Characterization of TCC on Chip-Multiprocessors

Austen McDonald, JaeWoong Chung, Hassan Chafi, Chi Cao Minh, Brian D. Carlstrom, Lance Hammond, Christos Kozyrakis, and Kunle Olukotun

Stanford University
http://tcc.stanford.edu/
Take Away Points

• Parallel programming is hard
• Transactions make parallel programming easier
  – Knight ’86, Herlihy ’93...Ananian ’05, Moore ’05, Rajwar ’05
  – Transactional Coherence and Consistency

Contributions:

1. Present a simple implementation of TCC for CMPs.
   Address basic challenges and explore design options.
2. Performance is comparable with a MESI-based CMP.
   Gain the ease of TCC without significant loss of performance.
The Problems of Parallel Programming

- **Critical sections make programming hard**
  - Coarse-grained locks: serialization
  - Fine-grained locks: deadlocks
  - Poor composability, not fault tolerant

- **Parallel programming environment complex**
  - Consistency models are complex
  - Performance tuning requires detailed and difficult-to-acquire data
Enter Transactions...

- Transactions provide non-blocking synchronization
  Large, programmer-defined atomic regions.

- Transactions simplify programming environment
  - Simplify reasoning about consistency
  - Performance tuning is easier (Chafi ’05 at ICS)

- Transactions enable speculative parallelism
  - Programmers identify suspected parallel regions
TCC Execution Model

“All transactions, all the time.”

CPU 0

Execute Code

Arbitrate

Commit

... ld 0xdedd
... st 0xbeef

CPU 1

Execute Code

Arbitrate

Commit

... ld 0xaaaa
... ld 0xbbbb
... ld 0xbeef

CPU 2

Re-Execute Code

Violate

... ld 0xbeef
... ld 0xbeef

CMP Environment

- CMP with simple CPUs
  - write-back L1
  - shared L2
  - two wide, pipelined logical buses
    - 16B bus, 3 cyc pipelined arbitration, 3 cyc pipelined transfer
- Same CMP setup for TCC and MESI

Changes for TCC support
An Architecture for TCC

Speculative state stored in caches

Speculatively-Read Bits:
\[ \text{ld 0xdeadbeef} \]

Speculatively-Modified Bits:
\[ \text{st 0xc0ffee} \]

Commit:
Read pointers from Store Address FIFO, flush data with SM bits set

Violation Detection:
Compare incoming address to SR bits

September 19, 2005
Other Implementations

- Speculative state in lower-level caches
  - L2 and main memory
- Parallel commit
  - More than one transaction commits at once
- Commit in place
  - Flush writes only when needed

Options may be useful for large-scale TCC. Simple is good enough in CMPs.
Architectural Options

- We explored some architectural options
  - Double buffering
    Simple, single buffering is sufficient
  - Invalidate vs. update
    Doesn’t matter for our applications
  - Word- vs. line-level granularity
    Word-level is better due to false sharing
  - Associative Overflows...
Associative Overflows

- Limited speculative state tracking
- Capacity overflows rare (Rajwar '05 handles them)
- Associative overflows the common case
  - Can’t afford an expensive mechanism
- Simple victim cache
The Rest of the Talk

• Differences between TCC and MESI
• Performance Comparison
  – Bandwidth Usage
  – Speedup Summary
  – In depth: MP3D
    • The advantages of TCC on a difficult-to-parallelize program
## Differences between TCC and MESI

<table>
<thead>
<tr>
<th></th>
<th>TCC</th>
<th>MESI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Synchronization</strong></td>
<td>Non-blocking, large, multi-object regions</td>
<td>Blocking, small regions</td>
</tr>
<tr>
<td><strong>Speculation</strong></td>
<td>Speculatively parallel</td>
<td>None in basic form</td>
</tr>
<tr>
<td><strong>Coherence Frequency</strong></td>
<td>Communicates often and more—large chunks</td>
<td>Communicates only when needed</td>
</tr>
<tr>
<td><strong>Coherence Granularity</strong></td>
<td>Word-level</td>
<td>Line-level→false sharing</td>
</tr>
</tbody>
</table>
Performance Comparison

Comparing TCC to MESI...

- Scalability on applications tuned for MESI
  - Execution-driven simulation of SPECfp, SPLASH, SPLASH-2, SPECjbb
  - Measures sustained performance vs. ease of parallelizing
Bandwidth Usage

- Broadcasting commits does not hinder performance in a CMP
  - On-chip bandwidth sufficient
Comparing TCC to MESI...
In Depth: MP3D

- Rarefied hypersonic flow simulator
  - Monte Carlo
- Molecules statically allocated to processors
  - Causes false sharing
- Barrier-based synchronization (not many locks)
MP3D Results

- Execution time in MP3D.
Conclusions

- Transactions simplify parallel programming

Contributions:
- We evaluated TCC for CMP systems
  - TCC can be efficiently implemented in a simple manner
  - Associative overflows handled with a simple victim cache
- Compared performance against a MESI-based CMP
  - TCC performs similarly
  - Bandwidth requirements are not excessive
- TCC enables the ease of transactions without hindering performance
Questions?

(whew!)