Accelerating Dynamic Data Race Detection Using Static Thread Interference Analysis

Peng Di and Yulei Sui

School of Computer Science and Engineering
The University of New South Wales
2052 Sydney Australia

March 12-13, 2016
Outline

- Motivation
- Analysis phases
- Evaluation
Dynamic Data Race Detector: ThreadSanitizer

ThreadSanitizer (TSan) finds data races in multithreaded programs by inserting instrumentations at compile-time to perform runtime checks for all memory accesses.

```c
void foo(int *p) {
    *p = 42;
}
```

*Original C code*

```c
void foo(int *p) {
    __tsan_func_entry(__builtin_return_address(0));
    __tsan_write4(p);
    *p = 42;
    __tsan_func_exit();
}
```

*Code after instrumentation*
TSan performance slowdown over native code for SPLASH2 benchmarks (under compiler option -O0)

Machine: Ubuntu Linux 3.11.0-15-generic Intel Xeon Quad Core HT, 3.7GHz, 64GB
Outline

- Motivation
- Analysis phases
- Evaluation
Framework

Source

Clang Front-End

bc file

Analyses, Optimizations

Memory Pairs Collection

Memory pairs

Call Graph Construction

Reachability Optimization

Reachable pairs

Interleaving Analysis

Interleaving Optimization

MHP pairs

Pointer Analysis

Alias Optimization

Aliased pairs

Thread-Local Analysis

Thread-Local Optimization

Escaped pairs

Lockset Analysis

Lockset Optimization

Unlocked pairs

Guided Instrumentation

Instrumented bc file

Code Generation

Binary

PMAM 2016 co-located with PPoPP 2016
Framework

Source

Clang Front-End

bc file

Analyses, Optimizations

Memory Pairs Collection

Memory pairs

Call Graph Construction

Reachability Optimization

Reachable pairs

Interleaving Analysis

Interleaving Optimization

MHP pairs

Pointer Analysis

Alias Optimization

Aliased pairs

Thread-Local Analysis

Thread-Local Optimization

Escaped pairs

Lockset Analysis

Lockset Optimization

Unlocked pairs

Guided Instrumentation

Instrumented bc file

Code Generation

Binary

PMAM 2016 co-located with PPoPP 2016
Refining Pairs

Only statements in the pairs that are not filtered are instrumented for runtime check.
Context-Sensitive Abstract Threads

An abstract thread $t$ refers to a call of `pthread_create()` at a context-sensitive fork site during the analysis.

```c
void main(){
    for(i=0;i<10;i++){
       fork(t[i], foo)
    }
}

void foo(){
    cs1: foo();
    cs2: foo();
}
```

$t_1$ refers to fork site under context [1,3] $t_1'$ refers to fork site under context [2,3]

t and $t_1'$ are context-sensitive threads
Context-Sensitive Abstract Threads

An abstract thread \( t \) refers to a call of `pthread_create()` at a context-sensitive fork site during the analysis.

```c
void main()
{
    for (i=0; i<10; i++) {
        fork(t[i], foo)
    }
}

void foo()
{
    cs1: foo();
    cs2: foo();
}
```

\( t_1 \) refers to fork site under context \([1,3]\)

\( t_1' \) refers to fork site under context \([2,3]\)

\( t_1 \) and \( t_1' \) are context-sensitive threads

A thread \( t \) always refers to a context-sensitive fork site, i.e., a unique runtime thread unless \( t \in \mathcal{M} \) is multi-forked, in which case, \( t \) may represent more than one runtime thread.
Outline

• Motivation
• Analysis phases
  • Thread Interleaving Analysis
  • Alias Analysis
  • Thread Local Analysis
  • Lock Analysis
• Evaluation
Context-sensitive Thread Interleaving Analysis

\[(t_1, c_1, s_1) \parallel (t_2, c_2, s_2) \text{ holds if:} \]
\[
\begin{cases} 
 t_2 \in \mathcal{I}(t_1, c_1, s_1) \land t_1 \in \mathcal{I}(t_2, c_2, s_2) & \text{if } t_1 \neq t_2 \\
 t_1 \in \mathcal{M} & \text{otherwise}
\end{cases}
\]

where \(\mathcal{I}(t, c, s)\): denotes a set of interleaved threads may run in parallel with \(s\) in thread \(t\) under calling context \(c\), \(\mathcal{M}\) is the set of multi-forked threads.
Interleaving Analysis

Computing $\mathcal{I}(t, c, s)$ is formalized as a forward data-flow problem $(V, \sqcap, F)$.

- $V$: the set of all thread interleaving facts.
- $\sqcap$: meet operator ($\sqcup$).
- $F$: $V \rightarrow V$ transfer functions associated with each node in an ICFG.
Interleaving Analysis Rule

\begin{align*}
\text{[I-DESCENDANT]} & \quad t \xrightarrow{(c, fk_i)} t' \quad (t, c, fk_i) \rightarrow (t, c, \ell) \quad (c', \ell') = \text{Entry}(S_{t'}) \\
& \quad \{t'\} \subseteq \mathcal{I}(t, c, \ell) \quad \{t\} \subseteq \mathcal{I}(t', c', \ell') \\
\text{[I-SIBLING]} & \quad t \Join t' \quad (c, \ell) = \text{Entry}(S_t) \quad (c', \ell') = \text{Entry}(S_{t'}) \quad t \neq t' \land t' \neq t \\
& \quad \{t\} \subseteq \mathcal{I}(t', c', \ell') \quad \{t'\} \subseteq \mathcal{I}(t, c, \ell) \\
\text{[I-JOIN]} & \quad t \xleftarrow{(c, jn_i)} t' \\
& \quad I(t, c, jn_i) = I(t, c, jn_i) \setminus \{t'\} \\
\text{[I-CALL]} & \quad (t, c, \ell) \xrightarrow{\text{call}_i} (t, c', \ell') \quad c' = c.\text{push}(i) \\
& \quad I(t, c, \ell) \subseteq I(t, c', \ell') \\
\text{[I-INTRA]} & \quad (t, c, \ell) \rightarrow (t, c, \ell') \\
& \quad I(t, c, \ell) \subseteq I(t, c, \ell') \\
\text{[I-RET]} & \quad (t, c, \ell) \xrightarrow{\text{ret}_i} (t, c', \ell') \quad i = c.\text{peek()} \quad c' = c.\text{pop()} \\
& \quad I(t, c, \ell) \subseteq I(t, c', \ell')
\end{align*}
Interleaving Analysis Rule

\[ I-DESCENDANT \]

\[ t \xrightarrow{(c, f_{k_i})} t' \quad (t, c, f_{k_i}) \to (t, c, \ell) \quad (c', \ell') = Entry(S_{t'}) \]

\[ \{t'\} \subseteq I(t, c, \ell) \quad \{t\} \subseteq I(t', c', \ell') \]

\[ t \xleftarrow{\text{fork}} t' \]

\[ I(t, c, s) = \{t'\} \quad I(t', c', s') = \{t\} \]

\[ (t, c, s) \parallel (t', c', s') \]
Interleaving Analysis Rule

\[
\begin{align*}
\text{[I-JOIN]} & \quad t \xleftarrow{(c,jn_i)} t' \\
\mathcal{I}(t, c, jn_i) & = \mathcal{I}(t, c, jn_i) \setminus \{t'\}
\end{align*}
\]

\[\text{fork} \quad I(t, c, s) = \{t'\}\]

\[\text{join} \quad I(t', c', s') = \{t\}\]

\[I(t,c,s1) = \{\}\]

\[(t,c,s) \parallel (t,c,s) \quad (t',c',s') \not\!
\parallel (t,c,s1)\]
Interleaving Analysis Rule

[I-SIBLING] \[ t \bowtie t' \quad (c, \ell) = \text{Entry}(S_t) \quad (c', \ell') = \text{Entry}(S_{t'}) \quad t \not\succ t' \land t' \not\succ t \]

\[
\{t\} \subseteq I(t', c', \ell') \quad \{t'\} \subseteq I(t, c, \ell)
\]

\[
(t, c, s) \lor (t', c', s')
\]

\[
I(t, c, s) = \{\}
\]

\[
I(t', c', s') = \{\}
\]

\[
(t, c, s) \lor (t', c', s')
\]
Interleaving Analysis Rule

\[ [I\text{-SIBLING}] \quad t \Join t' \quad (c, \ell) = \text{Entry}(S_t) \quad (c', \ell') = \text{Entry}(S_{t'}) \quad t \not\succ t' \land t' \not\succ t \]

\[ \{t\} \subseteq I(t', c', \ell') \quad \{t'\} \subseteq I(t, c, \ell) \]

![Diagram of Interleaving Analysis Rule]

\( I(t,c,s) = \{t'\} \)

\( I(t',c',s') = \{t\} \)

\( (t,c,s) \parallel (t',c',s') \)
Interleaving Analysis Rule

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
</table>
| **[I-DESCENDANT]** | $t \xrightarrow{(c,fk_i)} t' \quad (t, c, fk_i) \rightarrow (t, c, \ell) \quad (c', \ell') = \text{Entry}(S_{t'})$
|          | $\{t'\} \subseteq I(t, c, \ell) \quad \{t\} \subseteq I(t', c', \ell')$ |
| **[I-SIBLING]** | $t \bowtie t' \quad (c, \ell) = \text{Entry}(S_t) \quad (c', \ell') = \text{Entry}(S_{t'})$  \quad $t \neq t' \land t' \neq t$
|          | $\{t\} \subseteq I(t', c', \ell') \quad \{t'\} \subseteq I(t, c, \ell)$ |
| **[I-JOIN]** | $t \xleftarrow{(c,jn_i)} t' \quad \mathcal{I}(t, c, jn_i) = \mathcal{I}(t, c, jn_i) \setminus \{t'\}$ |
| **[I-CALL]** | $t \xrightarrow{(t, c, \ell)} (t, c', \ell') \quad c' = \text{c.push}(i)$ |
|          | $\mathcal{I}(t, c, \ell) \subseteq \mathcal{I}(t, c', \ell')$ |
| **[I-INTRA]** | $t \xrightarrow{(t, c, \ell)} (t, c', \ell') \quad \mathcal{I}(t, c, \ell) \subseteq \mathcal{I}(t, c', \ell')$ |
| **[I-RET]** | $t \xrightarrow{(t, c, \ell)} (t, c', \ell') \quad i = \text{c.peak()} \quad c' = \text{c.pop()}$ |
|          | $\mathcal{I}(t, c, \ell) \subseteq \mathcal{I}(t, c', \ell')$ |
Outline

• Motivation
• Analysis phases
  • Thread Interleaving Analysis
  • Alias Analysis
  • Thread Local Analysis
  • Lock Analysis
• Evaluation
Alias Analysis

Obtain aliasing pairs by refining MHP store-load and store-store pairs \([(t, c, s), (t', c', s')]\), where \(\text{Alias}(\ast p, \ast q)\) is the set of objects pointed to by both \(p\) and \(q\).

\[
\begin{align*}
\text{s : } \ast p &= \ast q \quad \text{or } \ast q &= \ast p \\
\text{(t, c, s) } \parallel \text{(t', c', s')} &\quad o \in \text{Alias}(\ast p, \ast q) \\
\text{s } &\xrightarrow{\circ} \text{ s'}
\end{align*}
\]

```
int x, y;
int *p, *q, *r;
p=&x;
q=&x;
r=&y;

void main(){
    fork(t, foo);
    s1: *p=...;
    s2: *r=...;
}

void foo(){
    s3: ...=*q;
}
```
Outline

- Motivation
- Analysis phases
  - Thread Interleaving Analysis
  - Alias Analysis
  - Thread Local Analysis
  - Lock Analysis
- Evaluation
Thread-Local Analysis

An object is not *thread-local*, i.e. *escaping*, if it escapes via

- arguments at a fork site
- global pointers

void main()
{
    int x,y;
    fork(t,foo,&x);
    s1:  x=...;
        join(t);
    s3:  y=...;
}

void foo(int* p){
    s2:  ...=*p;
}
Outline

• Motivation
• Analysis phases
  • Thread Interleaving Analysis
  • Alias Analysis
  • Thread Local Analysis
  • Lock Analysis
• Evaluation
Lockset Analysis

Statements from different lock-unlock spans, are interference-free if these spans are protected by a common lock. Our framework does this by performing a flow- and context-sensitive analysis for lock/unlock operations.

```
int x;
mutex m;
void main(){
    fork(t, foo);
    s1: x=...;
    lock(m);
    s3: x=...;
    unlock(m);
    join(t);
}

void foo(){
    lock(m);
    s2: ...=x;
    unlock(m);
}
```
Outline

• Motivation
• Analysis phases
• Evaluation
Evaluation

- Implementation:
  - On top of our previous open-source tool SVF (http://unsw-corg.github.io/SVF/)
  - Based on our previous papers CGO ’16 and ICPP ’15

- Benchmarks:
  - 11 SPLASH2 Pthread benchmarks

- Machine setup:
  - Ubuntu Linux 3.11.0-15-generic Intel Xeon Quad Core HT, 3.7GHZ, 64GB
## Instrumentation Statistics (under Option -O0)

<table>
<thead>
<tr>
<th></th>
<th>Pthread API</th>
<th>TSan</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fork</td>
<td>Join</td>
<td>Lock</td>
</tr>
<tr>
<td>barnes</td>
<td>1</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>fft</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>lu_cb</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>lu_ncb</td>
<td>1</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>ocean_cp</td>
<td>1</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>ocean_ncp</td>
<td>1</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>radiosity</td>
<td>3</td>
<td>3</td>
<td>38</td>
</tr>
<tr>
<td>radix</td>
<td>1</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>raytrace</td>
<td>1</td>
<td>1</td>
<td>13</td>
</tr>
<tr>
<td>water_nsquared</td>
<td>1</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>water.spatial</td>
<td>1</td>
<td>1</td>
<td>19</td>
</tr>
</tbody>
</table>
Speedups over Original TSan (under Option -O0)

![Graph showing speedups over Original TSan for various benchmarks. The graph compares speedups for 4 Threads and 16 Threads for benchmarks such as barnes, fft, lu_cb, lu_ncb, ocean_cp, ocean_ncp, radiosity, radix, raytrace, water_nsquared, water_spatial, and average.]
Thanks!

Q & A