Combining Precision Tuning and Rewriting

Brett Saiki, Oliver Flatt, Chandrakana Nandi, Pavel Panchekha, Zachary Tatlock
No one tradeoff is right for every application.

Engineers need to explore the Pareto frontier of optimal accuracy vs. speed candidate implementations!
No one tradeoff is right for every application.

Engineers need to explore the Pareto frontier of optimal accuracy vs. speed candidate implementations!

**Precision Tuning**

e.g., lower 64-bit ⇒ 32-bit
No one tradeoff is right for every application. Engineers need to explore the Pareto frontier of optimal accuracy vs. speed candidate implementations!

**Precision Tuning**
e.g., lower 64-bit ⇒ 32-bit

**Program Rewriting**
e.g., take series expansion
Precision Tuning

Program Rewriting
Precision Tuning

Lower bitwidth $\Rightarrow$ higher throughput

- Major barrier: the memory wall!
- Enable more vectorization, etc.

Difficult to tell where lowering is safe

- Accums. large, but elts small?
- Past work adapts delta debugging
  - [Khalifa et al. FTSCS ‘19]
  - [Rubio-González et al. SC ‘13]
Precision Tuning

- Lower bitwidth ⇒ higher throughput
  - Major barrier: the memory wall!
  - Enable more vectorization, etc.

- Difficult to tell where lowering is safe
  - Accums. large, but elts small?
  - Past work adapts \textit{delta debugging}
    - [Khalifa et al. FTSCS ‘19]
    - [Rubio-González et al. SC ‘13]

Program Rewriting

- Avoid pitfalls and/or use coarser approx
  - Avoid cancellation, intro series
    - e.g., generally want \((x + 1) - x \Rightarrow 1\)

- Difficult to find / carry out good rewrites
  - Need to guide rewrite search
  - Past work applies PL \textit{synthesis}
    - [Schkufza et al. PLDI ‘14]
    - [Panchekha et al. PLDI ‘15]
How to optimize?

\[ \sqrt{\frac{e^{2x}}{e^x - 1} - 1} \]
How to optimize $\sqrt{\frac{e^{2x} - 1}{e^x - 1}}$ via precision tuning?
How to optimize $\sqrt{\frac{e^{2x} - 1}{e^x - 1}}$ via precision tuning?
How to optimize via precision tuning?

\[ \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \]

\( \left( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \right)_{\text{binary32}} \)

Error

Latency

Lower to 32-bit
How to optimize \( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \) via precision tuning?

\[
\left( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \right)_{\text{binary32}}
\]

\[
\left( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \right)_{\text{binary80}}
\]

Raise to 80-bit
How to optimize \( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \) via rewriting?
How to optimize via rewriting?

Just simplification!

\[ \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \]
How to optimize via \( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \) rewriting?

Series expansions

\[ \sqrt{2} \]

\[ \sqrt{2 + x} \]

\[ \sqrt{2 + x + \frac{x^2}{2}} \]

\[ \sqrt{e^x + 1} \]
Optimizing in general: precision tuning OR rewriting?
Optimizing in general: precision tuning OR rewriting?

When and how to use?

- Tune then rewrite?
- Rewrite then tune?
- Alternate? Run to fixpoint?
- Share accuracy analyses?
Optimizing in general: precision tuning AND rewriting!
How to optimize $\sqrt{\frac{e^{2x} - 1}{e^x - 1}}$ via precision tuning AND rewriting!

\[
\text{if } |x| \leq 0.05 : \\
\sqrt{2 + x}
\]
\[
\text{else : } \\
\sqrt{\langle (e^x + 1)_{\text{binary32}} \rangle_{\text{binary64}}}
\]
How to optimize \( \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \) via precision tuning AND rewriting!

Different techniques for different inputs

\[
\begin{align*}
\text{if } |x| &\leq 0.05 : \\
&\sqrt{2 + x} \\
\text{else : } &\sqrt{\langle (e^x + 1)_{\text{binary32}} \rangle_{\text{binary64}}} \\
\end{align*}
\]

Sometimes just rewrite

Sometimes rewrite + tune
Optimizing in general: precision tuning AND rewriting!

Our Result

Combine precision tuning and rewriting to produce a rich set of Pareto-optimal accuracy versus speed trade-offs.
Optimizing in general: precision tuning AND rewriting!

Our Result

Combine precision tuning and rewriting to produce a rich set of Pareto-optimal accuracy versus speed trade-offs.

Key Insights:

- Finer-grained interleavings $\Rightarrow$ better Pareto frontiers
- Precision tuning can be rephrased as a rewriting problem
- “Local Error Analysis” helps both precision tuning and rewriting
Optimizing in general: precision tuning AND rewriting!

Our Result

Combine precision tuning and rewriting to produce a rich set of Pareto-optimal accuracy versus speed trade-offs.

Key Insights:

- Finer-grained interleavings $\Rightarrow$ better Pareto frontiers
- Precision tuning can be rephrased as a rewriting problem
- "Local Error Analysis" helps both precision tuning and rewriting
Outline

- **Herbie**: Improving Accuracy via Rewriting
  - Key Insight: local error guides rewriting

- **Pherbie**: Extending Herbie with Precision Tuning
  - Key Insight: local error also guides precision tuning!

- **Evaluation**: Applying Pherbie to Classics + Graphics
  - Key Insight: Finer-grained interleaving → better optimization!
Optimizing in general: precision tuning AND rewriting!

Our Result

Combine precision tuning and rewriting to produce a rich set of Pareto-optimal accuracy versus speed trade-offs.

Key Insights:

● Finer-grained interleavings $\Rightarrow$ better Pareto frontiers

● Precision tuning can be rephrased as a rewriting problem

● "Local Error Analysis" helps both precision tuning and rewriting

Rewriting:

\[
\begin{align*}
\text{if } |x| \leq 0.05 : & \\
\quad & \frac{\sqrt{2} + x}{\sqrt{2}} \\
\text{else : } & \\
\quad & \sqrt{(e^x + 1)_{\text{binary32}}_{\text{binary64}}} \\
\end{align*}
\]
Developed continuously since 2015
Improves Accuracy Automatically
Rewriting Only
Developed continuously since 2015
Improves Accuracy Automatically
Rewriting Only
Sample Points Measure Error → Rewrites → New Program Less Error
Candidates

Input

Error

$P$ \rightarrow \text{Candidates} \rightarrow P'$

How To Pick?

Improve Loop
How To Pick?

Candidates

Improve Loop

$P \rightarrow \text{Candidates} \rightarrow \text{Improve Loop} \rightarrow P'$
How To Pick?

Input

Error

Candidates

Improve Loop

$P$ → Candidates → $P'$

How To Pick?
Candidates → Regimes

**Improve Loop**

Combine Candidates

$P$ → Candidates

$P'$
**Target High Error**

**A+B → B+A**

**Other Techniques**

**Keep Accurate Programs**

**Prune**

**Rewrite**

**Generate**

**Filter**
Small And Accurate!

Input
\[
\sqrt{\frac{e^{2x} - 1}{e^x - 1}}
\]

Output
\[
\sqrt{e^x + 1}
\]

Accurate, But Slow!

Input
\[(x + 1)\left(\frac{1}{n}\right) - x\left(\frac{1}{n}\right)\]

Output
\[
0.5 \cdot \frac{\log (1 + x)^2}{n \cdot n}
+ \frac{\log (1 + x) - \log x}{n}
+ 0.16666666666666666 \cdot \left(\frac{\log (1 + x)}{n}\right)^3
- \frac{\log x^2}{n \cdot n} \cdot \left(0.5 + 0.16666666666666666 \cdot \frac{\log x}{n}\right)
\]
Outline

✓ **Herbie**: Improving Accuracy via Rewriting
  ● Key Insight: local error guides rewriting

❑ **Pherbie**: Extending Herbie with Precision Tuning
  ● Key Insight: local error also guides precision tuning!

❑ **Evaluation**: Applying Pherbie to Classics + Graphics
  ● Key Insight: Finer-grained interleaving → better optimization!
Pherbie Starting Point: Herbie

$P$ \rightarrow \text{Candidates} \rightarrow \text{Regimes} \rightarrow P'$

\[ \text{Localize} \rightarrow \text{Rewrite} \rightarrow \ldots \rightarrow \text{Prune} \]
Pherbie: Extending Herbie to Combine Tuning + Rewriting
Pherbie: Extending Herbie to Combine Tuning + Rewriting

- $P$
- Candidates
- Regimes
- Prune
- Localize
- Rewrite
- $P_N$
- $P_i$
- $P_1$
Pherbie: Extending Herbie to Combine Tuning + Rewriting

\[ P \rightarrow \text{Candidates} \rightarrow \text{Regimes} \rightarrow \text{Prune} \]

Localize \rightarrow \text{Rewrite} \rightarrow \ldots \rightarrow \text{Prune}
Pherbie: Precision Rewrites

Herbie: \[ \frac{a + b}{c} \Rightarrow \frac{a}{c} + \frac{b}{c} \]

Single global precision
Pherbie: Precision Rewrites

Herbie  \[(a + b) / c \Rightarrow (a / c) + (b / c)\]

Pherbie  \[(a +_{f64} b) /_{f64} c \Rightarrow (a /_{f64} c) +_{f64} (b /_{f64} c)\]
Pherbie: Precision Rewrites

Herbie\[(a + b) / c \Rightarrow (a / c) + (b / c)\]

Pherbie\[(a +_{f64} b) /_{f64} c \Rightarrow (a /_{f64} c) +_{f64} (b /_{f64} c)\]

\[(x)_p \Rightarrow \text{cast}_p (x)_q\]
Pherbie: Precision Rewrites

Herbie \[(a + b) / c \Rightarrow (a / c) + (b / c)\]

Pherbie \[(a +_{f64} b) /_{f64} c \Rightarrow (a /_{f64} c) +_{f64} (b /_{f64} c)\]

\[(x)_p \Rightarrow \text{cast}_p (x)_q\]
Pherbie: Precision Rewrites

Herbie

\[ (a + b) / c \Rightarrow (a / c) + (b / c) \]

Pherbie can use the same rewriting machinery as Herbie!

Pherbie

\[ (x)_p \Rightarrow \text{cast}_p (x)_q \]
Pherbie: Precision Rewrites

\[
\frac{a + b}{c} \Rightarrow \left( \frac{a}{c} \right) + \left( \frac{b}{c} \right)
\]

Pherbie can use the same rewriting machinery as Herbie!

But where should Pherbie apply precision rewrites?
Pherbie: Guide Tuning w/ Local Error
Pherbie: Guide Tuning w/ Local Error

- Rewriting to increase precision at locations with *high local error* improves accuracy.
Pherbie: Guide Tuning w/ Local Error

- Rewriting to increase precision at locations with *high local error* improves accuracy.

- Rewriting to decrease precision at locations with *low local error* improves speed.
Pherbie: Extending Herbie to Combine Tuning + Rewriting

\[ P \rightarrow \text{Candidates} \rightarrow \text{Regimes} \rightarrow \text{Prune} \]

- \( P \)
- Candidates
- Regimes
- Prune

Steps:
1. **Candidates**
2. **Rewrite**
3. **Localize**
4. **Regimes**
5. **Prune**
Pherbie: Pruning

Pruning in general

Generates many candidates

Discards "non-optimal" candidates

Prune
Pherbie: Pruning

Pruning in Herbie:

Criteria
Must be more accurate than every other expression on at least one sampled point
Pherbie: Pruning

Pruning in Herbie:

Criteria
Must be more accurate than every other expression on at least one sampled point

Use in Pherbie?
Accuracy only $\Rightarrow$ slow expressions
Pherbie: Pruning

Pruning in Pherbie:

Candidates

Prune

Criteria
Must be more accurate on at least one sampled point than every other expression \textit{at or below the cost of the candidate}
Pherbie: Pruning

Pruning in Pherbie:

\[ P \rightarrow \text{Candidates} \]

**Criteria**
Must be more accurate on at least one sampled point than every other expression \textit{at or below the cost of the candidate}.

What is “cost”? How do we measure it?
What is “cost”? How do we measure it?

Too expensive to measure precise latency of each candidate

- Need to evaluate candidate many times to get accurate estimator
- Pherbie produces thousands of candidates
**Pherbie: Pruning**

What is “cost”? How do we measure it?

**Key Insight:** Only need relative speed comparison → use a simple cost model!

- Quickly estimates latency
- Sufficient for relative ordering of candidates
What is “cost”? How do we measure it?

Example cost model:

- Operators assigned a cost:
  - Arithmetic: low number (1)
  - Library functions: large number (100)

- Multiply operator cost by bitwidth of representation

- Conditionals: branch conditions cost + largest branch cost
Pherbie: Pruning

What is “cost”? How do we measure it?

Cost models in general

- Simple cost models are good enough
- Better cost models exist
- Pherbie is modular, so users can plug and play
Pherbie: Pruning

At each sampled point

Error

Latency

Keeps accurate expressions
Pherbie: Pruning

At each sampled point, it keeps accurate expressions and also keeps fast expressions as well!
Pherbie: Pruning

At each sampled point

Keeps fast expressions as well!

And every Pareto-optimal candidate in between

Keeps accurate expressions
Pherbie: Extending Herbie to Combine Tuning + Rewriting

\[ P \]

Candidates → Regimes

Localize → Rewrite → Prune

\( P_N \), \( P_i \), \( P_1 \)
Pherbie: Regimes

Pherbie: accuracy and cost

- Need to produce a Pareto frontier!
- Iteratively run Herbie’s regimes algorithm on subset of candidates
Pherbie: Regimes

Pherbie regimes algorithm

1. Run Herbie regimes algorithm on subset cheaper than cost bound
Pherbie: Regimes

Pherbie regimes algorithm

1. Run Herbie regimes algorithm on subset cheaper than cost bound
2. Decrease cost bound so next iteration produces different candidate
Pherbie: Regimes

Pherbie regimes algorithm

1. Run Herbie regimes algorithm on subset cheaper than cost bound
2. Decrease cost bound so next iteration produces different candidate
3. Repeat until no candidate is below cost bound
Pherbie Regimes Example: **Iter 1 / 5**

while 0 < |Candidates| :

\[ p = \text{ExtractMinError(Candidates)} \]

Candidates.removeAboveCost(p)
while 0 < |Candidates| :

\[ p = \text{ExtractMinError}(\text{Candidates}) \]

Candidates.removeAboveCost(p)
Pherbie Regimes Example: **Iter 1 / 5**

while $0 < |\text{Candidates}|$ :

\[
p = \text{ExtractMinError}(\text{Candidates})
\]

\[
\text{Candidates}.\text{removeAboveCost}(p)
\]

\[
\sqrt{\frac{e^{2x} - 1}{e^x - 1}}
\]

\[
\sqrt{\frac{1 + (e^x)^{1.5} \cdot (e^x)^{1.5}}{1 + (e^x + (e^x)^{1.5}) \cdot (\sqrt{e^x} - 1)}}
\]
Pherbie Regimes Example: Iter 2 / 5

while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
Pherbie Regimes Example: \textbf{Iter 2 / 5}

while $0 < |\text{Candidates}|$ :

\begin{align*}
p &= \text{ExtractMinError}(\text{Candidates}) \\
\text{Candidates}.\text{removeAboveCost}(p)
\end{align*}
Pherbie Regimes Example: Iter 2 / 5

while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
Pherbie Regimes Example: **Iter 3 / 5**

\[ \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \]

while \( 0 < |\text{Candidates}| \):

\[ p = \text{ExtractMinError}(\text{Candidates}) \]

\[ \text{Candidates}.\text{removeAboveCost}(p) \]
Pherbie Regimes Example: Iter 3 / 5

while $0 < |Candidates| :$

\begin{align*}
    p &= \text{ExtractMinError}(Candidates) \\
    \text{Candidates}.\text{removeAboveCost}(p)
\end{align*}
Pherbie Regimes Example: **Iter 3 / 5**

```python
while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
```

\[
\sqrt{\frac{e^{2x} - 1}{e^x - 1}}
\]

\[
\text{if } x \leq -0.10591501462198885 : \quad \langle (\sqrt{1 + e^x})_{\text{float 5 16}} \rangle_{\text{binary64}}
\]
\[
\text{else : } \quad \sqrt{2 + (x + x \cdot (x \cdot 0.5))}
\]
Pherbie Regimes Example: Iter 4 / 5

while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)

\[ \sqrt{\frac{e^{2x} - 1}{e^x - 1}} \]
Pherbie Regimes Example: **Iter 4 / 5**

```
while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
```
while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
while 0 < |Candidates| :

    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
Pherbie Regimes Example: Iter 5 / 5

while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
Pherbie Regimes Example: Iter 5 / 5

while 0 < |Candidates| :

    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
while 0 < |Candidates| :
    p = ExtractMinError(Candidates)
    Candidates.removeAboveCost(p)
Outline

✓ **Herbie:** Improving Accuracy via Rewriting
  ● Key Insight: local error guides rewriting

✓ **Pherbie:** Extending Herbie with Precision Tuning
  ● Key Insight: local error also guides precision tuning!

❖ **Evaluation:** Applying Pherbie to Classics + Graphics
  ● Key Insight: Finer-grained interleaving → better optimization!
Evaluation: Benchmark Suites

● **NMSE** - *Numerical Methods for Scientists and Engineers* (Hamming, 1986)
  ○ Standard textbook on numerical analysis

● **PBRT** - *Physically Based Rendering* (Pharr et. al, 2016)
  ○ Open-source textbook describing rendering photorealistic scenes
Evaluation

Pherbie produces Pareto-optimal implementations

Curve Intersection (PBRT)

\[
\left( \sin \left( (1 - u) \cdot \text{normAngle} \right) \cdot \frac{1}{\sin \text{normAngle}} \right) \cdot n_0 + \left( \sin \left( u \cdot \text{normAngle} \right) \cdot \frac{1}{\sin \text{normAngle}} \right) \cdot n_1
\]
Evaluation

Pherbie produces Pareto-optimal implementations

Curve Intersection (PBRT)

\[
\left( \sin \left( (1 - u) \cdot \text{normAngle} \right) \cdot \frac{1}{\sin \text{normAngle}} \right) \cdot n_{0_i} + \left( \sin \left( u \cdot \text{normAngle} \right) \cdot \frac{1}{\sin \text{normAngle}} \right) \cdot n_{1_i}
\]

\[
n_{0_i} + u \cdot (n_{1_i} - n_{0_i})
\]

Herbie’s result
Evaluation

Pherbie produces Pareto-optimal implementations

Nearby Tangent Difference (NMSE)

\[ \tan(x + \epsilon) - \tan x \]
Evaluation

Pherbie produces Pareto-optimal implementations

Nearby Tangent Difference (NMSE)

\[ \tan(x + \epsilon) - \tan x \]

On Pareto frontier!

Herbie’s result
Evaluation

Pherbie produces Pareto-optimal implementations

Beckmann Distribution Sampling (PBRT)

\[
\frac{-\log(1 + u)}{c2p/(\alpha_x \cdot \alpha_x) + s2p/(\alpha_y \cdot \alpha_y)}
\]
Evaluation

Pherbie produces Pareto-optimal implementations

Beckmann Distribution Sampling (PBRT)

\[ \frac{-\log(1 + u)}{c2p/(\alpha_x \cdot \alpha_x) + s2p/(\alpha_y \cdot \alpha_y)} \]

More optimal

Herbie’s result
**Evaluation**

Finer interleavings ⇒ Better Pareto frontier

Comparing different methods of using rewriting and precision tuning:

<table>
<thead>
<tr>
<th>Single Technique</th>
<th>Chaining Techniques</th>
<th>Interleaving Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Herbie</td>
<td>Rewrite-then-tune (RW+BFPT)</td>
<td>Coarse-grained interleaving (PP)</td>
</tr>
<tr>
<td>Herbie x100 (RW)</td>
<td>Tune-then-rewrite (BFPT+RW)</td>
<td>Fine-grained interleaving (Pherbie)</td>
</tr>
<tr>
<td>Tuning-only (BFPT)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

Finer interleavings ⇒ Better Pareto frontier

Method:

- For a given cumulative cost, what is the minimum cumulative error we can achieve by selecting one output expression from each benchmark?
Evaluation

Finer interleavings $\Rightarrow$ Better Pareto frontier

Suite: NMSE

NMSE contains high-error examples
Evaluation

Finer interleavings ⇒ Better Pareto frontier

Suite: PBRT

PBRT is “real world” code
Outline

✓ **Herbie**: Improving Accuracy via Rewriting
  ● Key Insight: local error guides rewriting

✓ **Pherbie**: Extending Herbie with Precision Tuning
  ● Key Insight: local error also guides precision tuning!

✓ **Evaluation**: Applying Pherbie to Classics + Graphics
  ● Key Insight: Finer-grained interleaving → better optimization!
Related Work

- Scalable error analysis
  - [Gopalakrishnan et al. SC’20]

- Improving accuracy of imperative floating point programs
  - [Martel et al. AFM’17]

- Tunable precision of floating point programs
  - [Schkufza et al. PLDI ‘14]

- Sound compilation of real computations
  - [Darulova et al. POPL’14]

- Debugging and correct rounding of floating point programs
  - [Nagarakatte et al. POPL’21, PLDI’21]
Team and Acknowledgments

Brett Saiki  
*UW*

Oliver Flatt  
*Univ. of Utah*

Chandrakana Nandi  
*UW*

Pavel Panchekha  
*Univ. of Utah*

Zachary Tatlock  
*UW*
Pherbie: Precision Tuning + Rewriting

✓ **Herbie**: Improving Accuracy via Rewriting
  - Key Insight: local error guides rewriting

✓ **Pherbie**: Extending Herbie with Precision Tuning
  - Key Insight: local error also guides precision tuning!

✓ **Evaluation**: Applying Pherbie to Classics + Graphics
  - Key Insight: Finer-grained interleaving → better optimization!

herbie.uwplse.org