Expanding the Horizons of Finite-Precision Analysis

Debasmita Lohar

PhD Defense Talk

27th March, 2024
def controller(x: Real, y: Real, z: Real): Real = {
    val res = -x*y - 2*y*z - x - z
    return res
}
def controller(x: Real, y: Real, z: Real): Real = {
    val res = -x*y - 2*y*z - x - z
    return res
}

• Reals are implemented in Floating-point / Fixed-point data type
Errors in Finite-Precision

```python
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    val res = -x*y - 2*y*z - x - z
    return res
}
```

• Reals are implemented in Floating-point / Fixed-point data type

• Introduces roundoff errors at potentially every operation
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    val res = -x * y - 2 * y * z - x - z
    return res
}

0.1 + 0.2 = 0.3

>>> 0.1 + 0.2
0.30000000000000004

real arithmetic

32-bit floating-point arithmetic
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    val res = -x*y - 2*y*z - x - z
    return res
} +/- error

0.1 + 0.2 = 0.3 real arithmetic

>>> 0.1 + 0.2
0.30000000000000004 32-bit floating-point arithmetic

Does it even affect real-world systems?
Finite-Precision Errors in Real World

Gulf War: Loss of accuracy led to failure in US defense system, 28 killed!

April 1992, Schleswig-Holstein, Germany
Rounding error changed Parliament makeup!

Overflow led to explosion of Ariane 5, 39s after lift-off, $370 million lost!

...
Finite-Precision Errors in Real World

Gulf War: Loss of accuracy led to failure in US defense system, 28 killed!  
February 1991, Dhahran, Saudi Arabia

Rounding error changed Parliament makeup!  
April 1992, Schleswig-Holstein, Germany

Overflow led to explosion of Ariane 5, 39s after lift-off, $370 million lost!  
June 1996

Rounding error in luminance computation crashed Android phones  
May, 2020
Finite-Precision Errors in Real World

February 1991, Dhahran, Saudi Arabia
Gulf War: Loss of accuracy led to failure in US defense system, 28 killed!

April 1992, Schleswig-Holstein, Germany
Rounding error changed Parliament makeup!

June 1996
Overflow led to explosion of Ariane 5, 39s after lift-off, $370 million lost!

May 2020
Rounding error in luminance computation crashed Android phones

How do we compute the errors?
def controller(x, y, z): = {
    val res = -x*y - 2*y*z - x - z
    return res
}

ensuring (res +/- ?)
compute a bound on the error
Finite-Precision Accuracy Analysis

```python
(x:Float32, y:Float32, z:Float32): Float32
def controller(x, y, z): = {
  val res = -x*y - 2*y*z - x - z
  return res
}
}
```

Absolute error:

\[ \max_{x,y,z \in I} |f(x, y, z) - \tilde{f}(\tilde{x}, \tilde{y}, \tilde{z})| \]
Finite-Precision Accuracy Analysis

$$\max_{x, y, z \in I} | f(x, y, z) - \tilde{f}(\tilde{x}, \tilde{y}, \tilde{z}) |$$

worst-case error analysis for small programs

Daisy  FLUCTUAT  Rosa
FPTaylor  PRECiSA  ...
def controller(x, y, z):
    res = -x*y - 2*y*z - x - z
    return res

Errors depend on Precision used

(x:Float16, y:Float16, z:Float16)

def controller(x, y, z):
    res = -x*y - 2*y*z - x - z
    return res

} ensuring (res +/- ?)
def controller(x, y, z): ___ = {
    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)

2.02e+00
def controller(x, y, z):
    res = -x*y - 2*y*z - x - z
    return res

Errors depend on Precision used

(x: , y: , z: )

(2.02e+00, 1.58e-4)

Float16 Float32
def controller(x, y, z):
    ___ = {
        val res = -x*y - 2*y*z - x - z
        return res
    } ensuring (res +/- ?)

Errors depend on Precision used

(x:____, y:____, z:____)

<table>
<thead>
<tr>
<th>Precision</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Float16</td>
<td>2.02e+00</td>
</tr>
<tr>
<td>Float32</td>
<td>1.58e-4</td>
</tr>
<tr>
<td>Float64</td>
<td>2.95e-13</td>
</tr>
<tr>
<td>Float128</td>
<td>4.84e-31</td>
</tr>
</tbody>
</table>
So are the Resource Costs!

```python
def controller(x, y, z):
    res = -x*y - 2*y*z - x - z
    return res
```

 Ensuring $\text{res} +/\text{- }$ ?

(x: ___, y: ___, z: ___)
def controller(x, y, z): ___ = {
    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)

So are the Resource Costs!

(x: ___, y: ___, z: ___)
(x:__, y:__, z:__)
def controller(x, y, z): __ = {
    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)

So are the Resource Costs!

We need to find a tradeoff between accuracy and resources!
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)

finite-precision optimization

(x:Float32, y:Float32, z:Float32)

find the lowest precision satisfying error bound
def controller(x: ?, y: ?, z: ?): ? = {
  val res = -x*y - 2*y*z - x - z
  return res
}

ensuring res +/- 0.00197

mixed-precision optimization

- minimize resource cost still satisfying the error
- assign different precisions to different variables
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)

(x:Float32, y:Float32, z:Float32)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)

• minimize resource cost still satisfying the error
• assign different precisions to different variables

Daisy
FPTuner

worse-case tuning for small (floating-point) programs
The Horizons of Finite-Precision Analysis

Accuracy Analysis

- worst-case error analysis for small programs

  Daisy
  FLUCTUAT
  Rosa
  FPTaylor
  PRECiSA
  ...

Optimization

- worst-case tuning for small (floating-point) programs

  Daisy
  FPTuner
Our Work: Extending the Horizon of Finite-Precision Analysis

Accuracy Analysis

- considering probability distribution of inputs

- worst-case error analysis for small programs

- iFM '19  EMSOFT '18

- Probabilistic Analysis

---

Optimization

- worst-case tuning for small (floating-point) programs

- Daisy  FLUCTUAT  Rosa

- FPTaylor  PRECiSA  ...

- Daisy  FPTuner
Our Work: Extending the Horizon of Finite-Precision Analysis

Accuracy Analysis

- worst-case error analysis for small programs
- Probabilistic Analysis
- Static + Dynamic Analysis

Handling larger programs

Optimization

- worst-case tuning for small (floating-point) programs
- Daisy
- FLUCTUAT
- Rosa
- FPTaylor
- PRECiSA
- ...
Our Work: Extending the Horizon of Finite-Precision Analysis

Accuracy Analysis

Probabilistic Analysis

Static + Dynamic Analysis

worst-case error analysis for small programs

Daisy

FLUCTUAT

Rosa

FPTaylor

PRECiSA

Optimization

specializing mixed fixed tuning for NNs

EMSOFT ‘19

EMSOFT ‘18

TACAS ‘21

NN Quantization

worst-case tuning for small (floating-point) programs

Daisy

FPTuner
Today's Talk: Probabilistic Error Analysis and NN Quantization

Accuracy Analysis
- Probabilistic Analysis
- Static + Dynamic Analysis
  - worst-case error analysis for small programs
- Tools:
  - Daisy
  - FLUCTUAT
  - Rosa
  - FPTaylor
  - PRECiSA
  - ...

Optimization
- NN Quantization
  - worst-case tuning for small (floating-point) programs
- Tools:
  - Daisy
  - FPTuner

Conferences:
- iFM '19
- EMSOFT '18
- TACAS '21
- EMSOFT '23
How do we take into account uncertainties in the inputs and compute the distribution of errors at the output?
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    require (-15.0 <= x, y, z <= 15.0)
    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)

State-of-the-Art: Worst-Case Error Analysis

Daisy FLUCTUAT Rosa
FPTaylor PRECiSA ...

absolute error: 1.7e-4
Worst-case can be pessimistic!

```scala
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
  require (-15.0 <= x, y, z <= 15.0)
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```
Worst-case can be pessimistic!

```scala
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  require (-15.0 <= x, y, z <= 15.0)
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```

For most of inputs, errors are small!
Scenario 1: Applications may tolerate large infrequent errors

def controller(x:Float32, y:Float32, z:Float32): Float32 = {
    require (-15.0 <= x, y, z <= 15.0)
    val res = -x*y - 2*y*z - x - z
    return res
}

ensuring (error <= 1.5e-4, 0.85)

tolerates big errors occurring with ≤ 0.15 probability
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    require (-15.0 <= x, y, z <= 15.0)
    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (error <= 1.5e-4, 0.85)

Scenario 1: Applications may tolerate large infrequent errors

tolerates big errors occurring with <= 0.15 probability

(x: Float64, y: Float64, z: Float64): Float64

worst-case error: 1.7e-4
Scenario 1: Applications may tolerate large infrequent errors

```
import numpy as np

def controller(x: float, y: float, z: float) -> float:
    require (-15.0 <= x, y, z <= 15.0)
    val res = -x*y - 2*y*z - x - z
    return res

ensuring (error <= 1.5e-4, 0.85)
```

tolerates big errors occurring with <= 0.15 probability

We need to analyze roundoff errors probabilistically!
Our Contribution: Probabilistic Analysis for Roundoff Errors

```python
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    require (-15.0 <= x, y, z <= 15.0)

    x := gaussian(4.0, 0.5)
    y := gaussian(4.75, 0.2)
    z := gaussian(4.8, 0.25)

    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (error <= 1.5e-4, 0.85)
```
Our Contribution: Probabilistic Analysis for Roundoff Errors

```python
def controller(x: Float32, y: Float32, z: Float32): Float32 = {
    require (-15.0 <= x, y, z <= 15.0)
    x := gaussian(4.0, 0.5)
    y := gaussian(4.75, 0.2)
    z := gaussian(4.8, 0.25)
    val res = -x*y - 2*y*z - x - z
    return res
} ensuring (error <= 1.5e-4, 0.85)
```

✓ probability distribution of errors
✓ a refined error that occurs with the threshold probability
Overview: Sound Probabilistic Roundoff Error Analysis

finite-precision program with probabilistic inputs

probabilistic error analysis
Overview: Sound Probabilistic Roundoff Error Analysis

finite-precision program with probabilistic inputs

probabilistic interval subdivision

probabilistic error analysis
Probabilistic Interval Subdivision

finite-precision program with probabilistic inputs

require \((-15.0 \leq x, y, z \leq 15.0)\)
Probabilistic Interval Subdivision

finite-precision program with probabilistic inputs

require (-15.0 <= x, y, z <= 15.0)
Probabilistic Interval Subdivision

finite-precision program with probabilistic inputs

Probabilistic Interval Subdivision

require (-15.0 <= x, y, z <= 15.0)

∀i \in x, ∀j \in y, ∀k \in z, p_{ijk} = x_i \times y_j \times z_k

subdomain with a probability taking Cartesian Product:
Probabilistic Error Analysis

finite-precision program with probabilistic inputs

probalistic interval subdivision

error analysis

\[ -x*y - 2*y*z - x - z \]

\[ < s_{ijk}, p_{ijk} > \]
Probabilistic Error Analysis

finite-precision program with probabilistic inputs

\[ < s_{ijk}, P_{ijk} > \]

error distribution:

\[-x*y - 2*y*z - x - z\]

cumulative prob.

abs. error
Probability Distribution of Errors

finite-precision program with probabilistic inputs

probabilistic interval subdivision
probabilistic error analysis

...
Refined Error Bounds

finite-precision program with probabilistic inputs

probabilistic interval subdivision
probabilistic error analysis

error metric extraction

threshold = 0.85
Refined Error Bounds

- finite-precision program with probabilistic inputs
- probabilistic interval subdivision
- probabilistic error analysis
- error metric extraction
- threshold = 0.85
- refined error, probability

Graph showing cumulative probability with a threshold of 0.85 and worst-case refined error.
Summary of Results: Probabilistic Error Analysis

threshold probability: 0.85, 32-bit floating-point error

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#inputs</th>
<th>#arith-ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1 - 9</td>
<td>4 - 25</td>
</tr>
</tbody>
</table>
Summary of Results: Probabilistic Error Analysis

Threshold probability: 0.85, 32-bit floating-point error

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#inputs</th>
<th>#arith-ops</th>
<th>error reduction (%) from worst-case to the largest frequent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>average</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>gaussian</td>
</tr>
<tr>
<td>25</td>
<td>1 - 9</td>
<td>4 - 25</td>
<td>17.0</td>
</tr>
</tbody>
</table>
**Summary of Results: Probabilistic Error Analysis**

threshold probability: \(0.85\), 32-bit floating-point error

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#inputs</th>
<th>#arith-ops</th>
<th>average</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1 – 9</td>
<td>4 – 25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>gaussian</th>
<th>uniform</th>
<th>gaussian</th>
<th>uniform</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>17.0</td>
<td>16.2</td>
<td>48.9</td>
<td>45.1</td>
</tr>
</tbody>
</table>

Reductions up to 73.1%

with approximate hardware specifications!
Takeaways

• Not all applications need worst-case guarantees

• Providing bounds on most frequent errors can be resource-efficient

• An automated probabilistic error analyzer: PrAn
  - strikes a balance between accuracy and complexity
  - handles different distributions, dependencies, and thresholds
Today's Talk: Probabilistic Error Analysis and NN Quantization

Accuracy Analysis:
- Probabilistic Analysis
  - iFM '19
  - EMSOFT '18
  - TACAS '21
- Static + Dynamic Analysis

Optimization:
- NN Quantization
  - EMSOFT '23
- Static + Dynamic Analysis

Tools:
- Daisy
- FLUCTUAT
- Rosa
- FPTaylor
- PRECiSA
- ...

Worst-case error analysis for small programs
Worst-case tuning for small (floating-point) programs
How do we generate quantized implementations for neural networks that meet specified worst-case error bounds?
Neural networks are ubiquitous in safety-critical systems!

Adaptive Cruise Control

Collision Avoidance System

Unicycle Controller

Drone Controller
Neural Networks as Controllers

Unicycle Controller

controller
Neural Networks as Controllers

```python
def UnicycleController(in: Vector): Vector = {
    weights1 = Matrix(...)
    weights2 = Matrix(...)
    bias1 = Vector(...)
    bias2 = Vector(...)
    x1 = relu(weights1 * in + bias1)
    out = linear(weights2 * x1 + bias2)
    return out
}
```

feed-forward regression models
Models are trained in High-Precision

\[ x_1 = \text{relu}(W_1 \times [\text{in}] + [b_1]) \]

\[ \text{out} = \text{linear}(W_2 \times [x_1] + [b_2]) \]
Models are trained in High-Precision

\[ x_1 = \text{relu}(W_1 \cdot \text{in} + b_1) \]

\[ \text{out} = \text{linear}(W_2 \cdot x_1 + b_2) \]
Models are trained in High-Precision

\[ x_1 = \text{relu}(W_1 \cdot \text{in} + b_1) \]

\[ \text{out} = \text{linear}(W_2 \cdot x_1 + b_2) \]

Input data → training

64-bit floating-point
Models are trained in High-Precision

\[ x_1 = \text{relu}(W_1 \ast \text{in} + b_1) \]

\[ \text{out} = \text{linear}(W_2 \ast x_1 + b_2) \]

In real-valued arithmetic

\[
\begin{bmatrix}
-5.23724322e-03 & \cdots & 1.30853499e-04 \\
-7.29779880e-01 & \cdots & -2.27958648e-04 \\
\end{bmatrix}
\]

64-bit floating-point

Input data → training

High-precision GPU

+/− error
Model Deployment requires Quantization

input data → training → high-precision GPU

Model Deployment
quantization
fixed low precision system

+/- error
Model Deployment requires Quantization

We need to quantize respecting the error bound!
Sound Mixed Fixed-Point Quantization

Unicycle Controller

-0.6 \leq \text{in1} \leq 9.55
-4.5 \leq \text{in2} \leq 0.2
-0.06 \leq \text{in3} \leq 2.11
-0.3 \leq \text{in4} \leq 1.51

res +/- 1e-3

mixed precision fixed-point code

directly synthesized

XILINX
State-of-the-art is not enough!

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

no fixed-point support!

FPTuner

Daisy

- not scalable
- needs unrolled structures
- over-approximates a lot

mixed precision
fixed-point code

directly synthesized

res +/- 1e-3
State-of-the-art is not enough!

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

Our Contribution: Sound Scalable Quantizer for NNs
Key Idea: Quantization for efficiency is an optimization problem!

\[ \text{minimize: } \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \]

- integer-valued cost

res +/- 1e-3

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]
Key Idea: Quantization for efficiency is an optimization problem!

\[
\begin{align*}
\text{minimize: } & \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \\
\text{subject to: } & \epsilon_n \leq \epsilon_{\text{target}}
\end{align*}
\]

- integer-valued cost
- real-valued error constraint

\(-0.6 \leq \text{in1} \leq 9.55\)
\(-4.5 \leq \text{in2} \leq 0.2\)
\(-0.06 \leq \text{in3} \leq 2.11\)
\(-0.3 \leq \text{in4} \leq 1.51\)
Key Idea: Quantization for efficiency is an optimization problem!

\[
\begin{align*}
\text{minimize:} & \quad \gamma = \sum_{i=1}^{n} \gamma_i^\text{dot} + \gamma_i^\text{bias} + \gamma_i^\alpha \\
\text{subject to:} & \quad \epsilon_n \leq \epsilon_{\text{target}} \\
& \quad I_i^{op} \geq \text{intBits} \left( R_i^{op} + \epsilon_i \right) \\
\end{align*}
\]

- integer-valued cost
- real-valued error constraint
- integer-valued range constraint

\[-0.6 \leq \text{in1} \leq 9.55 \]
\[-4.5 \leq \text{in2} \leq 0.2 \]
\[-0.06 \leq \text{in3} \leq 2.11 \]
\[-0.3 \leq \text{in4} \leq 1.51 \]
minimize: \( \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \)

subject to:

\( \varepsilon_n \leq \varepsilon_{\text{target}} \)

\( I_i^{\text{op}} \geq \text{intBits} \left( R_i^{\text{op}} + \varepsilon_i \right) \)
minimize: $\gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha}$

subject to:

$\epsilon_{n} \leq \epsilon_{\text{target}}$

$I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_{i} \right)$
Sound Mixed Fixed-Point Quantization

mixed-integer problem

$$\text{minimize: } \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha}$$

subject to:

$$\epsilon_n \leq \epsilon_{\text{target}}$$

$$I_i^{\text{op}} \geq \text{intBits} \left( R_i^{\text{op}} + \epsilon_i \right)$$

mixed-integer non-linear hard problem!

Our Solution: Reduce to Mixed Integer Linear Programming (MILP) Problem!
def UnicycleController(in: Vector): Vector = {
    weights1 = Matrix[...]
    weights2 = Matrix[...]
    bias1 = Vector(...)
    bias2 = Vector(...)
    x1 = relu(weights1 * in + bias1)
    out = linear(weights2 * x1 + bias2)
    return out
} ensuring (res +/- 1e-3)

high-level model
Aster: Sound Quantizer for NNs

def UnicycleController(in: Vector): Vector = {
    weights1 = Matrix[...]
    weights2 = Matrix[...]
    bias1 = Vector(...)
    bias2 = Vector(...)
    x1 = relu(weights1 * in + bias1)
    out = linear(weights2 * x1 + bias2)
    return out
}

ensuring (res +/- 1e-3)

mixed-precision fixed-point code

#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>
#include <ap_fixed.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2,
         ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_bias_0 = (layer2_dot_0 + (ap_fixed<27,1>) (bias2_0));
    ap_fixed<27,8> layer2_bias_1 = (layer2_dot_1 + (ap_fixed<27,1>) (bias2_1));
    ap_fixed<27,8> layer2_0 = (layer2_bias_0);
    ap_fixed<27,8> layer2_1 = (layer2_bias_1);
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}
Summary of Results: Mixed Fixed-Point Quantization of NNs

target error: $10^{-3}$, max precision: 32-bit, TO: 5 hours

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#params</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daisy</td>
<td>Aster</td>
<td></td>
</tr>
<tr>
<td>mid-sized</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(14)</td>
<td>60 - 3920</td>
<td>4s - 2h 46m 20s</td>
</tr>
<tr>
<td>large</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td>12K - 44.5K</td>
<td>12m 7s - 3h 49m 31s</td>
</tr>
</tbody>
</table>
Summary of Results: Mixed Fixed-Point Quantization of NNs

target error: 1e-3, max precision: 32-bit, TO: 5 hours

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#params</th>
<th>analysis time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Daisy</td>
</tr>
<tr>
<td>mid-sized</td>
<td>60 - 3920</td>
<td>4s - 2h 46m 20s</td>
</tr>
<tr>
<td>(14)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>large</td>
<td>12K - 44.5K</td>
<td>TO</td>
</tr>
<tr>
<td>(4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Summary of Results: Mixed Fixed-Point Quantization of NNs

Target error: $1e^{-3}$, max precision: 32-bit, TO: 5 hours

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#params</th>
<th>analysis time Daisy</th>
<th>Aster</th>
<th>latency (clock-cycles) Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>mid-sized</td>
<td>60 – 3920</td>
<td>4s – 2h 46m 20s</td>
<td>2s – 50s</td>
<td>12 – 178</td>
<td>12 – 27</td>
</tr>
<tr>
<td>large</td>
<td>12K – 44.5K</td>
<td>TO</td>
<td>12m 7s – 3h 49m 31s</td>
<td>TO</td>
<td>8K – 13K</td>
</tr>
</tbody>
</table>
## Summary of Results: Mixed Fixed-Point Quantization of NNs

Target error: $10^{-3}$, max precision: 32-bit, **TO**: 5 hours

<table>
<thead>
<tr>
<th>#benchmarks</th>
<th>#params</th>
<th>analysis time Daisy</th>
<th>analysis time Aster</th>
<th>latency (clock-cycles) Daisy</th>
<th>latency (clock-cycles) Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>mid-sized</td>
<td>60 - 3920</td>
<td>4s - 2h 46m 20s</td>
<td>2s - 50s</td>
<td>12 - 178</td>
<td>12 - 27</td>
</tr>
<tr>
<td>(14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>large</td>
<td>12K - 44.5K</td>
<td>TO</td>
<td>12m 7s - 3h 49m 31s</td>
<td>TO</td>
<td>8K - 13K</td>
</tr>
<tr>
<td>(4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Aster is more precise than Daisy — Daisy reports 5 infeasibility, Aster reports 3!
Takeaways

• Specializing optimization in application contexts can be beneficial
• Optimization with linearizations and abstractions is effective for NNs
• An automated NN quantizer: Aster
  - generates sound quantized code that can be directly synthesized in Xilinx
  - is precise and scalable
Expanding the Horizons of Finite-Precision Analysis

Accuracy Analysis
- Probabilistic Analysis
- Static + Dynamic Analysis
- Thesis Contributions
- TACAS '21
- EMSOFT '19

Optimization
- NN Quantization
- EMSOFT '23

Thesis Contributions
- iFM '19
- EMSOFT '18
- TACAS '21

Worst-case error analysis for small programs
- Daisy
- FLUCTUAT
- Rosa
- FPTaylor
- PRECiSA
- ...
Future Research Directions

- Scalable Accuracy Analysis
  - considering probabilistic inputs
  - by combining static, dynamic analysis and machine learning techniques
Future Research Directions

• Scalable Accuracy Analysis
  - considering probabilistic inputs
  - by combining static, dynamic analysis and machine learning techniques

• Scalable Optimization
  - considering probabilistic inputs
  - specialize in other application contexts
Future Research Directions

• Scalable Accuracy Analysis
  - considering probabilistic inputs
  - by combining static, dynamic analysis and machine learning techniques

• Scalable Optimization
  - considering probabilistic inputs
  - specialize in other application contexts

• Finite-precision in the context of
  - heterogeneous HPC systems

... and others!
Collaborators

Eva Darulova
Sylvie Putot
Eric Goubault
Milos Prokop
Joshua Sobel
Clothilde Jeangoudoux
Maria Christakis
Anastasia Volkova
Thank You
For Your Attention!