Toward Efficient Strong Memory Model Support for the Java Platform via Hybrid Synchronization

Aritra Sengupta,
Man Cao,
Michael D. Bond
and
Milind Kulkarni

PPPJ 2015, Melbourne, Florida, USA
Programming Language Semantics?

Data Races

Java provides weak semantics
Weak Semantics

\[
\begin{align*}
T1 & : \quad A & a = \text{null}; \\
& & \text{boolean init} = \text{false}; \\
& & a = \text{new A}(); \\
& & \text{init} = \text{true}; \\
T2 & : \quad \text{if (init)} \\
& & a.\text{field}++; 
\end{align*}
\]
Weak Semantics

T1

No data dependence

T2

a = new A();

init = true;

if (init)

a.field++;

No data dependence
Weak Semantics

\[ T1 \]
\[
A \ a = \text{null}; \\
\text{boolean init} = \text{false};
\]

\[ T2 \]
\[
a = \text{new A}(); \\
\text{if (init)} \\
a.\text{field}++; \\
\text{init} = \text{true};
\]
Weak Semantics

\[
\begin{align*}
\text{T1} & \quad \text{T2} \\
\text{init} & = \text{true;}
\end{align*}
\]

\[
\begin{align*}
\text{if (init)} \\
\text{a.field++;}
\end{align*}
\]

\[
\begin{align*}
a & = \text{new A();}
\end{align*}
\]
init = true;

a = new A();

if (init)
a.field++;
Java Memory Model

- JMM (Manson et al., POPL, 2005) variant of DRF0 (Adve and Hill, ISCA, 1990)
- Atomicity of synchronization-free regions for data-race-free programs
- Data races: weak semantics
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
Memory Models: Run-time cost vs Strength

1. Ouyang et al. ... and region serializability for all. In HotPar, 2013.
Memory Models: Run-time cost vs Strength

Statically Bounded Region Serializability (SBRS)

- Compiler demarcated regions execute atomically
- Execution is an interleaving of these regions

– Sengupta et al., ASPLOS, 2015
Statically Bounded Region Serializability (SBRS)

- `methodCall()`
- `acq(lock)`
- `rel(lock)`

Synchronization operations
Method calls
Loop backedges
Statically Bounded Region Serializability (SBRS)
Statically Bounded Region Serializability (SBRS)

- Statically and dynamically bounded
- Loop backedges
Overview

Enforcement of SBRS with dynamic locks
• Precise dynamic locks: EnfoRSer-D (our prior work), low contention

Enforcement of SBRS with static locks
• Imprecise static locks: EnfoRSer-S, low instrumentation overhead

Hybridization of locks
• EnfoRSer-H: static and dynamic locks, right locks for right sites?

Results
• EnfoRSer-H does at least as well as either. Some cases significant benefit
Enforcement of SBRS

Prevent two concurrent accesses to the same memory location where one is a write
Enforcement of SBRS

Prevent two concurrent accesses to the same memory location.

Prevent regions that have races from running concurrently!
Enforcement of SBRS

- Acquire locks before each memory access
- Acquire locks at the start of the region
Enforcement of SBRS

- Acquire locks before each memory access
- Acquire locks at the start of the region
- Precise object locks: dynamic locks
Enforcement of SBRS

Acquire locks before each memory access

Region level locks: statically chosen for an access site

Acquire locks at the start of the region
EnfoRSer-D

Precise dynamic locks

Per-access locks with retry mechanism

Compiler Transformations: Speculative execution
SBRS with Dynamic Locks

\[ Y = X \]

Dynamic per-access locks
SBRS with Dynamic Locks

Y = X
Z =

Program access
SBRS with Dynamic Locks

\[ Y = X = Z \]

Ownership transferred
SBRS with Dynamic Locks

Dynamic locks: precise location, hence no reasoning about races statically!

Ownership transferred
SBRS with Dynamic Locks

Precise conflict detection - efficient for high-conflicting regions

High instrumentation cost at each access

High overhead for low-conflicting programs

Ownership transferred

$Z = X$
Experimental Methodology

• Benchmarks
  • DaCapo 2006, 9.12-bach
  • Fixed-workload versions of SPECjbb2000 and SPECjbb2005

• Platform
  • Intel Xeon system: 32 cores
Implementation and Evaluation

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

%overhead over unmodified JVM

-5

hsqldb6 xalan6 avrora9 luindex9 lusearch6 lusearch9 sunflow9 xalan9 pjbb2000 pjbb2005 geomean

EnfoRSer-D

30
Run-time Performance

EnfoRSer-D

27% overhead on average
Run-time Performance

Precise conflict detection but high instrumentation overhead!

Complex compiler transformations (additional code)

Can we do better?
EnfoRSer with Static Locks

Reduce the instrumentation overhead of EnfoRSer-D

Less complex code generation
EnfoRSer-S

| Static region level locks | Racing sites acquire same lock | Coarsened locks to reduce instrumentation overhead |
SBRS with Locks on Static Sites

Static locks: racy sites protected by same lock

\[ Y = \]
\[ = X \]
\[ Z = \]
SBRS with Locks on Static Sites

\[ Y = Z = \]

All locks acquired before access
SBRS with Locks on Static Sites

\[ Y = Z = X \]

Ownership transferred
SBRS with Locks on Static Sites

Y = X
Z = L

Imprecise and does not lower instrumentation overhead!

Ownership transferred
SBRS with Locks on Static Sites

Y = L012

Z = X
SBRS with Locks on Static Sites

- Low instrumentation overhead
- Imprecise conflict detection

Coarsened single lock
EnfoRSer-S

R1                      R2                      R3                      R4

s0                      s2                      s5                      s7

s1                      s3                      s4                      s8

s4                      s3                      s5
EnfoRSer-S

R1                                R2                              R3                          R4

s0                                  s2                               s5                          s7
                                    s3                               s6                          s8
                                    s1
EnfoRSer-S

R1

S0
S1

R2

S2
S3
S4

R3

S5
S6

R4

S7
S8

RACE
EnfoRSer-S

Naik et al.’s 2006 race detection algorithm, Chord
EnfoRSer-S
EnfoRSer-S

R1

L0
L1

s0
s1

R2

L0
L1
L2

s2
s3
s4

R3

L1
L2

s5
s6

R4

L3
L4

s7
s8
EnfoRSer-S

Does not reduce instrumentation overhead!
EnfoRSer-S
EnfoRSer-S

R1                                  R2                               R3                         R4

s0                                  s2                                      s5
s1                                  s3                                      s6
s4

Same Region Static Locks (SRSL) ∪ RACE
EnfoRSer-S

Same Region Static Locks (SRSL) $\cup$ RACE
EnfoRSer-S
EnfoRSer-S

- Reduces instrumentation overhead
- Increases contention
Run-time Performance

% overhead over unmodified JVM

<table>
<thead>
<tr>
<th></th>
<th>EnfoRSer-D</th>
<th>EnfoRSer-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>hsqldb6</td>
<td>11,000</td>
<td>12,000</td>
</tr>
<tr>
<td>xalan6</td>
<td>3,500</td>
<td>3,400</td>
</tr>
<tr>
<td>avrora9</td>
<td>12,000</td>
<td>10,000</td>
</tr>
<tr>
<td>luindex9</td>
<td>33,000</td>
<td>4,300</td>
</tr>
<tr>
<td>lusearch6</td>
<td>10,000</td>
<td>3,400</td>
</tr>
<tr>
<td>lusearch9</td>
<td>660</td>
<td>660</td>
</tr>
<tr>
<td>sunflow9</td>
<td>2,600</td>
<td>2,600</td>
</tr>
<tr>
<td>xalan9</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>pjbb2000</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>pjbb2005</td>
<td>95</td>
<td>95</td>
</tr>
<tr>
<td>geomean</td>
<td>115</td>
<td>115</td>
</tr>
</tbody>
</table>

2600% overhead on average
Run-time Performance

EnfoRSer-S performs better.

2600% overhead on average.
Hybridizing Locks

High contention sites: Precise dynamic locks (precise conflict detection)

Low contention sites: Single static lock (low instrumentation overhead)
<table>
<thead>
<tr>
<th>EnfoRSer-H</th>
<th>Static locks to reduce instrumentation</th>
<th>Dynamic locks for precise conflict detection</th>
<th>Correctly and efficiently combine: best of both</th>
</tr>
</thead>
</table>

- Static locks to reduce instrumentation
- Dynamic locks for precise conflict detection
- Correctly and efficiently combine: best of both
Run-time Performance

26% overhead on average
Run-time Performance

Provides nearly the best of either approaches!

63% reduction in overhead
Run-time Performance

49% reduction in lock acquires and not increasing contention!

87% reduction in overhead

%overhead over unmodified JVM

hsqldb6 xalan6 avrora9 luindex9 lusearch6 lusearch9 sunflow9 xalan9 pjbb2000 pjbb2005 geomean
Run-time Performance

49% reduction in lock acquires

Dramatically improves on both when hybridization helps!

87% reduction in overhead
Hybridizing Locks

Right synchronization for right program sites

Combining different synchronization mechanisms

Best of different synchronization mechanisms
EnfoRSer-H

Cost-benefit model

- Right locks for right sites
- Assignment algorithm

Combine static and dynamic locks

Profiling
Two-iteration Methodology

- **Program execution**
  - First iteration: Profiling (use RACE relation)
  - Between iterations: Compute SRSL and estimate cost
  - Second iteration: Use hybridized instrumentation

- **Assignment Algorithm**
Assignment Algorithm
Compute initial cost

Change a static lock to dynamic lock

Change all its racing sites to dynamic locks

Recompute cost

current cost < previous cost

Retain state

No

Revert to previous state

Yes

Profiled data
Assignment Algorithm

estimatedCost = \sum_{i=0}^{N} estimateCost(R_i);
Assignment Algorithm

estimatedCost = \sum_{i}^{N} \text{estimateCost}(R_i);

1. Conflicts on each site
2. Lock acquires on each site
Assignment Algorithm

R1
- s0
- s1
- s9

R2
- s2
- s3

R3
- s4
- s5
- s6

R4
- s7
- s8
Assignment Algorithm

R1

s0

s1

s9

R2

s2

s3

R3

s4

s5

s6

R4

s7

s8
If (current cost < previous cost)  
// retain state
Assignment Algorithm

R1

s0

s1

s9

R2

s2

s3

R3

s4

s5

s6

R4

s7

s8

Switch on next site
Assignment Algorithm

R1
- s0
- s1
- s9

R2
- s2
- s3

R3
- s4
- s5
- s6

R4
- s7
- s8
Assignment Algorithm

If (current cost < previous cost)  
// retain state
Assignment Algorithm

Switch on next site
Assignment Algorithm

estimatedCost = \sum_{i}^{N} estimateCost(R_i);

current cost > previous cost  // revert state
Assignment Algorithm

R1

R2

R3

R4

s0

s2

s4

s7

s1

s3

s5

s8

s9

s6
Assignment Algorithm

R1

s0
s1
s9

R2

s2
s3

R3

s4
s5
s6

R4

s7
s8

s1, s9
Assignment Algorithm

R1

s0

s1

s9

R2

s2

s3

R3

s4

s5

s6

R4

s7

s8

Final State!
Lock Assignment

R1

s0
s1
s9

R2

s2
s3

R3

s4
s5

R4

s7
s8

L1
Lock Assignment

R1

L1

s0
s1
s9

R2

L1

s2
s3

R3

L1

s4
s5
s6

R4

s7
s8

L1
Lock Assignment

R1: L1
   - s0
   - s1
   - s9

R2: L1
   - s2
   - s3
   - s5

R3: L1
   - s4
   - s6

R4: L1
   - s7
   - s8
Lock Assignment

R1

L1

s0

s1

s9

R2

L1

s2

s3

R3

L1

s4

s5

s6

R4

L1

s7

s8
Lock Assignment

Per-object locks and transformations
Related Work

• *Use of static locks*
  Chimera, Lee et al., PLDI 2012

• *Use of static analysis*
  • Goldilocks, Elmas et al., PLDI 2007.
  • Red Card, Flanagan and Freund, ECOOP 2013

• *Hybridizing locks*
  Hybrid Tracking, Cao et al., WODET 2014
## Conclusion

| Combine synchronization mechanisms | • Static locks  
|                                 | • Dynamic locks  
| Best of different kinds of synchronization | • Precise conflict detection  
|                                 | • Low instrumentation overhead  
| Efficient enforcement of SBRS | • Select the best for a region  