Drinking from Both Glasses: Adaptively Combining Pessimistic and Optimistic Synchronization for Efficient Parallel Runtime Support

Man Cao
Minjia Zhang
Michael D. Bond

The Ohio State University
Dynamic Analyses for Parallel Programs

- Data Race Detector, Record & Replay, Transactional Memory, Deterministic Execution, etc.

- Performance is usually bad!
  - several times slower

- Fundamental difficulties?
Cross-thread dependences

- Crucial for dynamic analyses and systems
- Capturing cross-thread dependences
  - Detecting
    e.g. data race detector, dependence recorder
  - Controlling
    e.g. transactional memory, deterministic execution
Typical approach

• Per-object metadata (state)
  – E.g. last writer/reader thread

• At each object access:
  – Check current state
  – Analysis-specific action
  – Update state if needed
  – Perform the access
Typical approach

- Per-object metadata (state)
  - E.g. last writer/reader thread
- At each object access:
  - Check current state
  - Analysis-specific action
  - Update state if needed
  - Perform the access

Atomically

How to guarantee?
Pessimistic Synchronization

• Used by most existing work
  – Data Race Detector
    • [FastTrack, Flanagan & Freund, 2009]
  – Atomicity Violation Detector
    • [Velodrome, Flanagan et al., 2008]
  – Record & Replay
    • [Instant Replay, LeBlanc et al., 1987]
    • [Chimera, Lee et al., 2012]
Pessimistic Synchronization

LockMetadata()
Pessimistic Synchronization

LockMetadata()

Check and compute new metadata
Pessimistic Synchronization

- LockMetadata()
- Check and compute new metadata
- Analysis-specific actions
Pessimistic Synchronization

- $\textit{LockMetadata()}$
- Check and compute new metadata
- Analysis-specific actions
- Program access
Pessimistic Synchronization

- **LockMetadata()**
- Check and compute new metadata
- Analysis-specific actions
- Program access
- **UnlockAndViewMetadata()**
Pessimistic Synchronization

- Synchronization on every access
- 6X slowdown on average
Optimistic Synchronization

- Used to improve performance
  - Biased Locking
    - [Lock Reservation, Kawachiya et al., 2002]
    - [Bulk Rebiasing, Russell & Detlefs, 2006]
  - Distributed Memory System
    - [Shasta, Scales et al. 1996]
  - Framework Support
    - [Octet, Bond et al. 2013]
Optimistic Synchronization

• Avoid synchronization for non-conflicting accesses
• Heavyweight coordination for conflicting accesses
Optimistic Synchronization (Cont.)

T1
- write check
- wr o.f
- write check
- wr o.f

T2
Optimistic Synchronization (Cont.)

T1
write check
wr o.f
write check
wr o.f

T2
read check
Optimistic Synchronization (Cont.)

T1
- write check
- wr o.f
- write check
- wr o.f

T2
- read check
- Analysis-specific action

safe point
Optimistic Synchronization (Cont.)

T1
- write check
- wr o.f
- write check
- wr o.f
- safe point

T2
- read check
- Analysis-specific action
- change metadata
- rd o.f
Optimistic Synchronization (Cont.)

T1

- write check
- wr o.f
- write check
- wr o.f
- safe point

T2

- read check
- Analysis-specific action
- change metadata
- rd o.f

- 26% on average with outliers
  - Expensive if there are many conflicting accesses
Optimistic synchronization performs best if there are few conflicting accesses.
Pessimistic synchronization is cheaper for conflicting accesses.
Drink from both glasses?

• Goal:
  – Optimistic sync. for most non-conflicting accesses
  – Pessimistic sync. for most conflicting accesses

• Our approach:
  – Hybrid state model
  – Adaptive policy
  – Support for detecting and controlling cross-thread dependences
Adaptive Policy

• Decides **when** to perform Pess → Opt and Opt → Pess transitions

• Cost—Benefit model
  – Formulates the problem

• Online profiling
  – Efficiently collects information and approximates the Cost-Benefit model
Cost—Benefit model

• Compares total time spent in transitions if an object were optimistic or pessimistic
  – Whichever takes less time is beneficial

\[ T_{\text{total\_Opt}} > T_{\text{total\_Pess}} \]

• Only relies on numbers (or just the ratio) of non-conflicting and conflicting transitions
Evaluation

• Implementation
  – Jikes RVM 3.1.3

• Parallel programs
  – SPEC JBB 2000 & 2005

• Platform
  – 32 cores (AMD Opteron 6272)
Performance

<table>
<thead>
<tr>
<th></th>
<th>Pure Pessimistic</th>
<th>Pure Optimistic</th>
<th>Adaptive</th>
</tr>
</thead>
<tbody>
<tr>
<td>eclipse6</td>
<td>770</td>
<td>270</td>
<td>326</td>
</tr>
<tr>
<td>hsqldb6</td>
<td>650</td>
<td>95</td>
<td>50</td>
</tr>
<tr>
<td>lusearch6</td>
<td>326</td>
<td>270</td>
<td>100</td>
</tr>
<tr>
<td>xalan6</td>
<td>650</td>
<td>430</td>
<td>170</td>
</tr>
<tr>
<td>avroras9</td>
<td>95</td>
<td>170</td>
<td>100</td>
</tr>
<tr>
<td>jython9</td>
<td>270</td>
<td>2,400</td>
<td>29,000</td>
</tr>
<tr>
<td>luindex9</td>
<td>430</td>
<td>210</td>
<td>430</td>
</tr>
<tr>
<td>lusearch9</td>
<td>170</td>
<td>120</td>
<td>29,000</td>
</tr>
<tr>
<td>pmd9</td>
<td>100</td>
<td>470</td>
<td>326</td>
</tr>
<tr>
<td>sunflow9</td>
<td>2,400</td>
<td>210</td>
<td>650</td>
</tr>
<tr>
<td>xalan9</td>
<td>29,000</td>
<td>270</td>
<td>95</td>
</tr>
<tr>
<td>jbb2000</td>
<td>2,400</td>
<td>430</td>
<td>170</td>
</tr>
<tr>
<td>jbb2005</td>
<td>210</td>
<td>120</td>
<td>100</td>
</tr>
<tr>
<td>geomean</td>
<td>120</td>
<td>470</td>
<td>29,000</td>
</tr>
</tbody>
</table>
Performance

Overhead (%)

Pure Pessimistic
Pure Optimistic
Adaptive

<table>
<thead>
<tr>
<th></th>
<th>eclipse6</th>
<th>hsqldb6</th>
<th>lusearch6</th>
<th>xalan6</th>
<th>avrora9</th>
<th>jython9</th>
<th>luindex9</th>
<th>lusearch9</th>
<th>pmd9</th>
<th>sunflow9</th>
<th>xalan9</th>
<th>jbb2000</th>
<th>jbb2005</th>
<th>geomean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Pess.</td>
<td>770</td>
<td>270</td>
<td>326</td>
<td>650</td>
<td>95</td>
<td>270</td>
<td>430</td>
<td>170</td>
<td>100</td>
<td>29,000</td>
<td>2,400</td>
<td>210</td>
<td>120</td>
<td>470</td>
</tr>
<tr>
<td>Pure Opt.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adaptive</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

27
Framework support

• Detecting cross-thread dependences
  – dependence recorder

• Key challenge
  – Identify the source location of a happens-before edge for a pessimistic conflicting transition
  – Current solution requires acquiring a lock and writing to remote thread’s log
Framework support (Cont.)

• Controlling cross-thread dependences
  – enforcing Region Serializability (in progress)

• Key challenge
  – Need to keep locking pessimistic objects until the end of a region
Framework support (Cont.)

• Controlling cross-thread dependences
  – enforcing Region Serializability (in progress)

• Key challenge
  – Need to keep locking pessimistic objects until the end of a region

• Possible solution
  – Defer unlocking of pessimistic objects until program lock releases
    • Helps dependence recorder
    • Simplifies instrumentation
Conclusion & Future work

• Hybrid, adaptive synchronization achieves better performance
  – never significantly degrades performance
  – sometimes improves performance substantially

• Future directions
  – Explore different adaptive policies (e.g. aggregate profiling)
  – Reduce instrumentation cost by deferring unlock operations of pessimistic synchronization
  – Apply to control cross-thread dependences