Verifying & Improving Halide's Term Rewriting System with Program Synthesis

Julie L. Newcomb (University of Washington), Andrew Adams (Adobe Research), Steven Johnson (Google), Ras Bodik (University of Washington), Shoaib Kamil (Adobe Research)

Halide

```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

A key component!

Running benchmark suite without this engine resulted in a 5x increase in compilation times and a 26x increase in runtimes

Uses of the reasoning engine

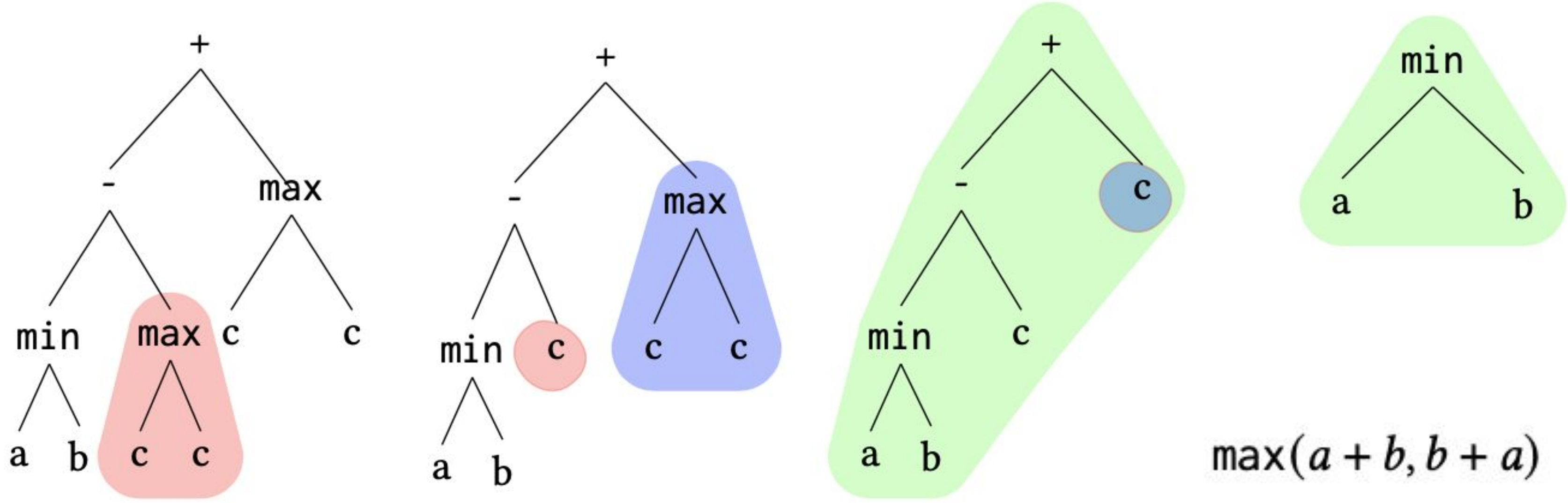
Halide code	**If we can show ….**	**Optimization**
Euclidean division	Denominator > 0	Machine insn division
Loop	Constant loop extent	Map loop to CUDA threads
`malloc`	Constant allocation size	Keep on stack

Talk roadmap

- Background
- Soundness
- Termination
- Greater reasoning power

Term rewriting systems

$$\mathsf{max}(x, x) \rightarrow_R x$$
$$(x - y) + y \rightarrow_R x$$



$$\mathsf{max}(a + b, b + a)$$

Limits to reasoning power

Reasoning power vs. performance

Nonlinear integer arithmetic is undecidable → the Halide TRS cannot be complete

Talk roadmap

- Background
- **Soundness**
- Termination
- Greater reasoning power

Proofs of Correctness: Soundness

SMT Solver

Verified 88% of ruleset

Proof assistant

Verified 12% of ruleset

An example of an unsound rule

Wrong	$\frac{x*c_0}{c_1} \rightarrow_R \frac{x}{(c_1/c_0)}$ if $c_1 \ \% \ c_0 = 0 \wedge \boxed{c_1 > 0}$
Fixed	$\frac{x*c_0}{c_1} \rightarrow_R \frac{x}{(c_1/c_0)}$ if $c_1 \ \% \ c_0 = 0 \wedge \boxed{c_0 > 0 \wedge \frac{c_1}{c_0} \neq 0}$
Counterexample	$c_0 = -1, c_1 = 2, x = 1$

Reverification of new semantics

Division redefined:

- Previously x/0 was undefined behavior
- Now, x/0 == 0
- Some sample rules that became incorrect:

$$
\begin{aligned}
(x/y) * y + x\%y &\rightarrow_R x \\
-1/x &\rightarrow_R \texttt{select}(x < 0, 1, -1) \\
(x+y)/x &\rightarrow_R y/x + 1
\end{aligned}
$$

Talk roadmap

- Background
- Soundness
- **Termination**
- Greater reasoning power

Proofs of correctness: termination

$$x + y \rightarrow_R y + x$$

$3 + x \rightarrow x + 3 \rightarrow 3 + x \rightarrow$ …...

Reduction order

A reduction order is a total order defined over terms.

$$x + x + y >_{+} (x * 2) + y$$

The Halide simplification order

$s > t$ if

1. There are more vector operations in s than in t. If they have the same number of vector operations …
2. There are more division, multiplication, and modulo operations (sum) in s than t. If they are the same …
3. There are more occurrences of variables and constants in s than in t …
4. ...
5. ...
6. ...

Talk roadmap

- Background
- Soundness
- Termination
- **Greater reasoning power**

Strengthening the ruleset via synthesis

But: set of potential rules is infinite

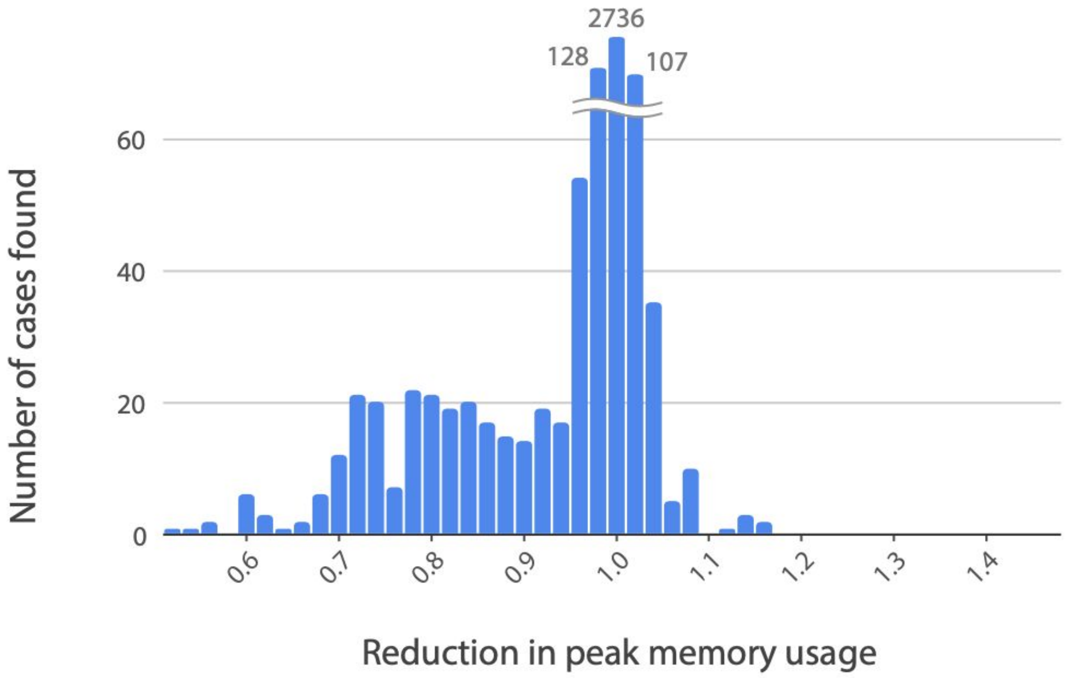
So we are guided by the bias in expressions seen by the compiler

Our synthesis pipeline (briefly)

Input expression	$(y+2) < \texttt{select}(u < z, -3, 4) + (y+2)$
Generated LHS patterns	$\dots$ $(x_1+2) < x_2 + (x_1+2)$ $x_1 < \texttt{select}(x_2, -3, 4) + x_1$ $\texttt{select}(x_1 < x_2, -3, 4)$ $\dots$
Superoptimized with CEGIS	$x_1 < \texttt{select}(x_2, -3, 4) + x_1 \rightarrow \boxed{\neg x_2}$
With symbolic constants	$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \ \rightarrow_R \ \neg x_2$
Rule with predicate	$x_1 < \texttt{select}(x_2, c_0, c_1) + x_1 \ \rightarrow_R \ \neg x_2 \text{ if } 0 < c_1 \wedge c_0 \leq 0$

Case studies & anecdotes

- Synthesized bug fixes as good or better than the human-authored fixes
- Used corpus of ~100k expressions to synthesize ~4000 rules, resulting in reductions of peak memory usages and no appreciable increase in compilation times.
- Human developers have used the synthesizer as an assistant



Verifying & Improving Halide's Term Rewriting System with Program Synthesis

Julie L. Newcomb (University of Washington), Andrew Adams (Adobe Research), Steven Johnson (Google), Ras Bodik (University of Washington), Shoaib Kamil (Adobe Research)