Verified Compilation and Optimization of Floating-Point Programs in CakeML

Heiko Becker, Robert Rabe, Eva Darulova, Magnus O. Myreen, Zachary Tatlock, Ramana Kumar, Yong Kiam Tan, Anthony Fox
Floating-Point Arithmetic in Unverified & Verified Compilers
Floating-Point Arithmetic in Unverified & Verified Compilers

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof
Floating-Point Arithmetic in Unverified & Verified Compilers

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees

GCC

LLVM

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

COMPCERT
Floating-Point Arithmetic in Unverified & Verified Compilers

GCC

• IEEE-754 arithmetic
• fast-math optimizations
• no correctness guarantees

LLVM

• no floating-point support

CAKE ML

• IEEE-754 arithmetic
• no performance optimizations
• full correctness proof

COMPCERT
Floating-Point Arithmetic in Unverified & Verified Compilers

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees

before our work

- no floating-point support

before our work

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof
Floating-Point Arithmetic in Unverified & Verified Compilers

- IEEE-754 arithmetic
- fast-math-style optimizations
- correctness & accuracy proofs

- IEEE-754 arithmetic
- no performance optimizations
- full correctness proof

in this talk

- IEEE-754 arithmetic
- fast-math optimizations
- no correctness guarantees
Fast-Math-Style Optimizations in Compilers

\[ x \times (x \times (x \times x)) \rightarrow (x \times x) \times (x \times x) \]
Fast-Math-Style Optimizations in Compilers

\[ x \times (x \times (x \times x)) \rightarrow (x \times x) \times (x \times x) \]

changes bit-level result
Fast-Math-Style Optimizations in Compilers

LLVM

\[ x \times (x \times (x \times x)) \rightarrow (x \times x) \times (x \times x) \]

changes bit-level result

preserve IEEE-754 floating-point arithmetic
Fast-Math-Style Optimizations in Compilers

\[ x \ast (x \ast (x \ast x)) \rightarrow (x \ast x) \ast (x \ast x) \]

changes bit-level result

preserve IEEE-754 floating-point arithmetic

requires bit-level accuracy
Fast-Math-Style Optimizations in Compilers

\[ x \times (x \times (x \times x)) \rightarrow (x \times x) \times (x \times x) \]

changes bit-level result

preserves IEEE-754 floating-point arithmetic

requires bit-level accuracy
Fast-Math-Style Optimizations in Compilers

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler

Heiko Becker¹, Eva Darulova¹, Magnus O. Myreen², and Zachary Tatlock³*

¹ MPI-SWS, Saarland Informatics Campus (SIC), {hbecker,eva}@mpi-sws.org
² Chalmers University of Technology, myreen@chalmers.se
³ University of Washington, ztatlock@cs.washington.edu

Abstract. Verified compilers like CompCert and CakeML offer increasingly sophisticated optimizations. However, their deterministic source

$x \cdot (x \cdot (x \cdot x))$

changes bit-level result

preserve IEEE-754 floating-point arithmetic

accuracy changes bit-level accuracy

CAV’19
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
verified floating-point optimizations

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
proof-of-concept optimizer

verified floating-point optimizations

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
proof-of-concept optimizer
verified floating-point optimizations
fine-grained control

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]

- proof-of-concept optimizer
- verified floating-point optimizations
- fine-grained control
- no accuracy guarantees
- non-deterministic semantics
- technical challenge
proof-of-concept optimizer

verified floating-point optimizations

fine-grained control

no accuracy guarantees

missing

technical challenge

non-deterministic semantics

Icing: Supporting Fast-math Style Optimizations in a Verified Compiler [CAV’19]
Why Accuracy Matters

(* require(1.0 \leq x \leq 100.0 \land 
1.0 \leq y \leq 100.0) *)

fun cartToPol_x (x:double, y:double):double =
 sqrt((x * x) + (y * y))
Why Accuracy Matters

```haskell
(* require(1.0 ≤ x ≤ 100.0 ∧
    1.0 ≤ y ≤ 100.0)
  *)
fun cartToPol_x (x:double, y:double):double =
  sqrt((x * x) + (y * y))
```
Why Accuracy Matters

(* require(1.0 ≤ x ≤ 100.0 ∧ 1.0 ≤ y ≤ 100.0) *)

fun cartToPol_x (x:double, y:double):double = sqrt((x * x) + (y * y))

machine code
Why Accuracy Matters

```
(* require(1.0 ≤ x ≤ 100.0 ∧ 1.0 ≤ y ≤ 100.0) *)
fun cartToPol_x (x:double, y:double):double =
sqrt((x * x) + (y * y))
```
Why Accuracy Matters

```plaintext
(* require(1.0 ≤ x ≤ 100.0 ∧
    1.0 ≤ y ≤ 100.0)
*)

fun cartToPol_x (x:double, y:double):double =
    sqrt((x * x) + (y * y))
```

machine code
Why Accuracy Matters

(* require(1.0 ≤ x ≤ 100.0 ∧ 1.0 ≤ y ≤ 100.0) *)
output error: 2^{-5} *)
fun cartToPol_x (x:double, y:double):double =
sqrt((x * x) + (y * y))

program designed for unavoidable input and output errors

machine code
Why Accuracy Matters

program designed for unavoidable input and output errors

(fun cartToPol_x (x:double, y:double):double = sqrt((x * x) + (y * y))

(output error: \(2^{-5}\))

(machine code)

roundoff error \(\leq\) output error
Why Accuracy Matters

(* require(1.0 ≤ x ≤ 100.0 ∧
    1.0 ≤ y ≤ 100.0) *)

output error: $2^{-5}$ *)

fun cartToPol_x (x:double, y:double):double =
    sqrt((x * x) + (y * y))

program designed for unavoidable
input and output errors

roundoff error ≤ output error

machine code  machine code  machine code
Why Accuracy Matters

(* require(1.0 ≤ x ≤ 100.0 ∧ 1.0 ≤ y ≤ 100.0) *)
output error: 2^(-5) *)

fun cartToPol_x (x:double, y:double):double =
    sqrt((x * x) + (y * y))

program designed for unavoidable input and output errors

roundoff error ≤ output error

error refinement
any optimized implementation below output error is fine
Contributions

RealCake:

• extends CakeML with **relaxed non-deterministic floating-point semantics**

• optimizes with a **fast-math optimizer**

• **soundly proves roundoff errors** of floating-point kernels with automated tools

• proves **error refinement**
Contributions

RealCake:

• extends CakeML with relaxed non-deterministic floating-point semantics
• optimizes with a fast-math optimizer
• soundly proves roundoff errors of floating-point kernels with automated tools
• proves error refinement
source program
with output error
The RealCake Compiler Zoomed In

source program with output error  \rightarrow \text{fast-math optimizer} \rightarrow \text{optimized program}
The RealCake Compiler Zoomed In

source program with output error → fast-math optimizer → optimized program

relaxed floating-point semantics
The RealCake Compiler Zoomed In

- **source program with output error**
  - fast-math optimizer
  - optimized program

- relaxed floating-point semantics

- **machine code**
The RealCake Compiler Zoomed In

- source program with output error
- fast-math optimizer
- optimized program
- relaxed floating-point semantics
- accuracy analysis
- accuracy bound
- machine code
The RealCake Compiler Zoomed In

- source program with output error
- fast-math optimizer
- optimized program
- accuracy analysis
- accuracy bound
- error refinement proof
- relaxed floating-point semantics
- real-valued semantics
- machine code

CAKEML
A Verified Implementation of ML
The RealCake Compiler Zoomed In

- **source program with output error**
  - real-valued semantics
  - error refinement proof

- **optimized program**
  - fast-math optimizer
  - relaxed floating-point semantics
  - accuracy analysis
  - accuracy bound

- **machine code**
  - oracle-based to encode non-determinism
Floating-Point Programs in CakeML

fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
       val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
       val d = (x1 * x1) + 1.0
       val s = t / d
       val s2 = t2 / d
       in
       x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) +
                  ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
                  ((((3.0 * x1) * x1) * s)) +
                  ((x1 * x1) * x1)) + x1) + (3.0 * s2))
   end)
fun jetEngine(x1:double, x2:double):double = 
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
      val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
      val d = (x1 * x1) + 1.0
      val s = t / d
      val s2 = t2 / d
  in
      x1 + (((((((2.0 * x1) * s) * (s - 3.0)) + 
               ((x1 * x1) * ((4.0 * s) - 6.0))) * d) + 
               ((((3.0 * x1) * x1) * s)) + 
               ((x1 * x1) * x1))) + x1) + (3.0 * s2)
  end)
fun jetEngine(x1:double, x2:double):double = 
  let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1 
  val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1 
  val d = (x1 * x1) + 1.0 
  val s = t / d 
  val s2 = t2 / d 
  in 
  x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) + 
    ((x1 * x1) * ((4.0 * s) - 6.0))) * d) + 
    (((3.0 * x1) * x1) * s)) + 
    ((x1 * x1) * x1)) + x1) + (3.0 * s2)) 
  end
Floating-Point Programs in CakeML

(* output error: $2^{-5}$,
  precondition P: $0.0 \leq x1 \leq 5.0 \land -20.0 \leq x2 \leq 5.0$ *)

fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
           val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
           val d = (x1 * x1) + 1.0
           val s = t / d
           val s2 = t2 / d
           in
           x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) + ((x1 * x1) * ((4.0 * s) - 6.0))) * d) + (((3.0 * x1) * x1) * s)) + ((x1 * x1) * x1)) + x1) + (3.0 * s2))
           end)

Floating-Point Programs in CakeML

(* output error: $2^{-5}$, 
precondition $P$: $0.0 \leq x_1 \leq 5.0 \land -20.0 \leq x_2 \leq 5.0$ *) 

fun jetEngine(x1:double, x2:double):double =
  opt: (let val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
          val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
          val d = (x1 * x1) + 1.0
          val s = t / d
          val s2 = t2 / d
          in
            x1 + (((((((2.0 * x1) * s) * (s - 3.0)) +
                       ((x1 * x1) * ((4.0 * s) - 6.0))) * d) +
                       (((3.0 * x1) * x1) * s)) +
                       ((x1 * x1) * x1)) + x1) + (3.0 * s2)
          end)

optimization annotation  
double operations  
tolerable noise
Floating-Point Programs in CakeML

(* output error: $2^{-5}$,
precondition $P$: $0.0 \leq x_1 \leq 5.0 \land -20.0 \leq x_2 \leq 5.0$ *)

fun jetEngine(x1:double, x2:double):double =
  opt: (let
  val t = (((3.0 * x1) * x1) + (2.0 * x2)) - x1
  val t2 = (((3.0 * x1) * x1) - (2.0 * x2)) - x1
  val d = (x1 * x1) + 1.0
  val s = t / d
  val s2 = t2 / d
  in
  x1 + (((((((((2.0 * x1) * s) * (s - 3.0)) + ((x1 * x1) * ((4.0 * s) - 6.0))) * d) + (((3.0 * x1) * x1) * s)) + ((x1 * x1) * x1)) + x1) + (3.0 * s2))
  end)

input constraints

optimization annotation

double operations

tolerable noise
Optimized Floating-Point Programs

(* guaranteed error bound: $2^{-5}$,
precondition $P$: $0.0 \leq x1 \leq 5.0 \land -20.0 \leq x2 \leq 5.0 *$

fun jetEngine(x1:double, x2:double):double =

noopt: (let
val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
val d = fma(x1, x1, 1.0)
val s = t / d
val s2 = t2 / d

in
x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s, x1 * fma(4.0, s, -6.0)),
  fma(x1 * x1, ((s + s) + s) + x1,
    x1 + ((s2 + s2) + s2)))
end)
Optimized Floating-Point Programs

(* guaranteed error bound: $2^{-5}$, 
precondition $P$: $0.0 \leq x_1 \leq 5.0 \land -20.0 \leq x_2 \leq 5.0$ *)

fun jetEngine(x1:double, x2:double):double =
  noopt: (let
    val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
    val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
    val d = fma(x1, x1, 1.0)
    val s = t / d
    val s2 = t2 / d
  in
    x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s, x1 * fma(4.0, s, -6.0)), fma(x1 * x1, ((s + s) + s) + x1, x1 + ((s2 + s2) + s2)))
  end)
Optimized Floating-Point Programs

(* guaranteed error bound: $2^{-5}$, precondition $P: 0.0 \leq x_1 \leq 5.0 \land -20.0 \leq x_2 \leq 5.0$ *)

fun jetEngine(x1:double, x2:double):double =
  noopt: (let
    val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
    val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
    val d = fma(x1, x1, 1.0)
    val s = t / d
    val s2 = t2 / d
    in
    x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s, x1 * fma(4.0, s, -6.0)), fma(x1 * x1, ((s + s) + s) + x1, x1 + ((s2 + s2) + s2)))
  end)

verified by accuracy analysis

faster, locally more accurate
Optimized Floating-Point Programs

(* guaranteed error bound: $2^{-5}$, precondition $P$: $0.0 \leq x1 \leq 5.0 \land -20.0 \leq x2 \leq 5.0$ *)

fun jetEngine(x1:double, x2:double):double =

noopt: (let
  val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
  val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
  val d = fma(x1, x1, 1.0)
  val s = t / d
  val s2 = t2 / d

  in
  x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s, x1 * fma(4.0, s, -6.0)), fma(x1 * x1, ((s + s) + s) + x1, x1 + ((s2 + s2) + s2)))

  end)
Optimized Floating-Point Programs

(* guaranteed error bound: $2^{-5}$, precondition $P: 0.0 \leq x_1 \leq 5.0 \land -20.0 \leq x_2 \leq 5.0 (*)

fun jetEngine(x1:double, x2:double):double =

noopt: (let
  val t = fma((x1+x1)+x1, x1, (x2 + x2) - x1)
  val t2 = fma((x1+x1)+x1, x1, fma(-2.0, x2, -x1))
  val d = fma(x1, x1, 1.0)
  val s = t / d
  val s2 = t2 / d
  in
  x1 + fma(x1 * d, fma((s - 3.0) + (s - 3.0), s, x1 * fma(4.0, s, -6.0)),
  fma(x1 * x1, ((s + s) + s) + x1, x1 + ((s2 + s2) + s2))
  end)

faster, locally more accurate

verified by accuracy analysis

machine code from IEEE-754 preserving compilation in CakeML

disallow further optimization
jetEngineInputsInPrecond \((s_1, s_2)(w_1, w_2)\) \(\land\) environmentOk \([\text{jetEngine}; s_1; s_2], fs\) \(\Rightarrow\) \(\exists w \ r.\)

CakeMLevaluatesAndPrints \((\text{jetEngineCode}, s_1, s_2, fs)(\text{toString } w)\) \(\land\)
initialFPcodeReturns \(\text{jetEngineUnopt}(w_1, w_2) w\) \(\land\)
realSemanticsReturns \(\text{jetEngineUnopt}(w_1, w_2) r\) \(\land\) abs \((\text{fpToReal } w - r)\) \(\leq 2^{-5}\)
The Final Specification Theorem

\[
\text{jetEngineInputsInPrecond} (s_1, s_2) (w_1, w_2) \land \text{environmentOk} ([\text{jetEngine} ; s_1 ; s_2], fs) \Rightarrow \\
\exists w \ r.
\]

\[
\text{CakeMLEvaluatesAndPrints} (\text{jetEngineCode}, s_1, s_2, fs) (\text{toString} \ w) \land \\
\text{initialFPcodeReturns} \ \text{jetEngineUnopt} (w_1, w_2) \ w \land \\
\text{realSemanticsReturns} \ \text{jetEngineUnopt} (w_1, w_2) \ r \land \text{abs} (\text{fpToReal} \ w - r) \leq 2^{-5}
\]
The Final Specification Theorem

inputs in specified constraints

the program is run with the correct inputs

jetEngineInputsInPrecond (s₁, s₂) (w₁, w₂) \land \text{environmentOk} ([jetEngine; s₁; s₂], fs) \Rightarrow \\
\exists w \, r. \\
\text{CakeMLevaluatesAndPrints} (jet\text{EngineCode}, s₁, s₂, fs) (\text{toString} w) \land \\
\text{initialFPcodeReturns} jet\text{EngineUnopt} (w₁, w₂) w \land \\
\text{realSemanticsReturns} jet\text{EngineUnopt} (w₁, w₂) r \land \text{abs} (\text{fpToReal} w - r) \leq 2^{-5}
The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

jetEngineInputsInPrecond \( s_1, s_2 \) \( (w_1, w_2) \) \( \land \) environmentOk (\( [jetEngine; s_1; s_2], fs \)) \( \Rightarrow \)
\[ \exists w \ r. \]
CakeMLEvaluatesAndPrints (\( jetEngineCode, s_1, s_2, fs \)) (toString \( w \)) \( \land \)
initialFPcodeReturns \( jetEngineUnopt \ (w_1, w_2) \) \( w \) \( \land \)
realSemanticsReturns \( jetEngineUnopt \ (w_1, w_2) \) \( r \) \( \land \) abs (fpToReal \( w \) \( - r \)) \( \leq 2^{-5} \)
The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

jetEngineInputsInPrecond_1(s_1, s_2)(w_1, w_2) \land \text{environmentOk}([\text{jetEngine}; s_1; s_2], fs) \Rightarrow \exists w \ r.

\text{CakeMLevaluatesAndPrints} (\text{jetEngineCode}, s_1, s_2, fs)(\text{toString} \ w) \land \text{initialFPcodeReturns} \ jetEngineUnopt (w_1, w_2) \ w \land \text{realSemanticsReturns} \ jetEngineUnopt (w_1, w_2) \ r \land |\text{fpToReal} \ w - r| \leq 2^{-5}

w also result of nondeterministic semantics
The Final Specification Theorem

inputs in specified constraints

program returns double word w

the program is run with the correct inputs

jetEngineInputsInPrecond \((s_1, s_2)(w_1, w_2) \land \text{environmentOk}([\text{jetEngine}; s_1; s_2], fs) \Rightarrow \exists w \ r.\)

\text{CakeMLevaluatesAndPrints} (\text{jetEngineCode}, s_1, s_2, fs) (\text{toString} w) \land
\text{initialFPcodeReturns} \text{jetEngineUnopt} (w_1, w_2) w \land
\text{realSemanticsReturns} \text{jetEngineUnopt} (w_1, w_2) r \land \text{abs (fpToReal} w - r) \leq 2^{-5}\)

w also result of nondeterministic semantics

real-number semantics returns r
inputs in specified constraints

program returns double word w

the program is run with the correct inputs

w also result of nondeterministic semantics

real-number semantics returns r

output error sound
The Final Specification Theorem

The program returns double word $w$.

The program is run with the correct inputs.

RealCake is the first verified compiler that proves end-to-end accuracy bounds for compiled fast-math optimized programs.
RealCake’s Four-Phase-Optimizer

canonical form
RealCake’s Four-Phase-Optimizer

canonical form

\[ x - y \rightarrow x + ((-1) \times y) \]
\[ ((x + y) + z) \rightarrow x + (y + z) \]

move constants to right-hand sides
RealCake’s Four-Phase-Optimizer

canonical form → undistribute

$x - y \rightarrow x + ((-1) \times y)$

$((x + y) + z \rightarrow x + (y + z))$

move constants to right-hand sides
Real Cake's Four-Phase-Optimizer

canonical form → undistribute

\( (x \times y) + (x \times z) \rightarrow x \times (y + z) \)

\( x - y \rightarrow x + ((-1) \times y) \)

\( ((x + y) + z \rightarrow x + (y + z) \)

move constants to right-hand sides
RealCake’s Four-Phase-Optimizer

canonical form → undistribute → peephole optimizer

\[(x \times y) + (x \times z) \rightarrow x \times (y + z)\]

\[x - y \rightarrow x + ((-1) \times y)\]

\[((x + y) + z) \rightarrow x + (y + z)\]

move constants to right-hand sides
RealCake’s Four-Phase-Optimizer

canonical form  \[ (x \times y) + (x \times z) \rightarrow x \times (y + z) \]

undistribute

peephole optimizer

- \[ x \times 0 \rightarrow 0 \]
- \[ x \times 1 \rightarrow x \]
- \[ x \times -1 \rightarrow -x \]
- \[ x \times 2 \rightarrow x + x \]
- \[ x \times 3 \rightarrow x + (x + x) \]
- \[ x \times y + z \rightarrow \text{fma}(x, y, z) \] ...

move constants to right-hand sides
RealCake’s Four-Phase-Optimizer

- canonical form
- undistribute
- peephole optimizer
- balance trees

(x × y) + (x × z) → x × (y + z)

x − y → x + ((−1) × y)
((x + y) + z → x + (y + z)

move constants to right-hand sides

x × 0 → 0
x × 1 → x
x × −1 → −x
x × 2 → x + x
x × 3 → x + (x + x)
x × y + z → fma(x, y, z) ...

move constants to right-hand sides
RealCake’s Four-Phase-Optimizer

**canonical form**

\[(x \times y) + (x \times z) \rightarrow x \times (y + z)\]

**undistribute**

\[x - y \rightarrow x + ((-1) \times y)\]
\[((x + y) + z) \rightarrow x + (y + z)\]

*move constants to right-hand sides*

**peephole optimizer**

\[w + (x + (y + z)) \rightarrow (w + x) + (y + z)\]

**balance trees**

\[x \times 0 \rightarrow 0\]
\[x \times 1 \rightarrow x\]
\[x \times -1 \rightarrow -x\]
\[x \times 2 \rightarrow x + x\]
\[x \times 3 \rightarrow x + (x + x)\]
\[x \ast y + z \rightarrow \text{fma}(x, y, z) ...\]
RealCake’s Four-Phase-Optimizer

canonical form → undistribute → peephole optimizer → balance trees

each phase proven correct for relaxed floating-point semantics

$x - y \to x + ((-1) \times y)$

$(x + y) + z \to x + (y + z)$

move constants to right-hand sides

$x \times -1 \to -x$

$x \times 2 \to x + x$

$x \times 3 \to x + (x + x)$

$x \times y + z \to \text{fma}(x, y, z) \ldots$
## Experimental Results – Roundoff Errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>fast-math</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>$2.815\times10^{-9}$</td>
<td>$2.463\times10^{-9}$</td>
<td>13%</td>
</tr>
<tr>
<td>delta</td>
<td>$1.970\times10^{-13}$</td>
<td>$2.940\times10^{-12}$</td>
<td>-198%</td>
</tr>
<tr>
<td>doppler2</td>
<td>$6.534\times10^{-13}$</td>
<td>$1.639\times10^{-12}$</td>
<td>50%</td>
</tr>
<tr>
<td>pid</td>
<td>$7.621\times10^{-15}$</td>
<td>$7.727\times10^{-15}$</td>
<td>-1%</td>
</tr>
<tr>
<td>sine_newton</td>
<td>$7.495\times10^{-15}$</td>
<td>$6.270\times10^{-15}$</td>
<td>16%</td>
</tr>
<tr>
<td>sqroot</td>
<td>$1.115\times10^{-15}$</td>
<td>$1.059\times10^{-15}$</td>
<td>5%</td>
</tr>
<tr>
<td>turbine1</td>
<td>$1.588\times10^{-13}$</td>
<td>$1.541\times10^{-13}$</td>
<td>3%</td>
</tr>
</tbody>
</table>
## Experimental Results – Roundoff Errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.815e-09</td>
</tr>
<tr>
<td>delta</td>
<td>1.970e-13</td>
</tr>
<tr>
<td>doppler2</td>
<td>6.534e-13</td>
</tr>
<tr>
<td>pid</td>
<td>7.621e-15</td>
</tr>
<tr>
<td>sine_newton</td>
<td>7.495e-15</td>
</tr>
<tr>
<td>sqroot</td>
<td>1.115e-15</td>
</tr>
<tr>
<td>turbine1</td>
<td>1.588e-13</td>
</tr>
</tbody>
</table>
## Experimental Results – Roundoff Errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>fast-math</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.815e-09</td>
<td>2.463e-09</td>
</tr>
<tr>
<td>delta</td>
<td>1.970e-13</td>
<td>2.940e-12</td>
</tr>
<tr>
<td>doppler2</td>
<td>6.534e-13</td>
<td>1.639e-12</td>
</tr>
<tr>
<td>pid</td>
<td>7.621e-15</td>
<td>7.727e-15</td>
</tr>
<tr>
<td>sine_newton</td>
<td>7.495e-15</td>
<td>6.27e-15</td>
</tr>
<tr>
<td>sqroot</td>
<td>1.115e-15</td>
<td>1.059e-15</td>
</tr>
<tr>
<td>turbine1</td>
<td>1.588e-13</td>
<td>1.541e-13</td>
</tr>
</tbody>
</table>
## Experimental Results – Roundoff Errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>fast-math</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.815e-09</td>
<td>2.463e-09</td>
<td>13%</td>
</tr>
<tr>
<td>delta</td>
<td>1.970e-13</td>
<td>2.940e-12</td>
<td>-198%</td>
</tr>
<tr>
<td>doppler2</td>
<td>6.534e-13</td>
<td>1.639e-12</td>
<td>50%</td>
</tr>
<tr>
<td>pid</td>
<td>7.621e-15</td>
<td>7.727e-15</td>
<td>-1%</td>
</tr>
<tr>
<td>sine_newton</td>
<td>7.495e-15</td>
<td>6.27e-15</td>
<td>16%</td>
</tr>
<tr>
<td>sqroot</td>
<td>1.115e-15</td>
<td>1.059e-15</td>
<td>5%</td>
</tr>
<tr>
<td>turbine1</td>
<td>1.588e-13</td>
<td>1.541e-13</td>
<td>3%</td>
</tr>
</tbody>
</table>
## Experimental Results – Roundoff Errors

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>fast-math</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.815e-09</td>
<td>2.463e-09</td>
<td>13%</td>
</tr>
<tr>
<td>delta</td>
<td>1.970e-13</td>
<td>2.940e-12</td>
<td>-198%</td>
</tr>
<tr>
<td>doppler2</td>
<td>6.534e-13</td>
<td>1.639e-12</td>
<td>50%</td>
</tr>
<tr>
<td>pid</td>
<td>7.621e-15</td>
<td>7.727e-15</td>
<td>-1%</td>
</tr>
<tr>
<td>sine_newton</td>
<td>7.495e-15</td>
<td>6.27e-15</td>
<td>16%</td>
</tr>
<tr>
<td>sqroot</td>
<td>1.115e-15</td>
<td>1.059e-15</td>
<td>5%</td>
</tr>
<tr>
<td>turbine1</td>
<td>1.588e-13</td>
<td>1.541e-13</td>
<td>3%</td>
</tr>
</tbody>
</table>

*trade accuracy for performance*
## Experimental Results – Performance

<table>
<thead>
<tr>
<th>Name</th>
<th>Csts</th>
<th>fast</th>
<th>math</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.05</td>
<td>1%</td>
<td>9%</td>
</tr>
<tr>
<td>delta</td>
<td>13.49</td>
<td>1%</td>
<td>16%</td>
</tr>
<tr>
<td>doppler2</td>
<td>36.00</td>
<td>91%</td>
<td>6%</td>
</tr>
<tr>
<td>pid</td>
<td>104.11</td>
<td>96%</td>
<td>0%</td>
</tr>
<tr>
<td>sine_newton</td>
<td>126.34</td>
<td>92%</td>
<td>0%</td>
</tr>
<tr>
<td>sqroot</td>
<td>87.06</td>
<td>95%</td>
<td>5%</td>
</tr>
<tr>
<td>turbine1</td>
<td>121.02</td>
<td>96%</td>
<td>0%</td>
</tr>
</tbody>
</table>
# Experimental Results – Performance

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.05</td>
</tr>
<tr>
<td>delta</td>
<td>13.49</td>
</tr>
<tr>
<td>doppler2</td>
<td>36.00</td>
</tr>
<tr>
<td>pid</td>
<td>104.11</td>
</tr>
<tr>
<td>sine_newton</td>
<td>126.34</td>
</tr>
<tr>
<td>sqroot</td>
<td>87.06</td>
</tr>
<tr>
<td>turbine1</td>
<td>121.02</td>
</tr>
</tbody>
</table>
## Experimental Results – Performance

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>fast-math</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.05</td>
<td>9%</td>
</tr>
<tr>
<td>delta</td>
<td>13.49</td>
<td>16%</td>
</tr>
<tr>
<td>doppler2</td>
<td>36.00</td>
<td>6%</td>
</tr>
<tr>
<td>pid</td>
<td>104.11</td>
<td>0%</td>
</tr>
<tr>
<td>sine_newton</td>
<td>126.34</td>
<td>0%</td>
</tr>
<tr>
<td>sqroot</td>
<td>87.06</td>
<td>5%</td>
</tr>
<tr>
<td>turbine1</td>
<td>121.02</td>
<td>0%</td>
</tr>
</tbody>
</table>
## Experimental Results – Performance

<table>
<thead>
<tr>
<th>Name</th>
<th>Original</th>
<th>Csts</th>
<th>fast-math</th>
</tr>
</thead>
<tbody>
<tr>
<td>cartToPol</td>
<td>2.05</td>
<td>1%</td>
<td>9%</td>
</tr>
<tr>
<td>delta</td>
<td>13.49</td>
<td>1%</td>
<td>16%</td>
</tr>
<tr>
<td>doppler2</td>
<td>36.00</td>
<td>91%</td>
<td>6%</td>
</tr>
<tr>
<td>pid</td>
<td>104.11</td>
<td>96%</td>
<td>0%</td>
</tr>
<tr>
<td>sine_newton</td>
<td>126.34</td>
<td>92%</td>
<td>0%</td>
</tr>
<tr>
<td>sqroot</td>
<td>87.06</td>
<td>95%</td>
<td>5%</td>
</tr>
<tr>
<td>turbine1</td>
<td>121.02</td>
<td>96%</td>
<td>0%</td>
</tr>
</tbody>
</table>
RealCake:

• proves error refinement for CakeML programs
• extends CakeML with oracle-based floating-point semantics
• optimizes with fast-math-style optimizations
Conclusion

RealCake:

- proves **error refinement for CakeML programs**
- extends CakeML with **oracle-based floating-point semantics**
- optimizes with **fast-math-style optimizations**
- is integrated into official CakeML codebase: [https://code.cakeml.org](https://code.cakeml.org)
Concluding remarks in the paper:
- verified constant lifting optimization
- heuristic to avoid slow-downs
- integration into CakeML toolchain
- implementation of real-numbered and IEEE-754 semantics