Legato: Bounded Region Serializability Using Commodity Hardware Transactional Memory

Aritra Sengupta
Man Cao
Michael D. Bond
and
Milind Kulkarni
Programming Language Semantics?

Data Races

C++ no guarantee of semantics – “catch-fire” semantics
Java provides weak semantics
Weak Semantics

Data data = null;
boolean done = false;

T1

data = new Data();
done = true;

T2

if (done)
    data.foo();
Weak Semantics

Data data = null;
boolean done = false;

T1

data = new Data();
done = true;

T2

if (done)
data.foo();

Null Pointer Exception
Weak Semantics

Data data = null;
boolean done = false;

T1

data = new Data();
done = true;

T2

if (done)
data.foo();

Null Pointer Exception

No data dependence:
Reordering effect
Need for Stronger Memory Models

“The inability to define reasonable semantics for programs with data races is not just a theoretical shortcoming, but a fundamental hole in the foundation of our languages and systems…”

Give better semantics to programs with data races
Stronger memory models

– Adve and Boehm, CACM, 2010
End-to-End Memory Models: Run-time cost vs Strength

Run-time cost vs Strength Diagram:

- SC
- DRF0 (C++/Java)
- (Unbounded) Synchronization-free Region Serializability
- Bounded Region Serializability
End-to-End Memory Models: Run-time cost vs Strength

Run-time cost vs Strength

Dynamically Bounded Region Serializability (DBRS)
Dynamically Bounded Region Serializability

SC+Atomicity of bounded regions\(^1\)

<table>
<thead>
<tr>
<th></th>
<th>DRF0</th>
<th>SC</th>
<th>DBRS</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsqldb6</td>
<td>Infinite loop</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>sunflow9</td>
<td>Null pointer exception</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>jbb2000</td>
<td>Corrupt output</td>
<td>Corrupt output</td>
<td>Correct</td>
</tr>
<tr>
<td>jbb2000</td>
<td>Infinite loop</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>sor</td>
<td>Infinite loop</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>lufact</td>
<td>Infinite loop</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>moldyn</td>
<td>Infinite loop</td>
<td>Correct</td>
<td>Correct</td>
</tr>
<tr>
<td>raytracer</td>
<td>Fails validation</td>
<td>Fails validation</td>
<td>Correct</td>
</tr>
</tbody>
</table>

EnfoRSer

Benchmarks: DaCapo 2006, 2009 and SPECjbb benchmarks
Intel Xeon with TSX. A 14-core processor

End-to-End Memory Models: Run-time cost vs Strength

- SC (Unbounded) Synchronization-free Region Serializability
- Dynamically Bounded Region Serializability via EnfoRSer
- This work: HTM-based dynamically bounded region serializability

DRF0 (C++/Java)
Outline

• Challenges
  • Naïve implementation with hardware transactional memory
  • Limitations of using HTM for DBRS

• Approach
  • Overcoming limitations
  • Our approach to DBRS enforcement: Legato

• Evaluation
Enforcing DBRS with Commodity Hardware Transactional Memory (HTM)
Enforcing DBRS with Commodity Hardware Transactional Memory (HTM)
Enforcing DBRS with Commodity Hardware Transactional Memory (HTM)
High cost of TSX instructions
Start/End ≈ three atomic operations!\(^2\)

Legato: Key Idea

Merge several regions into a single transaction: amortize the cost of starting and stopping a transaction
Legato: Key Idea
Legato: Key Idea

Transaction Start

Transaction Merge
Legato: Key Idea

Transaction Start

Transaction Merge

Transaction Merge
Legato: Key Idea
Challenges

**Conflicts** abort transactions: wasted work

**Capacity** aborts: larger transactions have **larger footprint**, unknown *a priori*

**HTM-unfriendly operations**: hardware interrupts, page faults etc.
Per-transaction cost vs Abort cost
Per-transaction cost vs Abort cost

Transient cause: conflicts, hardware interrupts
=>
Retry transaction

Capacity abort: end before culprit region
=>
Start new transaction
Per-transaction cost vs Abort cost
Per-transaction cost vs Abort cost

Transient cause: conflicts, hardware interrupts
=>
Retry transaction

Capacity abort: end before culprit region
=>
Start new transaction
Per-transaction cost vs Abort cost

Transient cause: conflicts, hardware interrupts
=>
Retry transaction

Capacity abort: end before culprit region
=>
Start new transaction
Per-transaction cost vs Abort cost

- Transient cause: conflicts, hardware interrupts
  - => Retry transaction
- Capacity abort: end before culprit region
  - => Start new transaction
Per-transaction cost vs Abort cost

Transient cause: conflicts, hardware interrupts => Retry transaction
Capacity abort: end before culprit region => Start new transaction
Per-transaction cost vs Abort cost

Transient cause: conflicts, hardware interrupts
=>
Retry transaction

Capacity abort: end before culprit region
=>
Start new transaction

Future transaction behavior unknown
Legato: Our Approach

Decide on a merge target: use history of previous transaction

1. Temporary target changes rapidly: capture transient effects

2. Setpoint or “steady state” target changes slowly: capture program phases
Merging Algorithm
Merging Algorithm: Initial Phase

Transaction Start

Curr Target: 8
Setpoint: 8
Merging Algorithm: Initial Phase

Transaction Merge

Curr Target: 8
Setpoint: 8
Merging Algorithm: Initial Phase

Curr Target: 8
Setpoint: 8
Merging Algorithm: Recover from Transient Error

Curr Target: 8/2 = 4
Setpoint: 8

Rapid action:
Merging Algorithm: Recover from Transient Error

Transaction Start

Curr Target: 4
Setpoint: 8
Merging Algorithm: Recover from Transient Error

Transaction Merge

Curr Target: 4
Setpoint: 8
Merging Algorithm: Recover from Transient Error

Curr Target: $4/2 = 2$
Setpoint: 8
Merging Algorithm: Recover from Transient Error

Curr Target: 2
Setpoint: 8

Transaction Start
Merging Algorithm: Recover from Transient Error

Transaction End

Curr Target: 2
Setpoint: 8

No Action: Skeptical
Merging Algorithm: Build up to Setpoint

Transaction End

Curr Target: 2*2 = 4
Setpoint: 8
Merging Algorithm: Build up to Setpoint

Transaction End

Curr Target: 4*2 = 8
Setpoint: 8

Aggressive
Merging Algorithm: Change of Program Phase

Curr Target: 9
Setpoint: $8 + I = 9$

Transaction End
Merging Algorithm: Change of Program Phase

Enter a low-conflict program phase
=> more aggressive merging

Curr Target: 9
Setpoint: 8 + 1 = 9

Transaction End
Merging Algorithm: Change of Program Phase

Transaction End

Curr Target: 10
Setpoint: 9 + 1 = 10
Implementation and Evaluation
Implementation

Developed in Jikes RVM 3.1.3
Code publicly available in Jikes RVM Research Archive
Run-time Performance
Run-time Performance

% Overhead over unmodified JVM

- EnfoRSer
- HTM: 1 DBR per-trans

- eclips6
- hsqldb6
- lusearch6
- xalan6
- arora9
- jython9
- lucene9
- lusearch9
- pmd9
- sunflow9
- xalan9
- pb2000
- pb2005
- geomean
Run-time Performance

% Overhead over unmodified JVM

<table>
<thead>
<tr>
<th>Tool</th>
<th>EnfoRSer</th>
<th>HTM: 1 DBR per-trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>eclips6</td>
<td>250</td>
<td>175</td>
</tr>
<tr>
<td>hsqldb6</td>
<td>275</td>
<td>175</td>
</tr>
<tr>
<td>lusearch6</td>
<td>225</td>
<td>175</td>
</tr>
<tr>
<td>xalan6</td>
<td>175</td>
<td>175</td>
</tr>
<tr>
<td>arrora9</td>
<td>150</td>
<td>175</td>
</tr>
<tr>
<td>jython9</td>
<td>150</td>
<td>175</td>
</tr>
<tr>
<td>lumdex9</td>
<td>100</td>
<td>175</td>
</tr>
<tr>
<td>lusearch9</td>
<td>50</td>
<td>175</td>
</tr>
<tr>
<td>pmid9</td>
<td>400</td>
<td>175</td>
</tr>
<tr>
<td>sunflow9</td>
<td>200</td>
<td>175</td>
</tr>
<tr>
<td>xalan9</td>
<td>40</td>
<td>175</td>
</tr>
<tr>
<td>pjb2000</td>
<td>150</td>
<td>40</td>
</tr>
<tr>
<td>pjb2005</td>
<td>150</td>
<td>40</td>
</tr>
<tr>
<td>Geomean</td>
<td>150</td>
<td>40</td>
</tr>
</tbody>
</table>
Run-time Performance

% Overhead over unmodified JVM

- EnfoRSer
- HTM: 1 DBR per-trans
- Legato

- eclipse6
- hsqldb6
- lusearch6
- xalan6
- arrora9
- jython9
- lundex9
- lusearch9
- pmid9
- sunflow9
- xalan9
- pbj2000
- pbj2005
- geomean

Overhead:
- Eclipse: 175%
- HSQLDB: 40%
- Xalan: 33%
Run-time Performance

% Overhead over unmodified JVM

- EnfoRSer
- HTM: 1 DBR per-trans
- Legato

More stable overhead
JIT Compilation Time
JIT Compilation Time

![Compilation Time Graph]

- **Base**
- **EnfoRSer**
- **Legato**

### Compilation Time

JIT Compilation Time

The image shows a bar chart comparing the JIT compilation time for different projects using three different tools: Base, EnfoRSer, and Legato. The compilation times are measured in units, with the x-axis listing the project names and the y-axis showing the compilation time. The chart indicates that EnfoRSer has the lowest compilation time for most projects, while Legato has the highest. The specific compilation times for each project are not shown in the text, but the chart highlights the relative performance of the tools. The overall average compilation time for all projects is shown at the bottom right of the chart, with EnfoRSer having a value of 2.38 and Legato having a value of 1.45.
JIT Compilation Time

![Compilation Time Graph]

- **Base**
- **EnfoRSer**
- **Legato**

Complex compiler transformations: 2.38
Simple instrumentation: 1.45
Related Work

• *Checks conflicts in bounded region*
  DRFx, Marino et al., PLDI 2010

• *Checks conflicts in synchronization-free regions*
  Conflict Exceptions, Lucia et al., ISCA 2010

• *Enforces atomicity of bounded regions*
  BulkCompiler, Ahn et al., MICRO 2009
  Atom-Aid, Lucia et al., ISCA 2008

• *Reducing dynamic misspeculations*
  BlockChop, Mars and Kumar, ISCA 2012
Memory Models: Run-time cost vs Strength