Parallelizing Data Race Detection

Benjamin Wester
Facebook

David Devecsery, Peter Chen, Jason Flinn, Satish Narayanasamy
University of Michigan
Data Races

\[ t_1 \]
- lock(\( l \))
- \( x = 1 \)
- unlock(\( l \))
- \( x = 3 \)

\[ t_2 \]
- lock(\( l \))
- \( x = 2 \)
- unlock(\( l \))
Race Detection

Detection is slow!
8x–200x
• Managed vs. binary
• Granularity
• Completeness
• Debug info gathered
Matches app concurrency
Race Detection

Normal run time

Parallel race detector

With race detector

Detection is slow!
8x–200x
• Managed vs. binary
• Granularity
• Completeness
• Debug info gathered

Matches app concurrency
How to use parallel hardware?

• Scale program?
  – Performance tied to app

• Parallel algorithm?
  – Lots of fine-grained dependencies

• Uniparallelism
  – Converts execution into parallel pipeline
  – Works for instrumentation

[Wallace CGO’ 07, Nightingale ASPLOS’ 08]
Uniparallelism
Uniparallelism

Epoch-sequential Execution

Epoch-parallel Execution
Uniparallelism

- **Scales well:** Epochs are independent execution units
- **More efficient:** Synchronization

Benjamin Wester - ASPLOS '13
Add Detector...

- **Scales well:** Epochs are independent execution units
- **More efficient:** Synchronization & Lock elision

Race detection depends on state!

Benjamin Wester - ASPLOS '13
Architecture

Epoch-sequential  Epoch-parallel  Commit

Benjamin Wester - ASPLOS '13
Moving Work

Identify meaningful **fast subset** of analysis state
- Low frequency, lightweight, self-contained
- Run in Epoch-Sequential phase
- Predicted and *usable in parallel phase*

**Symbolic** execution
- Replace missing state with symbols
- Defer some computation to commit phase
- Commit: replace symbols with concrete values
**Architecture**

- **Epoch-sequential**
  - Instrument and predict fast subset of analysis state

- **Epoch-parallel**
  - Perform deferred work on concrete values

- **Commit**
  - Full instrumentation
  - Symbolic execution

---

Benjamin Wester - ASPLOS '13
Parallel FastTrack

State: Vector clocks – e.g. \( \langle 0, 1, 0 \rangle \) – for each:

<table>
<thead>
<tr>
<th></th>
<th>Fast</th>
<th>Slow</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Thread</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Lock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>– Variable x 2: last read(s), last write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example computation:

\[
\text{Check}(\text{read}_x \leq \text{thread}_i); \ \text{write}_x = \text{thread}_i
\]

Transitive reduction

Use knowledge of happens-before relationship
Parallel Eraser

State:

- Set of locks held by thread (lockset)  
  Fast
- Lockset for each variable  
  Slow
- Position in state machine for each variable

Example computation:

If \( \text{state}_x \equiv \text{SHARED} \) then \( \text{ls}_x = \text{ls}_x \cap \{L, M\} \)

Lockset factorization

Factor out common behavior
Parallel Performance

Efficiency
Scalability

2 worker threads as baseline
8-CPU Platform

Benchmarks:
- SPLASH-2 (water, lu, ocean, fft, radix)
- Parallel app: pbzip2
Efficiency
Exactly 2 cores

- Sequential (base)
- Parallel HB
- Parallel LS
Efficiency
Exactly 2 cores

Median 8% faster
Median 13% faster
Scaling Parallel FastTrack

2 Worker Threads

- water
- lu
- ocean
- fft
- pbzip2
- radix

Number of Cores

Speedup

Benjamin Wester - ASPLOS '13
Scaling Parallel FastTrack

2 Worker Threads

Median 4.4x with 8 cores

Benjamin Wester - ASPLOS '13
Scaling Parallel Eraser

2 Worker Threads

Median 3.3x with 8 cores
Conclusion

- 3-Phase Uniparallel architecture
  - Parallel FastTrack
  - Parallel Eraser

- Parallel algorithms:
  - Are more efficient
  - Scale better
Questions?