Validating Optimizations of Concurrent C/C++ Programs

Soham Chakraborty

IIT Delhi

SAT + SMT, 2019
int X = 0; int Y = 0;

Y = 4; \text{ if } (X)
X = 1; \quad r = Y;
int X = 0; int Y = 0;

Y = 4;  // if (X)
X = 1;   // r = Y;

Race on $X \leadsto$ undefined semantics
$X == 1 \land r \neq 4$ is possible
(i.e., the program is wrong)
atomic_int X = 0;  int Y = 0;

Y = 4;
atomic_store(&X, 1, mo_release);

if (atomic_load(&X, mo_acquire))
    r = Y;
basic syntax:

```
atomic_int X = 0; int Y = 0;

Y = 4;
atomic_store(&X, 1, mo_release);
```

```
if (atomic_load(&X, mo_acquire))
    r = Y;
```
Concurrent Programming in C11

```
atomic_int X = 0; int Y = 0;

Y = 4;
atomic_store(&X, 1, mo_release);
if (atomic_load(&X, mo_acquire))
    r = Y;
```
atomic_int X = 0;  int Y = 0;

Y = 4;

atomic_store(&X, 1,
    mo_release);

if (atomic_load(&X,
    mo_acquire))
    r = Y;
atomic_int X = 0; int Y = 0;

Y = 4;
atomic_store(&X, 1, mo_release);

if (atomic_load(&X, mo_acquire))
    r = Y;
atomic_int X = 0; int Y = 0;

Y = 4;
atomic_store(&X, 1, mo_release);

if (atomic_load(&X, mo_acquire))
    r = Y;
Concurrent Programming in C11

```c
atomic_int X = 0;  int Y = 0;

Y = 4;
atomic_store(&X, 1, mo_release);
if (atomic_load(&X, mo_acquire))
   r = Y;
```
Concurrent Programming in C11

atomic_int X = 0; int Y = 0;

\[ Y = 4; \]
atomic_store(&X, 1, mo_release);

\[ \text{if (atomic_load(&X, mo_acquire)) r = Y; } \]

\[ \downarrow \]

\[ X = Y = 0; \]
\[ Y = 4; \]
\[ X_{\text{rel}} = 1; \]
\[ \text{if (X_{\text{acq})} r = Y; } \]
An Unsafe Reordering

\[
X = Y = 0; \\
Y = 4; \\
X_{\text{rel}} = 1; \\
\text{if}(X_{\text{acq}}) \\
r = Y;
\]

Always returns \( r == 4 \)

\[
X = Y = 0; \\
r = 4; \\
\text{if}(X_{\text{acq}}) \\
r = Y;
\]

Optimizations for sequential programs are NOT always safe for concurrent programs.

\[
X = Y = 0; \\
X_{\text{rel}} = 1; \\
Y = 4; \\
\text{if}(X_{\text{acq}}) \\
r = Y;
\]

May return \( r == 0 \)
An Unsafe Reordering

\[
X = Y = 0; \quad Y = 4; \quad X_{rel} = 1; \quad r = 4; \quad \text{if}(X_{acq}) \quad r = Y;
\]

Always returns \( r == 4 \)

\[
X = Y = 0; \quad X_{rel} = 1; \quad r = 4; \quad \text{if}(X_{acq}) \quad Y = 4; \quad r = Y;
\]

May return \( r == 0 \)

Optimizations for sequential programs are **NOT** always safe for concurrent programs.
Questions

Q1: Which of the transformations are allowed?

Q2: Does a compiler perform only allowed transformations?
Q1: Which of the transformations are allowed?

C11 $\sim$ C11 [POPL’15]

- access($X$); access($Y$); $\sim$ access($Y$); access($X$);
- access($X$); access($X$); $\sim$ access($X$);
Reordering Transformations

Reordering \((a;b \sim b;a)\)

<table>
<thead>
<tr>
<th>↓ a (\setminus) b (\rightarrow)</th>
<th>(R_{na})</th>
<th>(W_{na})</th>
<th>(R_{acq})</th>
<th>(W_{rel})</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_{na})</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>…</td>
</tr>
<tr>
<td>(W_{na})</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>…</td>
</tr>
<tr>
<td>(R_{acq})</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>…</td>
</tr>
<tr>
<td>(W_{rel})</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>…</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

\[t = X; s = Y_{acq}; \sim \rightarrow s = Y_{acq}; t = X; \quad ✓\]

\[s = Y_{acq}; t = X; \sim \rightarrow t = X; s = Y_{acq}; \quad \times\]
Read-after-Read (RAR)
\[ t = X; t' = X; \sim \rightarrow t = X; t' = t; \quad \checkmark \]

Read-after-Read (RAW)
\[ X = 1; t = X; \sim \rightarrow X = 1; t = 1; \quad \checkmark \]

Over-written Writes (OW)
\[ X = 1; X = 2; \sim \rightarrow X = 2; \quad \checkmark \]
Questions

Q1: Which of the transformations are allowed?

- C11 ↼ C11 [POPL’15]

Q2: Does a compiler perform only allowed transformations?

- C11 compilation by LLVM [CGO’16]
Another Example

\[
\begin{align*}
X &= Y = 0; \\
Y &= 4; \\
X_{\text{rel}} &= 1; \\
f &= \text{false}; \\
a &= f \ ? \ Y : 0; \\
b &= X_{\text{acq}} \ ? \ Y : 4;
\end{align*}
\]
Another Example

\[
\begin{align*}
X &= Y = 0; \\
Y &= 4; \\
X_{\text{rel}} &= 1; \\
f &= \text{false}; \\
\cdots \\
a &= f \ ? \ Y : 0; \\
b &= X_{\text{acq}} \ ? \ Y : 4;
\end{align*}
\]

Output: \( b == 4 \) always
\[ X = Y = 0; \]
\[ f = \text{false}; \]
\[ \ldots \]
\[ a = f \ ? \ Y : 0; \]
\[ b = X_{\text{acq}} \ ? \ Y : 4; \]

\[
\text{Context:} \begin{bmatrix}
    X = Y = 0; \\
    f = \text{false}; \\
    s = Y; \\
    a = f \ ? \ s : 0; \\
    t = X_{\text{acq}}; \\
    b = t \ ? \ s : 4; \\
\end{bmatrix}
\]

\[ \text{Output } b == 0 \text{ possible in target.} \]
LLVM Compilation Bug in More Detail

\[
X = Y = 0; \\
f = false; \\
\ldots \\
a = f \ ? \ Y : 0; \\
b = X_{\text{acq}} \ ? \ Y : 4;
\]

\[
X = Y = 0; \\
f = false; \\
\ldots \\
s = Y; \\
a = f \ ? \ s : 0; \\
t = X_{\text{acq}}; \\
r = Y; \\
b = t \ ? \ r : 4;
\]

\[
X = Y = 0; \\
f = false; \\
\ldots \\
s = Y; \\
a = f \ ? \ s : 0; \\
t = X_{\text{acq}}; \\
r = Y; \\
b = t \ ? \ s : 4;
\]
\begin{align*}
X &= Y = 0; \\
f &= \text{false}; \\
\ldots \\
a &= f \ ? \ Y : 0; \\
b &= X_{\text{acq}} \ ? \ Y : 4;
\end{align*}

\begin{align*}
X &= Y = 0; \\
f &= \text{false}; \\
\ldots \\
s &= Y; \\
a &= f \ ? \ s : 0; \\
t &= X_{\text{acq}}; \\
r &= Y; \\
b &= t \ ? \ r : 4;
\end{align*}

\begin{align*}
X &= Y = 0; \\
f &= \text{false}; \\
\ldots \\
s &= Y; \\
a &= f \ ? \ s : 0; \\
t &= X_{\text{acq}}; \\
r &= Y; \\
b &= t \ ? \ s : 4;
\end{align*}
\[ X = Y = 0; \]
\[ f = false; \]
\[ \ldots \]
\[ a = f \ ? \ Y : 0; \]
\[ b = X_{acq} \ ? \ Y : 4; \]

\[ X = Y = 0; \]
\[ f = false; \]
\[ \ldots \]
\[ s = Y; \]
\[ a = f \ ? \ s : 0; \]
\[ t = X_{acq}; \]
\[ r = Y; \]
\[ b = t \ ? \ r : 4; \]

C11: (1) Error
\[
X = Y = 0; \\
f = false; \\
\ldots \\
a = f \ ? \ Y : 0; \\
b = X_{acq} \ ? \ Y : 4;
\]

\[
X = Y = 0; \\
f = false; \\
\ldots \\
s = Y; \\
a = f \ ? \ s : 0; \\
t = X_{acq}; \\
r = Y; \\
b = t \ ? \ r : 4;
\]

\[
X = Y = 0; \\
f = false; \\
\ldots \\
s = Y; \\
a = f \ ? \ s : 0; \\
t = X_{acq}; \\
r = Y; \\
b = t \ ? \ s : 4;
\]

C11: (1) Error (2) Correct
\[
X = Y = 0; \\
f = false; \\
\ldots \\
a = f \ ? \ Y : 0; \\
b = X_{\text{acq}} \ ? \ Y : 4;
\]

\[
X = Y = 0; \\
f = false; \\
\ldots \\
s = Y; \\
a = f \ ? \ s : 0; \\
t = X_{\text{acq}}; \\
r = Y; \\
b = t \ ? \ r : 4;
\]

C11: (1) Error (2) Correct

LLVM: (1) Correct
\[ X = Y = 0; \]
\[ f = false; \]
\[ \ldots \]
\[ a = f \ ? \ Y : 0; \]
\[ b = X_{acq} \ ? \ Y : 4; \]

\[ X = Y = 0; \]
\[ f = false; \]
\[ \ldots \]
\[ (1) \ s = Y; \]
\[ a = f \ ? \ s : 0; \]
\[ t = X_{acq}; \]
\[ r = Y; \]
\[ b = t \ ? \ r : 4; \]

\[ (2) \ s = Y; \]
\[ a = f \ ? \ s : 0; \]
\[ t = X_{acq}; \]
\[ r = Y; \]
\[ b = t \ ? \ s : 4; \]

C11: (1) Error (2) Correct

LLVM: (1) Correct (2) Error
Our Approach: LLVM Validation

\[ P_{src} \xrightarrow{LLVM} P_{tgt} \ ? \ Correct : \ Potential \ Error \]

\[ \Downarrow \]

\[ P_{src} \xrightarrow{(R \cup E)^*} P_{tgt} \ ? \ Correct : \ Potential \ Error \]

w.r.t. safe reorderings (R) & eliminations (E):
- For the LLVM model
- For the C11 model
Exposed concurrency compilation bugs in LLVM 3.6

- Reported and fixed in LLVM 3.7
Validation Schemes

Compiler Independent Matching (CIM)
- Can be used in validating other compilers.

Metadata Based Matching (MD)
- LLVM specific, uses metadata in LLVM.
Compiler Independent Matching (CIM)

- Can be used in validating other compilers.

Steps:

- Identify corresponding program paths
- Compute deletability of accesses
- Match access sequences and analyze
Example: Compiler Independent Matching

\[
\begin{align*}
  s_1 &= X \\
  s_2 &= X \\
  V &= 1 \\
  s_4 &= Z_{acq} \\
  Y &= 1 \\
  Y &= 2 
\end{align*}
\]
Example: Compiler Independent Matching

\[
\begin{align*}
\checkmark & \quad s_1 = X \\
&s_2 = X \\
&V = 1 \\
&s_4 = Z_{\text{acq}} \\
&Y = 1 \\
&Y = 2
\end{align*}
\]
Example: Compiler Independent Matching

\[ \checkmark \quad s_1 = X \]

\[ \times \quad s_2 = X \]

\[ V = 1 \]

\[ s_4 = Z_{acq} \]

\[ Y = 1 \]

\[ Y = 2 \]
Example: Compiler Independent Matching

✓ $s_1 = X$

✗ $s_2 = X$

$V = 1$

✓ $s_4 = Z_{\text{acq}}$

$Y = 1$

$Y = 2$
Example: Compiler Independent Matching

- $s_1 = X$
- $s_2 = X$ (Wrong)
- $V = 1$
- $s_4 = Z_{acq}$
- $Y = 1$
- $Y = 2$ (Correct)

Check that unmatched accesses are deletable
Check that reorderings are allowed
Example: Compiler Independent Matching

\[ s_1 = X \]
\[ s_2 = X \]
\[ V = 1 \]
\[ s_4 = Z_{acq} \]
\[ Y = 1 \]
\[ Y = 2 \]
Example: Compiler Independent Matching

 ✓ $s_1 = X$
✗ $s_2 = X$
 ✓ $V = 1$
 ✓ $s_4 = Z_{acq}$
✗ $Y = 1$
 ✓ $Y = 2$
Example: Compiler Independent Matching

\[ \checkmark \ s_1 = X \]
\[ \times \ s_2 = X \]
\[ \checkmark \ V = 1 \]
\[ \checkmark \ s_4 = Z_{acq} \]
\[ \times \ Y = 1 \]
\[ \checkmark \ Y = 2 \]

\[ t_1 = X \]
\[ t_2 = Z_{acq} \]
\[ Y = 2 \]
\[ V = 1 \]
Example: Compiler Independent Matching

- $s_1 = X$
- $s_2 = X$
- $V = 1$
- $s_4 = Z_{acq}$
- $Y = 1$
- $Y = 2$

$t_1 = X$
$t_2 = Z_{acq}$
$Y = 2$
$V = 1$
Example: Compiler Independent Matching

\[ \begin{align*}
\checkmark \quad s_1 &= X \\
\times \quad s_2 &= X \\
\checkmark \quad V &= 1 \\
\checkmark \quad s_4 &= Z_{\text{acq}} \\
\times \quad Y &= 1 \\
\checkmark \quad Y &= 2
\end{align*} \]

\[ \begin{align*}
t_1 &= X \\
t_2 &= Z_{\text{acq}} \\
Y &= 2 \\
V &= 1
\end{align*} \]
Example: Compiler Independent Matching

\[
\begin{align*}
\checkmark \quad s_1 &= X \\
\times \quad s_2 &= X & t_1 &= X \\
\checkmark \quad V &= 1 \\
\checkmark \quad s_4 &= Z_{\text{acq}} \\
\times \quad Y &= 1 & t_2 &= Z_{\text{acq}} \\
\checkmark \quad Y &= 2 & Y &= 2 \\
\checkmark \quad Y &= 2 & V &= 1
\end{align*}
\]
Example: Compiler Independent Matching

\[
\begin{align*}
\checkmark \quad s_1 &= X \\
\times \quad s_2 &= X \\
\checkmark \quad V &= 1 \\
\checkmark \quad s_4 &= Z_{acq} \\
\times \quad Y &= 1 \\
\checkmark \quad Y &= 2 \\
\checkmark \quad t_1 &= X \\
\checkmark \quad t_2 &= Z_{acq} \\
\checkmark \quad Y &= 2 \\
\checkmark \quad V &= 1
\end{align*}
\]
Example: Compiler Independent Matching

\[ \begin{align*}
\checkmark s_1 &= X \\
\times s_2 &= X \\
\checkmark V &= 1 \\
\checkmark s_4 &= Z_{\text{acq}} \\
\times Y &= 1 \\
\checkmark Y &= 2
\end{align*} \]

- Check that unmatched accesses are deletable
- Check that reorderings are allowed
Example: Compiler Independent Matching

- \( s_1 = X \)
- \( s_2 = X \)
- \( V = 1 \)
- \( s_4 = Z_{acq} \)
- \( Y = 1 \)
- \( Y = 2 \)

- \( t_1 = X \)
- \( t_2 = Z_{acq} \)
- \( Y = 2 \)
- \( V = 1 \)

- Check that unmatched accesses are deletable
- Check that reorderings are allowed
Example: Compiler Independent Matching

- $s_1 = X$
- $s_2 = X$
- $V = 1$
- $s_4 = Z_{acq}$
- $Y = 1$
- $Y = 2$

$t_1 = X$
$t_2 = Z_{acq}$
$Y = 2$
$V = 1$

Correct

- Check that unmatched accesses are deletable
- Check that reorderings are allowed
Program with Control Flow

\{ABCDEF, ABCEF\} \implies\ ABCF, \ \{ACDEF, ACEF\} \implies\ AGCF

- Use branching conditions to match the paths
  - using Z3 SMT solver

- Match access sequences for each path pair
- Unroll loops a fixed number of times
- Use branching conditions to match the paths
  - using Z3 SMT solver
- Match access sequences for each path pair
Compiler optimizations require careful analysis

Reported LLVM concurrency compilation bugs; all were fixed.

Validator: http://plv.mpi-sws.org/validc/
Integrate with validator for sequential programs.
Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Thank you!
Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Handle other language features (e.g. array, pointer)
Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Handle other language features (e.g. array, pointer)

use SAT/SMT solvers
Future Work

Integrate with validator for sequential programs.

Handle loops effectively.

Handle other language features (e.g. array, pointer)

use SAT/SMT solvers

Thank you!