Scaling Up
Superoptimization

Phitchaya Mangpo Phothilimthana (UC Berkeley)
Aditya Thakur (Google)
Rastislav Bodik (University of Washington)
Dinakar Dhurjati (Qualcomm Research)
instl ...  inst2 ...  inst3 ...  inst4 ...  inst5 ...

Optimizer

instl' ...  inst2' ...  inst3' ...  inst4' ...
Rewrite Rules
Rewrite Rules

[opes et al. PLDI’15]

Optimizer
Rewrite Rules

[Lopes et al. PLDI’15]

Optimizer
Rewrite Rules

Search across all possible programs
ARM register-based ISA

GreenArrays stack-based ISA

```
gcc -O3

cmp   r1, #0
mov   r3, r1, asr #31
add   r2, r1, #7
mov   r3, r3, lsr #29
movge r2, r1
ldrb  r0, [r0, r2, asr #3]
add   r1, r1, r3
and   r1, r1, #7
sub   r3, r1, r3
asr   r1, r0, r3
and   r0, r0, #1

82% speedup
asr   r3, r1, #2
add   r2, r1, r3, lsr #29
ldrb  r0, [r0, r2, asr #3]
and   r3, r2, #248
sub   r3, r1, r3
asr   r1, r0, r3
and   r0, r1, #1
```

Expert's

```
push over - push and pop pop and over 0xffffffff or and or
```

Precondition: top 3 elements in the stack are <= 0xffffffff

2.5x speedup
dup push or and pop or
Goal

Develop a **search technique** that can synthesize optimal programs **faster and more consistently**.
We Develop…

Lens

- enumerative search algorithm
- OR
- symbolic search
- stochastic search

Context-aware window decomposition

Cooperative search instances

shared data

<table>
<thead>
<tr>
<th>inst1</th>
<th>inst2</th>
<th>inst3</th>
<th>inst4</th>
<th>inst5</th>
<th>inst6</th>
<th>inst7</th>
</tr>
</thead>
<tbody>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

$p_{pre}$

$p_{spec}$

$p_{post}$
Lens

1. Lens
   - enumerative search algorithm
   - OR
   - symbolic search
   - stochastic search

2. Context-aware window decomposition

3. Cooperative search instances

shared data

- $p_{pre}$
- $p_{spec}$
- $p_{post}$

instances

inst1 ... ...
inst2 ... ...
inst3 ... ...
inst4 ... ...
inst5 ... ...
inst6 ... ...
inst7 ... ...

11
Inductive Synthesis

Find program $p \equiv p_{\text{spec}}$

- **Searching**: Finds a program that passes all test cases
- **Verifying**: Checks equivalence on all inputs using a constraint solver

Candidate program

Counterexample

Yes → done

No
Search space of $k$-instruction long programs

- program $p \equiv p_{spec}$ (on all inputs)
- program $p \not\equiv p_{spec}$
$n$ test cases

Search space of $k$-instruction long programs

- program $p \equiv p_{spec}$ (on all inputs)
- program $p \not\equiv p_{spec}$
Search space of $k$-instruction long programs

- program $p$ possibly $\equiv p_{spec}$
- program $p \equiv p_{spec}$
- program $p \not\equiv p_{spec}$
Search space of $k$-instruction long programs

- program $p$ possibly $\equiv p_{\text{spec}}$
- program $p \equiv p_{\text{spec}}$
- program $p \not\equiv p_{\text{spec}}$
Existing techniques
[Barthe et al. PPoPP'13]
[Udupa et al. PLDI’13]
Existing techniques
[Barthe et al. PPoPP’13]
[Udupa et al. PLDI’13]

Inefficiency
Revisit programs that have been pruned away previously.
Existing techniques

[Barthe et al. PPOPP’13]
[Udupa et al. PLDI’13]

Inefficiency

Revisit programs that have been pruned away previously.
Existing techniques
[Barthe et al. PPoPP’13]
[Udupa et al. PLDI’13]

Inefficiency
Revisit programs that have been pruned away previously.
Existing techniques
[Barthe et al. PPoPP’13]
[Udupa et al. PLDI’13]
[Bansal et al. ASPLOS’06]

+ Selective refinement
Existing techniques
[Barthe et al. PPoPP’13]
[Udupa et al. PLDI’13]
[Bansal et al. ASPLOS’06]

+ Selective refinement

+ Bidirectional search
related work
[Bansal, Thesis’08]
Problem Formulation

Superoptimization = graph search problem

Problem: find program $p \equiv p_{\text{spec}}$ with respect to a set of test cases

Example: program state $<r_0, r_1>$

Ordered set of test inputs

Path from $s$ to $t = \text{program } p$

Expected outputs
Lens Algorithm

test case 1

Depth 0

forward
Depth 1

mov r1, 4
sub r1, r0, r1, lsl 1
clz r1, r1
clz r0, r0
<4,4>
<1,0>
<1,4>
<4,0>

backward
Depth 3

Depth 4

<1,->

<4,->
Lens Algorithm

Test case 1

Depth 0

Forward

Depth 1

Depth 2

Depth 3

Depth 4

Backward

<4,4>

<4,0>

<1,0>

<1,4>

<4,0>

<1,->

<-3,->

<1,->
Lens Algorithm

**Forward**

- Depth 0
- 1
- 2

**Backward**

- 3
- 4

**Test Case 1**
- $S_1$
- $<4,0>$
- $<4,4>$

**Refine**

- $u_1$
- $<1,0>$
- $<1,4>$

**Counterexample**

$p(<1,2>) \neq p_{spec}(<1,2>)$

**Test Case 2**
- $S_2$
- $<1,2>$
- $t_2$
- $<0,->$
## Lens Algorithm

### Test Case 1

<table>
<thead>
<tr>
<th>Depth</th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><img src="image1" alt="Forward 0" /></td>
<td><img src="image2" alt="Backward 0" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="image3" alt="Forward 1" /></td>
<td><img src="image4" alt="Backward 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image5" alt="Forward 2" /></td>
<td><img src="image6" alt="Backward 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image7" alt="Forward 3" /></td>
<td><img src="image8" alt="Backward 3" /></td>
</tr>
</tbody>
</table>

### Test Case 2

<table>
<thead>
<tr>
<th>Depth</th>
<th>Forward</th>
<th>Backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><img src="image9" alt="Forward 0" /></td>
<td><img src="image10" alt="Backward 0" /></td>
</tr>
<tr>
<td>1</td>
<td><img src="image11" alt="Forward 1" /></td>
<td><img src="image12" alt="Backward 1" /></td>
</tr>
<tr>
<td>2</td>
<td><img src="image13" alt="Forward 2" /></td>
<td><img src="image14" alt="Backward 2" /></td>
</tr>
<tr>
<td>3</td>
<td><img src="image15" alt="Forward 3" /></td>
<td><img src="image16" alt="Backward 3" /></td>
</tr>
</tbody>
</table>

**Depth 0**

1. **U₁**: clz r0, r0
2. **V₁**: sub r0, r0, r1
3. **T₁**: lsr r0, r0, 3
4. **S₁**: mov r1, 4
5. **Clz r0, r0**
6. **Sub r0, r0, r1**
7. **Clz r1, r1**

**Depth 1**

1. **U₂**: clz r0, r0
2. **V₂**: lsr r0, r0, 3
3. **T₂**: lsr r0, r0, 3
4. **S₂**: mov r1, 4
5. **Clz r0, r0**
6. **Sub r0, r0, r1**
7. **Clz r1, r1**

**Depth 2**

1. **U₃**: clz r0, r0
2. **V₃**: lsr r0, r0, 3
3. **T₃**: lsr r0, r0, 3
4. **S₃**: mov r1, 4
5. **Clz r0, r0**
6. **Sub r0, r0, r1**
7. **Clz r1, r1**
**Lens Algorithm**

<table>
<thead>
<tr>
<th>Depth 0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>forward</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>backward</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**test case 1**

**refine**

**test case 2**
Lens: Reduced Bitwidth

Challenge by backward traversal:

32-bit domain

...<0,7><1,6>...

<2^{31-1}, 2^{31}+8>

2^{32} edges!

4-bit domain

...<0,7><1,6>...

<7,0>

16 edges!

Solution:

- Search in reduced-bitwidth domain
- Verify in the original domain
Lens: Evaluation

**ARM**
- Bit-twiddling benchmarks from *Hacker’s Delight*
- Input = code generated from `gcc -O0`
- Timeout = 1 hour

**GreenArrays (GA)** 18-bit stack-based architecture
- Frequently-executed basic blocks from MD5, SHA-256, FIR, sine, and cosine functions
- Input = code generated from Chlorophyll compiler without optimizations [Phothilimthana et al. PLDI’14]
- Timeout = 20 min
Lens vs. Existing Techniques

Number of solved benchmarks

<table>
<thead>
<tr>
<th>Approach</th>
<th>Benchmarks</th>
<th>Search Time Speed Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Existing techniques (baseline)</td>
<td>4</td>
<td>2.3x</td>
</tr>
<tr>
<td>+ Selective refinement</td>
<td>7</td>
<td>5.2x</td>
</tr>
<tr>
<td>+ Reduced bitwidth</td>
<td>6</td>
<td>2.7x</td>
</tr>
<tr>
<td>+ Bidirectional search</td>
<td>9</td>
<td>12x</td>
</tr>
</tbody>
</table>

ARM (12 benchmarks)

GA (11 benchmarks)
Context-Aware Window Decomposition

1. Lens
   - enumerative search algorithm
   - OR symbolic search
   - stochastic search

2. Context-aware window decomposition

3. Cooperative search instances

shared data
Find program $p$ such that $p_{\text{pre}} + p + p_{\text{post}} \equiv p_{\text{pre}} + p_{\text{spec}} + p_{\text{post}}$

\begin{align*}
\text{Context} & \text{ provides implicit pre and post condition.}
\end{align*}

Related work: Conditionally Correct Superoptimization [Sharma et al. OOPSLA’15]
Optimize bitarray benchmark from MiBench (embedded system benchmark suite)

| cmp  r1, #0  | cmp r1, #0 |
| mov r3, r1, asr #31 | mov r3, r1, asr #31 |
| add r2, r1, #7 | add r2, r1, #7 |
| mov r3, r3, lsr #29 | mov r3, r3, lsr #29 |
| movge r2, r1 | movge r2, r1 |
| ldrb r0, [r0, r2, asr #3] | ldrb r0, [r0, r2, asr #3] |
| add r1, r1, r3 | bic r1, r2, #248 |
| and r1, r1, #7 | sub r3, r1, r3 |
| sub r3, r1, r3 | asr r1, r0, r3 |
| asr r1, r0, r3 | and r0, r1, #1 |
| and r0, r0, #1 | and r0, r1, #1 |
On 6 out of 12 benchmarks, context-aware decomposition improves code significantly.

![Bar chart showing normalized cost comparison between context-aware and non-context-aware decompositions.](chart)

Lower is better.
Cooperative Superoptimizer

1. **Lens**
   - Enumerative search algorithm
   - OR
   - Symbolic search
   - Stochastic search

2. Context-aware window decomposition

3. Cooperative search instances

$p_{best}$

shared data
### Other Search Techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enumerative</td>
<td><img src="image" alt="Diagram" /></td>
<td>Can apply many pruning strategies specific to program synthesis problem.</td>
<td>Takes a long time to get to big programs. Require a lot of memory.</td>
</tr>
</tbody>
</table>

**Example:**

```
 cmp r0, r1
 movls r0, #0

 cmp r0, r1
 movls r1, #0
```
Cooperative Superoptimizer

- **Enumerative**
  - Symbolic
  - Stochastic

- Performance cost

- Search instances

- Found new $p_{best}$

- Communication

- $p_{best}$ and decompose

- + new const

- $p_{best}$ and decompose

- + new const

- $p_{best}$ and mutate

- $p_{best}$ and mutate
Cooperative Superoptimizer

Search instances

performance cost

- **Enumerative**
- **Symbolic**
- **Stochastic**

---

- **without cooperation**
Cooperative: Evaluation

- Run each benchmark 3 times
- Normalize performance costs by cost of best known program
  Lower is better. Everything = 1 is the best.

```
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Normalized Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stochastic&lt;o&gt;</td>
<td>1</td>
</tr>
<tr>
<td>Symbolic</td>
<td></td>
</tr>
<tr>
<td>Enumerative</td>
<td></td>
</tr>
<tr>
<td>Cooperative</td>
<td></td>
</tr>
</tbody>
</table>
```

Perfect at this benchmark.
Cooperative: Evaluation

- Run each benchmark 3 times
- Normalize performance costs by cost of best known program
  Lower is better. Everything = 1 is the best.

Search time speed up:
- 13x
- 20x
- 1.3x
Runtime Speedup

Runtime speedup over `gcc –O3` on an actual ARM Cortex-A9

**Benchmarks** Hacker’s Delight, WiBench (wireless system kernel benchmarks), MiBench (embedded system kernel benchmarks)

<table>
<thead>
<tr>
<th>Program</th>
<th>Search time (s)</th>
<th>gcc –O3 length</th>
<th>Output length</th>
<th>Runtime speedup on ARM Cortex-A9</th>
</tr>
</thead>
<tbody>
<tr>
<td>p18</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>2.11</td>
</tr>
<tr>
<td>p21</td>
<td>1139</td>
<td>6</td>
<td>5</td>
<td>1.81</td>
</tr>
<tr>
<td>p23</td>
<td>665</td>
<td>18</td>
<td>16</td>
<td>1.48</td>
</tr>
<tr>
<td>p24</td>
<td>151</td>
<td>7</td>
<td>4</td>
<td>2.75</td>
</tr>
<tr>
<td>p25</td>
<td>2</td>
<td>11</td>
<td>1</td>
<td>17.80</td>
</tr>
<tr>
<td>WB-txrate5a</td>
<td>32</td>
<td>9</td>
<td>8</td>
<td>1.31</td>
</tr>
<tr>
<td>WB-txrate5b</td>
<td>66</td>
<td>8</td>
<td>7</td>
<td>1.29</td>
</tr>
<tr>
<td>MB-bitarray</td>
<td>612</td>
<td>10</td>
<td>6</td>
<td>1.82</td>
</tr>
<tr>
<td>MB-bitshift</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>1.11</td>
</tr>
<tr>
<td>MB-bitcnt</td>
<td>645</td>
<td>27</td>
<td>19</td>
<td>1.33</td>
</tr>
<tr>
<td>MB-susan-391</td>
<td>32</td>
<td>30</td>
<td>21</td>
<td>1.26</td>
</tr>
</tbody>
</table>
Provide cooperative search strategy.
Enable rapid retargeting of the superoptimizer to a new ISA.

github.com/mangpo/greenthumb