Valor: Efficient, Software-Only Region Conflict Exceptions

Swarnendu Biswas, Minjia Zhang, and Michael D. Bond
Ohio State University
Brandon Lucia
Carnegie Mellon University

OOPSLA 2015
A Simple C++ Program

```cpp
X* x = NULL;
bool done = false;

Thread T1
x = new X();
done = true;

Thread T2
if (done) {
    x->func();
}
```
A Simple C++ Program

```cpp
X* x = NULL;
bool done = false;
Thread T1
Thread T2
```
Data Races

```cpp
X* x = NULL;
bool done = false;

Thread T1:

x = new X();
done = true;

if (done) {
    x->func();
}

Thread T2:

Data race
```
Data Races are Evil

- **No semantic guarantees**: Lack of semantic guarantees make software unsafe
- **Complicates language specifications**: Challenging to reason about correctness for racy executions
- **Indicates other concurrency errors**: Leads to atomicity, order or sequential consistency violations
C++ treats data races as errors

```
X* x = NULL;
bool done = false;
```

Thread T1

```
x = new X();
done = true;
```

Thread T2

```
if (done) {
    x->func();
}
```
Catch-Fire Semantics

C++ treats data races as errors

x = new X();
done = true;
Catch-Fire Semantics

C++ treats data races as errors

\[ x = \text{new } X(); \]
\[ \text{done} = \text{true}; \]
A Java Example

Thread T1

```java
X = new Object();
done = true;
```

Thread T2

```java
while (!done) {}
X.compute();
```
A Java Example

Java tries to assign semantics, which are unsatisfactory

Thread T1

X = new Object();
done = true;

while (!done) {}  
X.compute();

Thread T2
C++ and Java Memory Models
C++ and Java Memory Models

- Data-race-free execution
- Strong semantics
- Execution is sequentially consistent
C++ and Java Memory Models

Data-race-free execution $\rightarrow$ Strong semantics

Execution is sequentially consistent

Synchronization-free regions execute atomically
C++ and Java Memory Models

- Data-race-free execution
- Strong semantics
- Execution is sequentially consistent
- Synchronization-free regions execute atomically

lock(l)
lock(m)
unlock(m)
unlock(l)
C++ and Java Memory Models

But what about data races?
C++ and Java Memory Models

But what about data races?
“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.”
Need for Stronger Memory Models

Adve and Boehm, CACM 2010

“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.”

“We call upon software and hardware communities to develop languages and systems that enforce data-race-freedom, ...”
Outline

- Programming language memory models and data races
- Data race and region conflict exceptions model
- Valor: Our contribution
- Evaluation
Data Race

Thread T1

X = new Object();
done = true;

Thread T2

while (!done) {}
X.compute();
Data Race Exceptions

Thread T1

X = new Object();
done = true;

Thread T2

while (!done) {}
X.compute();

EXCEPTION
REGION CONFLICT EXCEPTIONS MODEL
Region Conflict

Thread T1

Thread T2
Region Conflict

Thread T1

Thread T2

wr x
Region Conflict

Thread T1

wr x

Thread T2

rd/wr x
Region Conflict

Thread T1

wr x

Thread T2

rd/wr x

Conflict
Region Conflict

Thread T1

Thread T2

wr x

rd/wr x

Conflict

Reports a subset of true data races that violate region serializability
Execution Models

Reports a subset of true data races that violate region serializability sequentially consistent.

- data-race-free
- region-conflict-free
- region serializable
- sequentially consistent
Develop a practical region conflict detection technique
Region Conflict Detection

Hardware customizations required for good performance

- Limited by resources and applicability
  - Needs extensive modifications and is unscalable\(^1\)
  - Detects serializability violations of bounded regions\(^2\)

Outline

- Programming language memory models and data races
- Data race and region conflict exceptions model
- **Valor: Our contribution**
- Evaluation
Valor: Efficient, Software-Only Region Conflict Detector

- Elides tracking last readers, only tracks last writer
- Detect read-write conflicts lazily
Valor: Efficient, Software-Only Region Conflict Detector

- Tracking last writers
- Detecting read-write conflicts lazily
- Impact of lazy conflict detection
Tracking Last Writer
Per-Variable Metadata

Has an ongoing region updated x?

Epoch

Thread T1

wr x
Tracking Last Writer

Thread T1

Thread T2

x

p@T0
Tracking Last Writer

Thread T1

j

wr x

Thread T2

x

p@T0
Tracking Last Writer

Track last writer

Thread T1

Thread T2

j

wr x

x

p@T0
Tracking Last Writer

Track last writer

Thread T1

Thread T2

update metadata

wr x

j

j@T1
Tracking Last Writer

Track last writer

Thread T1

Thread T2

j

wr x

j@T1

rd/wr x
Tracking Last Writer

Track last writer

Thread T1
- j
- wr x

Thread T2
- rd/wr x

Conflict

j @ T1
Tracking Last Writer

- Track last writer
  - Allows precisely detecting write-write and write-read conflicts

Thread T1
- j
- wr x

Thread T2
- rd/wr x

Conflict

j@T1
Valor: Efficient, Software-Only Region Conflict Detector

- Tracking last writers
- Detecting read-write conflicts lazily
- Impact of lazy conflict detection
Detecting Read-Write Conflicts

Thread T1

rd x

Thread T2

wr x
Detecting Read-Write Conflicts
Detecting Read-Write Conflicts

Thread T1

Thread T2

update read metadata

rd x

j

j@T1
Detecting Read-Write Conflicts

Thread T1

Thread T2

j

rd x

wr x

j@T1
Detecting Read-Write Conflicts

Thread T1:
- \( j \)
  - \( \text{rd } x \)

Thread T2:
- \( \text{wr } x \)
  - Conflict

\( j_@T1 \)
Detecting Read-Write Conflicts

This simple mechanism used in prior work has problems
Remote Cache Misses Due to Tracking of Metadata

Thread T1

Write operation

update metadata

rd x

Leads to remote cache misses
Metadata Updates

Thread T1

\[ j \]

\[ \text{rd/wr } x \]
Metadata Updates
Synchronization on Metadata Updates

Thread T1

\[ j \]

- lock l
- rd/wr x
Synchronization on Metadata Updates

Thread T1

lock l
update metadata
rd/wr x
Synchronization on Metadata Updates

Thread T1

- j
- lock l
- update metadata
- rd/wr x
- unlock l
Synchronization on Metadata Updates

Thread T1

- lock l
- update metadata
- rd/wr x
- unlock l

Bad for mostly read-only data
Elide Tracking Last Readers
Elide Tracking Last Readers

Thread T1
- rd x

Thread T2
- wr x

?
Elide Tracking Last Readers

Detect read-write conflicts lazily

Thread T1
- rd x

Thread T2
- wr x
Elide Tracking Last Readers

Detect read-write conflicts lazily

- Log read accesses in thread-local buffers
- Validate reads at region boundaries

Thread T1

Thread T2

Per-Variable Metadata

Has an ongoing region updated x?

Thread T1

\(<v, j@T1>\)
Elide Tracking Last Readers

Detect read-write conflicts lazily

- Log read accesses in thread-local buffers
- Validate reads at region boundaries
Elide Tracking Last Readers

- Detect read-write conflicts lazily
  - Log read accesses in thread-local buffers
  - Validate reads at region boundaries

Thread T1

- \( j \)
- \( \log \langle x, v \rangle \)
- \( \text{rd } x \)

Thread T2

\( <v, p@T0> \)
Elide Tracking Last Readers

Detect read-write conflicts lazily

Thread T1

Thread T2
Elide Tracking Last Readers

Detect read-write conflicts lazily

Thread T1

log <x, v>

rd x

Thread T2

wr x

<x, v> → <v+1, k@T2>
Elide Tracking Last Readers

Detect read-write conflicts lazily

Thread T1
- log <x, v>
- rd x
- read validation

Thread T2
- wr x

<x> → <v+1, k@T2>
Elide Tracking Last Readers

Detect read-write conflicts lazily

T1 is not the last writer

Version read is outdated

Thread T1

j

log <x, v>

rd x

read validation

Thread T2

k

wr x

<v+1, k@T2>
Elide Tracking Last Readers

Detect read-write conflicts lazily

T1 is not the last writer

Version read is outdated

Thread T1

log \langle x, v \rangle

rd x

read validation

Conflict

Thread T2

wr x

\langle v+1, k@T2 \rangle
Elide Tracking Last Readers

Avoids

- Remote cache misses
- Synchronization overhead
Valor: Efficient, Software-Only Region Conflict Detector

- Tracking last writers
- Detecting read-write conflicts lazily
- Impact of lazy conflict detection
Precise Conflict Detection

Thread T1

rd x

Thread T2

wr x

Conflict
Precise vs Lazy Conflict Detection

Thread T1
- rd x

Conflict
- wr x

Thread T2
- wr x

Conflict validation
- delayed exception
Precise vs Lazy Conflict Detection

Thread T1

Thread T2

Thread T1

Thread T2

rd x

Delayed exceptions

delayed exception

Conflict

read validation
Delayed Exceptions

Do not compromise semantic guarantees

Effects should not be externally visible

Conflict validation

Thread T1

Thread T2

rd x

wr x

Thread T1

Thread T2

wr x

rd x

Conflict
Delayed Exceptions

Debugging might be slightly harder

Exception will be thrown at the next region boundary from the reader thread

But does not compromise on soundness and precision
Outline

- Programming language memory models and data races
- Data race and region conflict exceptions model
- Valor: Our contribution
- Evaluation
IMPLEMENTATION
Implementation

Developed on top of Jikes RVM 3.1.3
Implementation

Developed on top of Jikes RVM 3.1.3

Implemented FastTrack, state-of-art happens-before analysis based data race detector

Implementation

Developed on top of Jikes RVM 3.1.3

Implemented FastTrack, state-of-art happens-before analysis based data race detector

Shared on Jikes RVM Research Archive and ACM DL
EVALUATION
Experimental Methodology

• **Benchmarks**
  ▫ Large workload sizes of DaCapo 2006 and 9.12-bach suite
  ▫ Fixed-workload versions of SPECjbb2000 and SPECjbb2005

• **Platform**
  ▫ 64-core AMD Opteron 6272
Performance Comparison

Overheads (%) Over Unmodified Jikes RVM

- FastTrack
- Valor
Performance Comparison

Overheads (%) Over Unmodified Jikes RVM

- FastTrack
- Eager conflict detection
- Valor
Performance Comparison

Overheads (%) Over Unmodified Jikes RVM

First software-only region conflict detector with less than 100% overhead
Performance Comparison: Intel Xeon

Overheads (%) Over Unmodified Jikes RVM

Relative performance remains comparable on an Intel Xeon architecture.
Additional Experiments

Characterization of FastTrack and Jikes RVM

Data race coverage

Space overheads

Please check the paper
Valor: Contributions

Strong execution guarantees in software

Detects all violations of region serializability

Exception-free execution → Strong semantics

Advances state-of-art
- Provides strong semantics in software at less than 100% overhead
New Opportunities with Valor

**Semantic guarantees** Language runtimes could integrate this

**Debugging** Can be used to detect problematic data races

**Conflict exceptions** All-the-time monitoring in certain environments

**Aggressive optimizations** Reorder and eliminate redundant loads and stores within synchronization-free regions
Valor: Efficient, Software-Only Region Conflict Exceptions

Swarndeu Biswas, Minjia Zhang, and Michael D. Bond
Ohio State University
Brandon Lucia
Carnegie Mellon University

OOPSLA 2015