Automatic Synthesis of Weakest Preconditions for Compiler Optimizations

Nuno Lopes
Advisor: José Monteiro
Expectations for Compilers

- Improve performance
- Reduce code size
- Reduce energy consumption
• LLVM 3.2 introduced a Loop Vectorizer
• Performance improvement of 10-300% in benchmarks
But Compilers are Full of Bugs

- Yang, Chen, Eide, Regehr [PLDI’12]:
  - 79 bugs in GCC (25 P1)
  - 202 bugs in LLVM
  - 2 wrong-code bugs in CompCert

- Le, Afshari, Su [PLDI’14]:
  - 40 wrong-code bugs in GCC
  - 42 wrong-code bugs in LLVM

- Last week:
  - 395 open wrong-code bug reports in GCC
  - 14 open wrong-code bug reports in LLVM
Churn in Compiler’s code

- +0.5M LoC added to LLVM last year
- 20k commits
- Over 4M LoC in LLVM
Compilers by Dragon’s Lenses

Automatic Synthesis of Weakest Preconditions for Compiler Optimizations
Loop Unswitching

\[
\begin{align*}
\text{while} & \ I < N \ \text{do} \\
\text{if} & \ B \ \text{then} \\
& \ S_1 \\
\text{else} & \\
& \ S_2 \\
& \ I := I + 1
\end{align*}
\]

\[
\begin{align*}
\text{if} & \ B \ \text{then} \\
\text{while} & \ I < N \ \text{do} \\
& \ S_1 \\
& \ I := I + 1 \\
\text{else} & \\
\text{while} & \ I < N \ \text{do} \\
& \ S_2 \\
& \ I := I + 1
\end{align*}
\]

\(S_1, S_2\) are template statements

\(B\) is a template Boolean expression
Specifying Compiler Optimizations

- Transformation function
- Precondition
- Profitability heuristic
Contributions

• Automatic weakest precondition synthesis for compiler optimizations
• Automatic partial equivalence checking, applied to compiler optimization verification
Deriving preconditions by hand is hard; WPs are often non-trivial

WPs derived by hand are often wrong!

Weaker preconditions expose more optimization opportunities
For a logical right shift, we can fold if the comparison is not signed. We can also fold a signed comparison if the shifted mask value and the shifted comparison value are not negative. These constraints are not obvious, but we can prove that they are correct using an SMT solver such as "Z3":

http://rise4fun.com/Z3/Tslfh

```cpp
if (ShiftOpcode == Instruction::AShr) {
  // There may be some constraints that make this possible, but nothing simple has been discovered yet.
  CanFold = false;
}
```
Loop Unswitching

while I < N do
  if B then
    S_1
    I := I + 1
  else
    S_2
    I := I + 1

  if B then
    while I < N do
      S_1
      I := I + 1
  else
    while I < N do
      S_2
      I := I + 1
Loop Unswitching: Example Instantiation

... while I < N do
  if \( B \) then
    \( S_1 \) := \( A + N \)
  else
    \( S_2 \) := \( A + 1 \)
  I := I + 1
...

**Instantiation:**

\( B \mapsto N > 5 \)

\( S_1 \mapsto A := A + N \)

\( S_2 \mapsto A := A + 1 \)

... if \( N > 5 \) then
  while I < N do
    A := A + N
    I := I + 1
  else
    while I < N do
      A := A + 1
      I := I + 1

Loop Unswitching: Weakest Precondition

\[
\begin{align*}
\text{while } & I < N \text{ do } \\
\text{if } & B \text{ then } \\
& S_1 \\
\text{else } & \\
& S_2 \\
& I := I + 1
\end{align*}
\]

\[
\begin{align*}
\text{if } & B \text{ then } \\
& \begin{align*}
& \text{while } I < N \text{ do } \\
& & S_1 \\
& & I := I + 1
\end{align*}
\end{align*}
\]

\[
\begin{align*}
\text{else } & \\
& \begin{align*}
& \text{while } I < N \text{ do } \\
& & S_2 \\
& & I := I + 1
\end{align*}
\end{align*}
\]

Precondition:
\[
\begin{align*}
& I \notin R(B) \land \\
& W(S_1) \cap R(B) = \emptyset \land \\
& W(S_2) \cap R(B) = \emptyset
\end{align*}
\]
• Read and Write sets for each template statement/expression
• Arbitrary quantifier-free constraints over read/write sets
• In practice constraints are only over R/W and W/W intersection
  \[ v \notin R(B) \]
  \[ W(S_1) \cap R(B) = \emptyset \]
  \[ W(S_1) \cap W(S_2) = \emptyset \]
• Books and developers already informally speak about read and write sets
• Similar to PEC’s
• Can be efficiently discharged using current compiler technology:
  – Memory dependence analysis
  – Alias/pointer analysis
  – Loop analysis
  – Range analysis
  – …
Synthesizing WP for Loop Unswitching

\[
\text{while } I < N \text{ do} \\
\text{if } B \text{ then} \\
S_1 \\
\text{else} \\
S_2 \\
I := I + 1
\]

\[
\text{if } B \text{ then} \\
\text{while } I < N \text{ do} \\
S_1 \\
I := I + 1 \\
\text{else} \\
\text{while } I < N \text{ do} \\
S_2 \\
I := I + 1
\]
1) Find counterexample

\[
\text{while } I < N \text{ do } \\
\quad \text{if } B \text{ then } S_1 \\
\quad \text{else } S_2 \\
\quad I := I + 1 \\
\]
2) Synthesize WP for counterexample: VC Gen

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I &lt; N$</td>
<td>$I := I + 1$</td>
</tr>
<tr>
<td>$I &lt; N$</td>
<td>$I := I + 1$</td>
</tr>
<tr>
<td>$\neg B$</td>
<td>$I := I + 1$</td>
</tr>
<tr>
<td>$I \geq N$</td>
<td>$I \geq N$</td>
</tr>
</tbody>
</table>

$I_0 < N_0 \land B_0 \land$

$I_1 = \text{ite}(wS_1I, \ S_1I0, \ I_0) \land$

$N_1 = \text{ite}(wS_1N, \ S_1N0, \ N_0) \land$

$I_2 = I_1 + 1 \land$

$I_2 < N_1 \land$

$\neg B_1 \land$

$I_3 = \text{ite}(wS_2I, \ S_2I0, \ I_2) \land$

$N_2 = \text{ite}(wS_2N, \ S_2N0, \ N_1) \land$

$I_4 = I_3 + 1 \land$

$I_4 \geq N_2$
2) Synthesize WP for counterexample: Conditional Ackermannization

\[
\begin{align*}
I_0 &< N_0 \land \\
B_0 &\land \\
I_1 & = \text{ite}(wS_1I, S_1I0, I_0) \land \\
N_1 & = \text{ite}(wS_1N, S_1N0, N_0) \land \\
I_2 & = I_1 + 1 \land \\
I_2 &< N_1 \land \\
\neg B_1 &\land \\
I_3 & = \text{ite}(wS_2I, S_2I0, I_2) \land \\
N_2 & = \text{ite}(wS_2N, S_2N0, N_1) \land \\
I_4 & = I_3 + 1 \land \\
I_4 &\geq N_2
\end{align*}
\]

B_0 and B_1 are equal if the values of the variables in R(B) are equal:

\[
\left( (I \in R(B) \rightarrow I_0 = I_2) \land \right) \\
\left( (N \in R(B) \rightarrow N_0 = N_1) \rightarrow B_0 = B_1 \right)
\]
2) Synthesize WP for counterexample: Final constraint

\[ \exists S \forall V \text{ Path} \land \text{Ackermann} \land \text{MustWrite} \land ... \rightarrow \text{PathIsCorrect} \]

\[ S = \text{Read/Write sets} \]
\[ V = \text{Vars from VCGen, Must-write vars} \]

A possible model:
\[ W(S_1) = \emptyset \]
\[ R(S_1) = \emptyset \]
\[ R(B) = \emptyset \]
2) Synthesize WP for counterexample: Disjunction of all models

\begin{align*}
I &< N \\
B \\
S_1 \\
I &:= I + 1 \\
I &< N \\
\neg B \\
S_2 \\
I &:= I + 1 \\
I &\ge N
\end{align*}

**Precondition:**
$I \notin R(B) \land W(S_1) \cap R(B) = \emptyset$
3) Iterate until no more counterexamples can be found

\[
\text{while } I < N \text{ do }
\begin{align*}
\text{if } B & \text{ then } \\
S_1 & \\
\text{else} & \\
S_2 & I := I + 1
\end{align*}
\]

\[
\text{if } B \text{ then } \\
\text{while } I < N \text{ do }
S_1
I := I + 1
\]

\[
\text{else } \\
\text{while } I < N \text{ do }
S_2
I := I + 1
\]

Precondition:
\[
I \notin R(B) \land \\
W(S_1) \cap R(B) = \emptyset \land \\
W(S_2) \cap R(B) = \emptyset
\]
1) Find counterexample
2) Generate WP that rules out the counterexample
3) Iterate until no more counterexamples can be found
Optimizations

• Model generalization
• Exploit UNSAT cores
• Bias towards R/W and W/W intersections
### PSyCO: Results

<table>
<thead>
<tr>
<th>Optimization</th>
<th># Counterexamples</th>
<th># Models</th>
<th>WP Time</th>
<th>Total Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code hoisting</td>
<td>1</td>
<td>1</td>
<td>0.07s</td>
<td>0.23s</td>
</tr>
<tr>
<td>Constant propagation</td>
<td>1</td>
<td>1</td>
<td>0.04s</td>
<td>0.16s</td>
</tr>
<tr>
<td>Copy propagation</td>
<td>0</td>
<td>0</td>
<td>0s</td>
<td>0.11s</td>
</tr>
<tr>
<td>If-conversion</td>
<td>0</td>
<td>0</td>
<td>0s</td>
<td>0.11s</td>
</tr>
<tr>
<td>Partial redundancy elimin.</td>
<td>1</td>
<td>1</td>
<td>0.10s</td>
<td>0.30s</td>
</tr>
<tr>
<td>Loop fission</td>
<td>6</td>
<td>36</td>
<td>1.28s</td>
<td>2.18s</td>
</tr>
<tr>
<td>Loop flattening</td>
<td>1</td>
<td>1</td>
<td>0.07s</td>
<td>3.31s</td>
</tr>
<tr>
<td>Loop fusion</td>
<td>6</td>
<td>36</td>
<td>1.26s</td>
<td>2.19s</td>
</tr>
<tr>
<td>Loop interchange</td>
<td>11</td>
<td>25</td>
<td>1.42s</td>
<td>23.8s</td>
</tr>
<tr>
<td>Loop invariant code motion</td>
<td>3</td>
<td>3</td>
<td>0.22s</td>
<td>0.55s</td>
</tr>
<tr>
<td>Loop peeling</td>
<td>0</td>
<td>0</td>
<td>0s</td>
<td>0.27s</td>
</tr>
<tr>
<td>Loop reversal</td>
<td>4</td>
<td>7</td>
<td>0.25s</td>
<td>0.54s</td>
</tr>
<tr>
<td>Loop skewing</td>
<td>1</td>
<td>1</td>
<td>0.06s</td>
<td>163s</td>
</tr>
<tr>
<td>Loop strength reduction</td>
<td>1</td>
<td>2</td>
<td>1.14s</td>
<td>1.41s</td>
</tr>
<tr>
<td>Loop tiling</td>
<td>1</td>
<td>1</td>
<td>0.07s</td>
<td>4.60s</td>
</tr>
<tr>
<td>Loop unrolling</td>
<td>2</td>
<td>4</td>
<td>0.13s</td>
<td>0.50s</td>
</tr>
<tr>
<td>Loop unswitching</td>
<td>2</td>
<td>2</td>
<td>0.15s</td>
<td>0.77s</td>
</tr>
<tr>
<td>Software pipelining</td>
<td>1</td>
<td>2</td>
<td>0.13s</td>
<td>0.58s</td>
</tr>
</tbody>
</table>
Example of Synthesized WP: Software Pipelining

\[
\begin{align*}
\text{while } V_1 < V_2 \text{ do} & \quad \text{if } V_1 < V_2 \text{ then} \\
S_1 & \quad S_1 \\
S_2 & \quad \Rightarrow \\
V_1 := V_1 + 1 & \quad V_1 := V_1 + 1 \\
& \quad S_1 \\
& \quad S_2 \\
& \quad V_1 := V_1 + 1
\end{align*}
\]

Precondition:

\[V_2 \notin W(S_2) \land ((R(S_1) \cap W(S_2) = \emptyset) \land (R(S_1) \cap W(S_1) = \emptyset) \land (R(S_2) \cap W(S_2) = \emptyset) \lor V_1 \notin W(S_2))\]

(Weaker than PEC’s [PLDI’09])
Verifying Optimizations with CORK

• Template statements/expressions become UFs over the read and write sets
  \[ S_1 \rightarrow S_1(x, y, z) \text{ w/ } R(S_1) = \{x, y, z\} \]

• Originates 2 UF+IA programs
1. UF$s abstracted by polynomials
   - $S_1(x, y, z) \rightarrow ax + by + cz + d$ (w/ $u(S_1) \leq 2$)

2. Loops summarized using recurrences

3. Sequential composition
   - Reduces to safety checking of loop-free + integer arithmetic program
CORK: Polynomial Interpolation

![Graph showing polynomial interpolation with points f(a), f(b), f(c), and f(d).]
CORK: Results

<table>
<thead>
<tr>
<th>Optimization</th>
<th>PEC</th>
<th>Queries</th>
<th>Recurrences</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code hoisting</td>
<td>✓</td>
<td>2</td>
<td>0</td>
<td>0.32s</td>
</tr>
<tr>
<td>Constant propagation</td>
<td>✓</td>
<td>0</td>
<td>0</td>
<td>0.33s</td>
</tr>
<tr>
<td>Copy propagation</td>
<td>✓</td>
<td>0</td>
<td>0</td>
<td>0.33s</td>
</tr>
<tr>
<td>If-conversion</td>
<td>✓</td>
<td>2</td>
<td>0</td>
<td>0.34s</td>
</tr>
<tr>
<td>Partial redundancy elim.</td>
<td>✓</td>
<td>2</td>
<td>0</td>
<td>0.34s</td>
</tr>
<tr>
<td>Loop inv. code motion</td>
<td>✓</td>
<td>7</td>
<td>5</td>
<td>3.48s</td>
</tr>
<tr>
<td>Loop peeling</td>
<td>✓</td>
<td>9</td>
<td>5</td>
<td>3.26s</td>
</tr>
<tr>
<td>Loop unrolling</td>
<td>✓</td>
<td>13</td>
<td>8</td>
<td>12.17s</td>
</tr>
<tr>
<td>Loop unswitching</td>
<td>✓</td>
<td>14</td>
<td>14</td>
<td>8.19s</td>
</tr>
<tr>
<td>Software pipelining</td>
<td>✓</td>
<td>9</td>
<td>5</td>
<td>8.02s</td>
</tr>
<tr>
<td>Loop fission</td>
<td>✓&lt;sub&gt;p&lt;/sub&gt;</td>
<td>10</td>
<td>12</td>
<td>23.45s</td>
</tr>
<tr>
<td>Loop fusion</td>
<td>✓&lt;sub&gt;p&lt;/sub&gt;</td>
<td>10</td>
<td>12</td>
<td>23.34s</td>
</tr>
<tr>
<td>Loop interchange</td>
<td>✓&lt;sub&gt;p&lt;/sub&gt;</td>
<td>15</td>
<td>24</td>
<td>29.30s</td>
</tr>
<tr>
<td>Loop reversal</td>
<td>✓&lt;sub&gt;p&lt;/sub&gt;</td>
<td>7</td>
<td>5</td>
<td>8.41s</td>
</tr>
<tr>
<td>Loop skewing</td>
<td>✓&lt;sub&gt;p&lt;/sub&gt;</td>
<td>16</td>
<td>24</td>
<td>8.50s</td>
</tr>
<tr>
<td>Loop flattening</td>
<td>×</td>
<td>—</td>
<td>—</td>
<td>T/O</td>
</tr>
<tr>
<td>Loop strength reduction</td>
<td>×</td>
<td>6</td>
<td>4</td>
<td>5.63s</td>
</tr>
<tr>
<td>Loop tiling</td>
<td>×</td>
<td>7</td>
<td>9</td>
<td>10.94s</td>
</tr>
</tbody>
</table>
Future Work

- Apply to production compilers
- Synthesize implementation of optimizations (pattern matching, VC Gen, code transformation)
- Explain reasons for optimization failure
- Preserve debug info automatically
- Preserve analysis data across optimizations
• There is significant on-going effort to improve compilers, which compromises correctness
• Presented the first algorithm for the automatic synthesis of WPs for compiler optimizations
• Presented the first algorithm for automatic partial equivalence checking of UF+IA programs
  – Applied to verification of compiler optimizations
CORK: UFs -> Polynomials

- \( f(x_1, \ldots, x_n) = \sum_{\alpha \cdot 1 \leq d} C_{\alpha} X^\alpha \)

- \( u(f) \leq \binom{n+d}{n} \)
2) Synthesize WP for counterexample: Must-write vs may-write

If a variable is in the write set of a statement, it may or may not be written.

\[
\begin{align*}
I_0 & < N_0 \land \\
B_0 & \land \\
I_1 & = \text{ite}(wS_1I, S_1I0, I_0) \land \\
N_1 & = \text{ite}(wS_1N, S_1N0, N_0) \land \\
I_2 & = I_1 + 1 \land \\
I_2 & < N_1 \land \\
\neg B_1 & \land \\
I_3 & = \text{ite}(wS_2I, S_2I0, I_2) \land \\
N_2 & = \text{ite}(wS_2N, S_2N0, N_1) \land \\
I_4 & = I_3 + 1 \land \\
I_4 & \geq N_2
\end{align*}
\]

\[
wS_1I \rightarrow I \in W(S_1) \\
wS_1N \rightarrow N \in W(S_1)
\]
An Optimizer from the Future

Pattern Matching  VC Gen  Code Transformer

Analysis 1  …  Analysis n

Automatic Synthesis of Weakest Preconditions for Compiler Optimizations
An Optimizer from the Future: Pattern Matching

\[
\text{if } B \text{ then } S_1
\]

\[
B \leftrightarrow N > 5 \quad S_1 \leftrightarrow A := A + N
\]

\[
\text{while } I < N \text{ do }
\]

\[
\text{if } N > 5 \text{ then }
\quad A := A + N
\]

\[
\text{else }
\quad A := A + 1
\]

\[
I := I + 1
\]

...
An Optimizer from the Future: Verification

Analysis 1 \rightarrow \ldots \rightarrow \text{Analysis n}

Pattern Matching \rightarrow \text{VC Gen} \rightarrow \text{Code Transformer}

\begin{align*}
B & \mapsto N > 5 \\
S_1 & \mapsto A := A + N
\end{align*}

+ Precondition = \varphi

Range Analysis
Alias Analysis
Scalar Evolution
\ldots

Duality
HSF
Terminator
\ldots
An Optimizer from the Future: Code Transformation

**Pattern Matching** → **VC Gen** → **Code Transformer** → **Analysis n** → **Analysis 1** → **Analysis 1**

**Code Example**:

```
if B then
    S_1

B ⟷ N > 5
S_1 ⟷ A := A + N
```

**Expression**:

```
A_1 := A + 1
if N then
    skip
```