline

- Introduction
- Motivation & Observations
- SP-Prediction
- Evaluation
- Conclusion
Why Synchronization Points?

[Diagram showing synchronization points on four cores: Core 1, Core 2, Core 3, Core 4. The diagram includes synchronization points such as BARRIER, LOCK, UNLOCK, BARRIER, SIGNAL, and WAIT.文中提到的备忘录：Pthread notation]
Synchronization Epochs

[Diagram showing synchronization points and communication distribution for Core 0]

Communication Distribution of Core 0
(full interval)

Communication Distribution of Core 0
(different sync-epochs)

[Benchmark: Bodytrack / 16-threads]
Sync-Epoch Dynamic Instances

[Benchmark: Bodytrack / 16-threads]
Outline

- Introduction
- Motivation & Observations
  - SP-Prediction
- Evaluation
- Conclusion
SP-prediction – Overview

- Monitor destinations of each miss on each core.
- Extract communication signatures for each sync-epoch.
- Store and later reuse those signatures to predict misses in future sync-epoch instances.
- When initial predictions do not exist or are inaccurate, reconstruct the signatures within the sync-epochs.

- Sync Points must be exposed to the hardware so it can sense the beginning and end of sync-epochs.
  - A dedicated instruction must be inserted at the calling location of the synchronization point.
  - PC, lock variable and type must be extracted and pass to a history table.
SP-prediction: History-based

CORE 0

Sync-Point PC | PREDICTOR
---|---
A | [hot comm. set ]
SP-prediction: History-based (for Locks)

- **Lock Acquire:**
  - Retrieve Predictor

- **Lock Release:**
  - Store Core Id

- **SP Table**:
  - LOCK ADDR: A
  - PREDICTOR: [C0]
SP-prediction: First Sync-Epoch Instances

- No history exists for this point (first instance).
- Allow some warm-up time and then extract an “early” hot communication set.
- Use the set as a predictor for the rest of the interval.
SP-prediction: Adaptive Recovery

- Sync-point is detected, predictor is retrieved from SP-table
- Start using predictor for each miss, with high confidence.
- If prediction accuracy drops low, extract a new hot communication set on the spot.
- Continue predictions based on the new predictor.
Why SP-prediction

• In contrast to simple temporal prediction, it exploits application-defined interval-based communication localities.
  – No restricted on temporal locality among consecutive misses.
  – Can adapts faster to the changes.
  – Can recall old and forgotten communication patterns.

• Compared to address and instruction based prediction, it has very low storage requirements.
  – SP table must holds, on average 5-30 static sync points for a given application.

• Take advantage of the existing programming paradigm while being transparent to the programmer.
Outline

- Introduction
- Motivation & Observations
- SP-Prediction
- Evaluation
- Conclusion
Evaluation Methodology

- Simulated Machine Configuration (based on simics)
  - In order core
  - Private L1/L2
  - DIR slice.
  - Network logic
  - Coherence Logic

- Workloads
  - From Splash 2 & PARSEC Suites.
  - # static sync-epochs: 5-30
  - # dynamic sync-epochs: 22-20,000 (for the evaluated input sizes)

- SP-prediction implemented on top of Baseline Directory.
Results: Prediction Accuracy

<table>
<thead>
<tr>
<th>Average</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination Set Size (actual)</td>
<td>1.2</td>
</tr>
<tr>
<td>SP-prediction Set Size</td>
<td>2.6</td>
</tr>
</tbody>
</table>
Results: Latency & Bandwidth

- Latency: 13%
- Bandwidth: 18% (5%)
Results: Latency & Bandwidth

Execution Time Improvements: 7% on average.

Additional Energy Dissipation: ~7% (25% NoC+Cache lookups). (more than 90% lower compared to broadcasting) Can be reduced to <3% when combined with other low cost technique.
Comparison with other Predictors

% incurring Indirection

% Additional Bandwidth per Miss

- Last 2 misses
- ADDR-based
- INSTR-based
- SP-prediction
- DIRECTORY

BEST POSITION
Comparison with other Predictors

- Last 2 misses
- ADDR-based
- INSTR-based
- SP-prediction
- DIRECTORY

% incurring Indirection vs. % Additional Bandwidth per Miss

Best Position

Prediction Table Storage

Infinite Entries
Comparison with other Predictors

PREDICTION TABLE STORAGE

512 ENTRIES

% Incuring Indirection

% Additional Bandwidth per Miss

- Last 2 misses
- ADDR-based
- INSTR-based
- SP-prediction
- DIRECTORY

BEST POSITION
Conclusions

• SP-prediction is a new, run-time and application-driven approach on communication predictability.

• Promotes very low storage requirements, an important property for emerging CMP implementations.

• Scales independent of core count and cache sizes.

• Takes advantage of the existing shared memory programming paradigm and current consistency models.
Thank you for your attention!