SWORD: A Bounded Memory-Overhead Detector of OpenMP Data Races in Production Runs

Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric
School of Computing, University of Utah, Salt Lake City, UT 84112

Presented at IPDPS 2018
See paper for details

Ignacio Laguna, Greg L. Lee, Dong H. Ahn
Lawrence Livermore National Laboratory, Livermore, CA

Github.com / PRUNERS
What is a data race?
What is a data race?
What is a data race?

Thread 1

Thread 2

W

R/W
What is a data race?

Thread 1  Thread 2

No synchronizations

W  R/W
One way to eliminate this race
One way to eliminate this race
One way to eliminate this race
Another way to eliminate this race
Another way to eliminate this race

Signal using `special’ variables

• Java ‘volatile’ annotations
  • NOT C ‘volatiles’ 😞

• C++11 ’atomic’ annotations
A third way
A third way

Put a barrier
Why eliminate races?
Popular answer: "avoid nondeterminism"
Unclear what “nondeterminism” means..
Execution Order is Still Nondeterministic

\[ T_0 \quad X = 0 \quad UNLOCK \quad T_1 \quad t = X \quad LOCK \quad UNLOCK \]
More relevant: Avoid “pink elephants” 😊
More relevant: Avoid “pink elephants” 😊

Pink elephant (Sutter): “A value you never wrote but managed to read”

Aka “out of thin air” value
The birth of a **pink elephant**...

Compiler Optimizations

You may never have written "24" in your program
Details of how a **pink elephant** is made!

The compiler has NO IDEA that the user meant to communicate here!!

Compiler optimizations create these pink-elephant values...

\[
\begin{align*}
X &= 0 \\
Y &= 23 \\
X &= Y + 1 \\
\end{align*}
\]

\[
\begin{align*}
t &= X \\
\end{align*}
\]

\[
\begin{align*}
X &= 24 \\
Y &= 23 \\
t &= X \\
\end{align*}
\]
This is why code containing data races often fail (only) when optimized!
Race-freedom ensures intended communications

- You don’t observe “half baked” values
- Code does not reorder around sync. points
- No “word tearing”
- Pending writes flushed (fences inserted)
Exploding a myth!

There is no such thing as a benign race!!

What could possibly go wrong?
Races in OpenMP programs are hard to spot

• See tinyurl.com/ompRaces if you wish
  • but later 😊

• Static analysis tools never shown to work well

• First usable OpenMP dynamic race checker (afaik)
  • Archer [Atzeni, IPDPS’16]
  • More on that soon

• **This talk** will present the second usable dynamic race checker
  • Sword
This talk: Why and how of another OMP race checker
The Pink Elephant Actually Struck Us!

- HYDRA porting on Sequoia at LLNL
  - Large multiphysics MPI/OpenMP application
  - Non-deterministic crashes in OpenMP region
    - Only when the code was optimized!
  - Suspected data race
  - Emergency hack:
    - Disabled OpenMP in Hypre
Archer to the rescue!
Archer [IPDPS’16]
• Utah: Simone Atzeni, Ganesh Gopalakrishnan, Zvonimir Rakamaric
• LLNL: Dong H. Ahn, Ignacio Laguna, Martin Schulz, Gregory L. Lee
• RWTH: Joachim Protze, Matthias S. Muller

- In production use at LLNL

Part of the “PRUNERS” tool suite

PRUNERS was a finalist of the 2017 R&D 100 Award Selection
Archer’s “find”

Two threads writing 0 to the same location without synchronization
Archer’s “find”

Two threads writing 0 to the same location without synchronization
Did we live “happily ever after?”
No 😞
Archer has "memory-outs"; also misses races
Archer has “memory-outs”; also misses races

- Archer increases memory 500%
- It also misses races!

- These were known issues
  - Finally surfaced with the “right large example”
A programmable number of cells per address (4 shown, and is typical)
A programmable number of cells per address (4 shown, and is typical)

~4 shadow cells per application location

Shadow-cells immediately increase memory demand by a factor of four
Archer misses races due to shadow cell eviction
Archer misses races due to shadow cell eviction

```c
int a[N];

#pragma omp parallel for
for(int i = 0; i < N; i++) {
    a[i] = a[i] + a[3];
}
```
Archer misses races due to shadow cell eviction

```c
int a[N];

#pragma omp parallel for
for(int i = 0; i < N; i++) {
    a[i] = a[i] + a[3];
}
```

All threads read A[3]
Thread 3 writes a[3]
Capacity conflict $\rightarrow$ evict shadow cell

With shadow-cell evicted, races are missed
Archer misses races due to HB-masking
Archer misses races due to HB-masking

These are concurrent; there are two races here!

(a) No happens-before (race detected)

(b) Happens-before (no race detected)

These races are missed in this interleaving!
Solution : Get rid of shadow cells !!
Need New Approach with Online/Offline split

Offline Analysis

Race Reports
Details of the online phase

- Collect traces per core **un-coordinated**
  - Trace collection speeds increased; we use the OMPT tracing method

- Employ data compression to bring FULL traces out
  - Only 2.5 MB compression buffer per thread (**fits in L3 cache**).
Consequences for the offline phase

- We would have lost all the synchronization information
- We only know what each thread is doing

- We must recover the concurrency structure
- And in the context of its happens-before order, detect races!
Offline synchronization recovery and analysis

OpSem (HIPS’18)

IBarrier(3)
Barrier(1)
read(x)
write(y)
m_acq()
m_rel()
IBarrier(4)
Barrier(2)
write(y)
m_acq(M1)
m_rel(M1)
write(x)
m_acq()
m_rel()
IBarrier(6)
FOR-LOOP
IBarrier(7)
R1: race on y
R2: race on y
R3: race on x
Core 0
Core 1
Core 2
Core 3
Compression
Compression
Compression
Compression
OpSem (HIPS’18)
Core 0
Core 1
Core 2
Core 3
Compression
Compression
Compression
Compression
OpSem (HIPS’18)
Offset-Span Labels: How we record concurrency

(Mellor-Crummey, 1991)
Key state in OpSem: Maintain Barrier Intervals
Examples of Races Reported

- Race on y
- Race on x
- Race within the same barrier interval
Examples of Races Reported

R1: race on y
R2: race on y
R3: race on x

Races across parallel regions

Barrier Interval 3

Barrier Interval 2

Barrier Interval 5
Good news

• Online analysis proved really good
  • No memory pressure !!
Offline analysis *took a day to finish on “medium sized” examples*
Two Key Innovations Saved the Approach

• Self-balancing red-black interval trees

• On-the-fly generation of Integer Linear Programs
Reducing “a day” to “under a minute”

- Decompress, record strided accesses in self-balancing red-black interval trees
- Generate Integer Linear Programs on-the-fly, and check for overlaps
  - Handles bursts of accesses efficiently
OMP read/writes are bursty with strides!
OMP read/writes are bursty with strides!

Build Integer Linear Programs for each constant-stride interval
ILP system encodes accessed byte-addresses in each “burst”

Each of this is a multi-word access
Overlap of Access Bursts: ILP Generation!

\[
T_0 : \quad 8 \cdot x_0 + 10 + s_0 = a \\
\quad \land 0 \leq x_0 \leq 4 \\
\quad \land 0 \leq s_0 < 4
\]

\[
T_1 : \quad 8 \cdot x_1 + 14 + s_1 = a \\
\quad \land 0 \leq x_1 \leq 4 \\
\quad \land 0 \leq s_1 < 4
\]
Interval Trees to record accesses

- Recorded info is: [Begin, End], #Accesses, Kind, Stride, AtWhichPCValue
- Allows efficient comparison of access bursts across threads
- These Red-Black trees are highly tuned
  - Used within Linux to realize fair scheduling methods
Concluding Remarks: Sword is now practical!

Both Archer and Sword are available

Github.com / PRUNERS
Conclusions: Time for “Medium” Examples

<table>
<thead>
<tr>
<th></th>
<th>Online</th>
<th>Offline</th>
<th>Total</th>
<th>Efficacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archer</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Misses races</td>
</tr>
<tr>
<td>Sword</td>
<td>1</td>
<td>10*</td>
<td>11</td>
<td>Finds all races within the execution**</td>
</tr>
</tbody>
</table>

* : can be brought down to 1 by using an MPI cluster
** : we define the formal semantics of OMP race checking [HIPS’18]
# Conclusions: Time for Larger Examples

<table>
<thead>
<tr>
<th></th>
<th>Online</th>
<th>Offline</th>
<th>Total</th>
<th>Efficacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archer</td>
<td>Memory</td>
<td>0</td>
<td>1</td>
<td>Misses races</td>
</tr>
<tr>
<td>Sword</td>
<td>1</td>
<td>10*</td>
<td>11</td>
<td>Finds all races within the execution**</td>
</tr>
</tbody>
</table>

---

**Note:**
- Memory indicates the Archer misses races.
- * indicates a significant result.
- ** indicates a special note or condition.
More Concluding Remarks

• Sword works well; finds more races than Archer
  • Applied to realistic benchmarks
    • Archer test suite
    • RaceBench from LLNL
  • Offline analysis can be parallelized
  • Still “decent” on standard multicore platforms

• It took many ideas working together to realize Sword
  • Formal semantics of OpenMP Concurrency
  • Online / Offline checking split
  • Data compression
  • Self-balancing interval trees
  • ILP-systems to compress traces
  • Employs standard tracing methods based on OMPT
Future Work

- Continue to debug / tune Sword
- Incorporate ideas from upcoming pubs
- GPU race checking
Group Credits

Simone    Zvonimir    Dong    Ignacio    Greg
Extras
Data Races: Gist

• High-level code is just “fiction”
  • Code optimizations are done on a PER THREAD basis
  • Races occur if you don’t tell a compiler what’s shared

while(!f) {} \rightarrow r = f; while (!r) {} : this is OK if “f” is purely local

while(!f) {} \rightarrow r = f; while (!r) {} : not OK if f is shared and you don’t tell this to the compiler

• How to inform a compiler
  • Put the variables inside a mutex (or other synchronization block)
  • Declare them to be a Java volatile or C++11 atomic
    • C-volatiles won’t do (they don’t have a definite concurrency semantics)
Data Races: Gist

- High-level code is just “fiction”
  - Code optimizations are done on a PER THREAD basis
  - Races occur if you don’t tell a compiler what’s shared

```c
while(!f) {} → r = f; while (!r) {} : this is OK if “f” is purely local
```

```c
while(!f) {} → r = f; while (!r) {} : not OK if f is shared and you don’t tell this to the compiler
```

Position paper: Nondeterminism is unavoidable, but data races are pure evil

Hans-J. Boehm
HP Laboratories
Hans.Boehm@hp.com
GPUs races also can lead to “pink-elephants”

Initially : \( x[i] == y[i] == i \)

Warp-size = 32

```c
__global__ void kernel(int* x, int* y)
{
    int index = threadIdx.x;
    y[index] = x[index] + y[index];

    if (index != 63 && index != 31)
        y[index+1] = 1111;
}
```

The hardware schedules these instructions in “warps” (SIMD groups).

However, this “warp view” often appears to be lost

E.g. When compiling with optimizations

Expected Answer: 0, 1111, 1111, ..., 1111, 64, 1111, ...

New Answer: 0, 2, 4, 6, 8, ...