Floating-Point Mixed Precision Tuning
by Static Analysis
FEANICSES 2019

Dorra Ben Khalifa
Supervisors : Matthieu Martel & Assalé Adjé

University of Perpignan

21 juin 2019
Energy Consumption Concern

- Top 500 supercomputers energy consumption ≈ $400 million/year
- How to increase energy efficiency?
  - Green Computing: http://www.green500.org
  - Reduce application energy consumption
  - Sacrifice accuracy for performance ⇒ Floating-point precision tuning
What About...

- Computer architectures support multiple levels of precision
  - **Higher precision**: improves accuracy
  - **Lower precision**: reduces energy, running time and bandwidth capacity

- Automatically tune floating-point precision is challenging
  - Without affecting correctness
  - Improving performance

**Precision vs Accuracy!**

- **Precision**: number of bits representing a value (its format)
- **Accuracy**: how close a floating-point computation comes to the real value!
Related Work

- **TWIST**
  
  *Static analysis by constraints generation*

  TWIST [3]

- **CRAFT, Precimonious/HiFPTuner**
  
  *Search based methods*

  CRAFT [Lam’13 et al.], Precimonious/HiFPTuner [Rubio’13 et al.] [2]

- **FPTuner, Rosa/Daisy**
  
  *Rigorous error analysis methods*

  FPTuner [Chiang’17 et al.], Rosa/Daisy [Darulova’14 et al.]

- **Herbie, Salsa**
  
  *Automatically discovering unstable floating-point operations and applying transformations*

  Herbie [Panchekha’14 et al.], Salsa [DM18]
Overview of our Approach

Oversized Precision Program

User specified accuracy

Forward & Backward analysis

Static analysis by constraints generation

Final precision requirement

Tuned optimized program(s)

Output
Outline

1 Preliminary
2 Forward & Backward Static Analysis
3 Groundwork on Constraints
4 Preliminary Results
5 Future Studies
Outline

1. Preliminary
2. Forward & Backward Static Analysis
3. Groundwork on Constraints
4. Preliminary Results
5. Future Studies
Basic Concepts on Floating-Point Numbers

- **A floating-point number** $x$ in base $\beta$:
  
  $$x = s.m.\beta^{e-p+1}$$

- $s$ the sign, $m$ the mantissa, $e$ the exponent encoded in the bit string and $p$ is the format precision

- **IEEE-754 Formats**

<table>
<thead>
<tr>
<th>format</th>
<th>bit width</th>
<th>mantissa size $(p - 1)$</th>
<th>exponent size</th>
<th>bias</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary16</td>
<td>16</td>
<td>10</td>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>binary32</td>
<td>32</td>
<td>23</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>binary64</td>
<td>64</td>
<td>52</td>
<td>11</td>
<td>1023</td>
</tr>
<tr>
<td>binary128</td>
<td>128</td>
<td>112</td>
<td>15</td>
<td>16383</td>
</tr>
</tbody>
</table>
**Ufp and Ulp Functions**

Weight of the most significant bit:

\[ ufp(x) = \min\{i \in \mathbb{N} : 2^{i+1} > x\} = \lfloor \log_2(x) \rfloor \]

Weight of the least significant bit:

\[ ulp(x) = \begin{cases} 
  e - p & \text{round to nearest,} \\
  e + 1 - p & \text{otherwise.} 
\end{cases} \]

\( \mathbb{F}_p : \text{Set of floating point numbers} : |v - \hat{v}| \leq 2^{e-p+1} \)

\[ \forall x \in \mathbb{F}_p, \quad ulp(x) = ufp(x) - p + 1 \]

Error on \( x \):

\[ \epsilon(x) \leq 2^{ulp(x)} \]
Outline

1. Preliminary
2. Forward & Backward Static Analysis
3. Groundwork on Constraints
4. Preliminary Results
5. Future Studies
Forward & backward Analysis for Arithmetic Expressions

\[
x + y = r
\]

Abstract Domain
Concrete Addition in $\mathbb{F}_p$
Abstract Addition in $\mathbb{I}_p$
Concrete Multiplication in $\mathbb{F}_p$
Forward & backward Analysis for Arithmetic Expressions

User requirement

x + y forward r

Concrete Addition in $\mathbb{F}_p$
Abstract Addition in $\mathbb{I}_p$
Concrete Multiplication in $\mathbb{F}_p$
Forward & backward Analysis for Arithmetic Expressions

Generalizable technique into sets of values!
Abstract Domain

- **Abstract Values**: \([a, b]_p\) interval of \(\mathbb{F}_p\)

  e.g.: \(x, y \in [1.0, 3.0]_{16}, |v - \hat{v}| \leq 2^{ufp(x)-15}\)

- **Concretization function**:

  \[
  \gamma([a, b]_p) = x \in \mathbb{F}_p : a \leq x \leq b
  \]

- **Partial order**:

  \([a, b]_p \sqsubseteq [c, d]_q \iff [a, b] \subseteq [c, d] \land q \leq p\)

  \([a, b]_p\) is more precise than \([c, d]_q\) with a greater accuracy
Concrete Addition in $\mathbb{F}_p$

**Forward addition**: $p' : \text{size of } \epsilon_x, q' : \text{size of } \epsilon_y$

$$\oplus(x_{p'}, y_{q'}) = z_{r'}, \quad \text{with} \quad r = \text{ufp}(x + y) - \text{ufp}(\epsilon(x) + \epsilon(y))$$

**Backward addition**:

$$\ominus(z_{r'}, y_{q'}) = (z - y)_{p'}, \quad \text{avec} \quad p = \text{ufp}(z - y) - \text{ufp}(\epsilon(z) - \epsilon(y))$$
Abstract Addition in $\mathbb{I}_p$

\[ \overrightarrow{\oplus}([1.0, 3.0]_{16}, [1.0, 3.0]_{16}) = [2.0, 6.0]_{16} \]

\[ \overrightarrow{\oplus}([2.0, 6.0]_{10}, [1.0, 3.0]_{16}) = [1.0, 3.0]_{9} \]
Concrete Multiplication in $\mathbb{F}_p$

- **Forward multiplication**:
  \[ \overrightarrow{(x_{p'}, y_{q'})} = z_{r'} \quad \text{where} \quad r = \text{ufp}(x \times y) - \text{ufp}(\epsilon(x \times y)) \]
  \[ \quad \text{and} \quad \text{ufp}(\epsilon(x \times y)) = y.\epsilon(x) + x.\epsilon(y) + \epsilon(x).\epsilon(y) \]

- **Backward multiplication**:
  \[ \overleftarrow{(z_{r'}, y_{q'})} = (z \div y)_{r'} \quad \text{where} \quad p = \text{ufp}(z \div y) - \text{ufp}\left(\frac{y.\epsilon(z_r) - z.\epsilon(y_q)}{y.(y + \epsilon(y_q))}\right) \]

**Note**

Problem reduced to a system of constraints made of linear relations between integer elements only
Introduction
Preliminary
Forward & Backward Static Analysis
Groundwork on Constraints
Preliminary Results
Future Studies

Strategy
Systematic Constraint Generation

Outline

1. Preliminary
2. Forward & Backward Static Analysis
3. Groundwork on Constraints
4. Preliminary Results
5. Future Studies
Before Constraint Generation....

- Preliminary range measure by static analysis (no overflow)
- The accuracy may $\searrow$ in forward analysis $\rightsquigarrow$ Weaken the pre-conditions
- The accuracy may $\nearrow$ in backward analysis $\rightsquigarrow$ Strengthen the post-conditions

$$z = x \odot y \quad \text{with} \quad \odot \in \{+, -, \times, /\}$$

$$\text{lower}(Acc_B(z)) = \begin{cases} 
\text{lower } Acc_B(x) \text{ in order to lower } Acc_B(z) \\
\text{lower } Acc_B(y) \text{ in order to lower } Acc_B(z) \\
\text{lower both } Acc_B(x) \text{ and } Acc_B(y)
\end{cases}$$
Systematic Constraint Generation

**Expression** : \( e := c \uparrow p \ell | id \ell | e_{1}^{\ell_1} + \ell e_{2}^{\ell_2} | e_{1}^{\ell_1} - \ell e_{2}^{\ell_2} | e_{1}^{\ell_1} \times \ell e_{2}^{\ell_2} | e_{1}^{\ell_1} \div \ell e_{2}^{\ell_2} \)

**Boolean** : \( b := \text{true} | \text{false} | e_{1}^{\ell_1} < \ell e_{2}^{\ell_2} | e_{1}^{\ell_1} > \ell e_{2}^{\ell_2} | e_{1}^{\ell_1} = \ell e_{2}^{\ell_2} \)

**Statement** : \( c := c_{1}^{\ell_1} | c_{2}^{\ell_2} | id = \ell e_{1}^{\ell_1} | \text{while} \ell b^{\ell_0} \text{do} c_{1}^{\ell_1} | \text{if} \ell b^{\ell_0} \text{then} c_{1}^{\ell_1} \text{else} c | \text{require_accuracy}(x,n) \)

- \( l \in Lab \) unique label is attached to each expression and statement
- \( \Lambda : Id \rightarrow Id \times Lab, x = \ell e_{1}^{\ell_1} \)
- We assign to each label \( l \) three variables : \( acc_B(l), acc_F(l) \) and \( acc(l) \)

\[ 0 \leq acc_B(l) \leq acc(l) \leq acc_F(l) \]
Case of the Forward Addition (1/2)

\[ a = \text{ufp}(x) \quad b = \text{ufp}(y) \]
\[ \epsilon(x) \leq 2^{a-p+1} \quad \epsilon(y) \leq 2^{b-p+1} \quad \epsilon_+ < 2^{a-p+1} + 2^{b-p+1} \]

**Definition:**

\[ \iota(\epsilon(x), \epsilon(y)) = \begin{cases} 0 & \text{if ulp}(\epsilon(x)) > \text{ufp}(\epsilon(y)) \\ 1 & \text{otherwise}. \end{cases} \]

**Lemma 1:**

\[ \text{ufp}(\epsilon_+) \leq \max(a - p, b - q) + \iota(a - p, b - q) \]

\[ r_+ = \text{ufp}(x + y) - \max(a - p, b - q) - \iota(a - p, b - q) \]
**Case of the Forward Addition (2/2)**

\[ A = \text{ufp}(\epsilon_x) \quad B = \text{ufp}(\epsilon_y) \quad C = \text{ufp}(\epsilon_z) \]

- **How to compute** \( r' = \text{ufp}(\epsilon(z)) - ulp(\epsilon(z)) \)?

- We have:
  \[ U = \text{ufp}(\epsilon_z) \quad \text{and} \quad u = ulp(\epsilon_z) \]
  \[ U = \text{ufp}(z) - R \]

\[ u = \min \left\{ \begin{array}{c} 
\text{ufp}(x) - p - p' + 1 \\
\text{ufp}(y') - q - q' + 1 
\end{array} \right\} \]
Case of the Forward Multiplication

\[ \epsilon(x) \leq 2^{a-p+1}, \epsilon(y) \leq 2^{b-p+1} \]

\[ ufp(\epsilon_x) \leq 2^{a+1} \cdot 2^{b-q+1} + 2^{b+1} \cdot 2^{a-p+1} + 2^{a-p+1} \cdot 2^{b-q+1} \]
\[ = 2^{a+b-q+2} + 2^{a+b-p+2} + 2^{a+b-p-q+2} \]

\[ ufp(\epsilon_x) \leq \max(a + b - p + 2, a + b - q + 2) + \iota(p, q) \]
\[ \leq \max(a + b - p + 1, a + b - q + 1) + \iota(p, q) \]

Thus:

\[ r_x = ufp(x \times y) - \max(a + b - p + 1, a + b - q + 1) - \iota(p, q) \]

**Linear constraints!**
Outline

1 Preliminary
2 Forward & Backward Static Analysis
3 Groundwork on Constraints
4 Preliminary Results
5 Future Studies
Syntax of IMP

Expression: \( e : := \text{constant} \mid \text{id} \mid e + e \mid e - e \mid e \times e \mid e \div e \)

Boolean: \( b : := \text{true} \mid \text{false} \mid e < e \mid e > e \mid e \leq e \mid e \geq e \mid e = e \)

Statement: \( c : := c ; c \mid \text{id} = e \mid \text{while} \ b \ \text{do} \ c \mid \text{if} \ b \ \text{then} \ c \ \text{else} \ c \)

- **Work Environment**:
  - Java SE Development Kit 8
  - Eclipse IDE Java Oxygen.2 Release (4.7.2)
  - ANTLR4 IDE Eclipse Plugin for ANTLR 4

1. https://github.com/antlr
Example (1/4) : Parsing Tree

```plaintext
x = [1.0, 2.0];
y = [3.0, 4.0];
z = x + y;
require_accuracy(z, 25);
```
Example (2/4): Constraints Semantic

\[
\begin{align*}
\varepsilon[\upsilon_{c_{l_0}^{l_0}}] \Lambda &= \{ acc_{F}(l_0) = 53 \} \\
\varepsilon[\upsilon_{c_{l_2}^{l_2}}] \Lambda &= \{ acc_{F}(l_2) = 53 \} \\
\varepsilon[x^{l_4} + y^{l_5}] \Lambda &= C[x^{l_4}] \Lambda \cup C[y^{l_5}] \Lambda \cup F_{+}(l_4, l_5, l_6) \cup B_{+}(l_4, l_5, l_6) \\
C[z := l_7 x + y^{l_6}] \Lambda &= (C, \Lambda[z \rightarrow z^{l_7}]) \\
C[require\_accuracy(z, 25)^{l_0}] \Lambda &= \{ acc_{B}(\Lambda(z)) = 25 \} 
\end{align*}
\]
Example (3/4) : Constraints Generation (Program.z3)

```prolog
(assert (\< acc_y_lab3 accf_y_lab3))
(assert (= accf_lab0 0))
(assert (= accf_y_lab1 accf_lab0))
(assert (= accf_y_lab3 accf_lab2))
(assert (= accf_lab4 0))
(assert (= accf_lab5 0))
(assert (= rSup_lab6 (- 2 (ite (> (- 1 accf_lab4) (- 2 accf_lab5)) (- 1 accf_lab4) (- 2 accf_lab5)))))
(assert (= rInf_lab6 (- 2 (ite (< (- 0 accf_lab4) (- 1 accf_lab5)) (- 0 accf_lab4) (- 1 accf_lab5)))))
(assert (= ioSup_lab6 (ite (> 2 1) 0)))
(assert (= propaUlpSUP_lab6 (ite (< ulpULsup_lab4 ulpL2sup_lab5) ulpL1sup_lab4 ulpL2sup_lab5)))
(assert (= ioInf_lab6 (ite (> 0 1) 0)))
(assert (= accf_lab6 (ite (< (- rInf_lab6 ioInf_lab6) (- rSup_lab6 ioSup_lab6)) (- rInf_lab6 ioInf_lab6) (- rSup_lab6 ioSup_lab6))))
(assert (= accf_z_lab7 accf_lab6))
(assert (= accb_x_lab1 accb_lab0))
(assert (= accb_y_lab3 accb_lab2))
(assert (= rSup_lab6 (- 2 (ite (> (- 1 accf_lab4) (- 2 accf_lab5)) (- 1 accf_lab4) (- 2 accf_lab5)))))
(assert (= rInf_lab6 (- 2 (ite (< (- 0 accf_lab4) (- 1 accf_lab5)) (- 0 accf_lab4) (- 1 accf_lab5)))))
(assert (= ioSup_lab6 (ite (> 2 1) 0)))
(assert (= propaUlpSUP_lab6 (ite (< ulpULsup_lab4 ulpL2sup_lab5) ulpL1sup_lab4 ulpL2sup_lab5)))
(assert (= ioInf_lab6 (ite (> 0 1) 0)))
(assert (= accf_lab6 (ite (< (- rInf_lab6 ioInf_lab6) (- rSup_lab6 ioSup_lab6)) (- rInf_lab6 ioInf_lab6) (- rSup_lab6 ioSup_lab6))))
(assert (= accb_x_lab1 accb_lab0))
(assert (= accf_y_lab3 accf_lab2))
(assert (= accf_z_lab7 accf_lab6))
(assert (= accb_z_lab7 accb_lab6))
(assert (= accb_z_lab9 25))
```
Example (4/4) : Z3 SMT solver solution

\[
\begin{align*}
    x^{25} &= [1.0,2.0]^{25}; \\
    y^{24} &= [3.0,4.0]^{24}; \\
    z^{25} &= [1.0,2.0]^{25} + [25] [3.0,4.0]^{24}; \\
    \text{require\_accuracy}(z,25);
\end{align*}
\]

Cost function

Solutions are not unique. We need to add an additional constraint related to a cost function \( \phi \) to the constraints

\[
\phi(c) = \sum_{x \in Id, l \in Lab} acc(x^l) + \sum_{l \in Lab} acc(l)
\]
Outline

1. Preliminary
2. Forward & Backward Static Analysis
3. Groundwork on Constraints
4. Preliminary Results
5. Future Studies
Policy Iteration

**Motivation**
- Z3 SMT solver = decision tool ≠ optimization tool

**Idea**
- Using Policy iteration to improve accuracy [1]
- Generated constraints are of the form **min-max of discrete affine maps**
- Feeding the policy iteration with the Z3 solution as an initial policy!

**Finality**
- Comparing the policy iteration and Z3 solutions (in term of execution time and optimality)
Conclusion

- Floating-point computations determination minimal precision

**Contribution**

- Forward & Backward static analysis for numerical accuracy
- Formulation as first order linear constraints

**Extensions**: functions, arrays, fixed-point arithmetic, etc.

**Minimal precision determination**: Policy Iteration

**Experimentally tool validation**: embedded systems, numerical computation, etc.
References

Stephane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zennou.
Static analysis by policy iteration on relational domains.

Stef Graillat, Fabienne Jézéquel, Romain Picot, François Févotte, and Bruno Lathuilière.
PROMISE : floating-point precision tuning with stochastic arithmetic.

Matthieu Martel.
Floating-point format inference in mixed-precision.
THANK YOU FOR LISTENING

Q & A!