University of Perpignan Via Domitia
LAMPS Laboratory

Fast and Efficient Bit-Level Precision Tuning

Ph.D. Defense

Dorra BEN KHALIFA

Jury Members

Directors: Matthieu MARTEL
Assalé ADJE

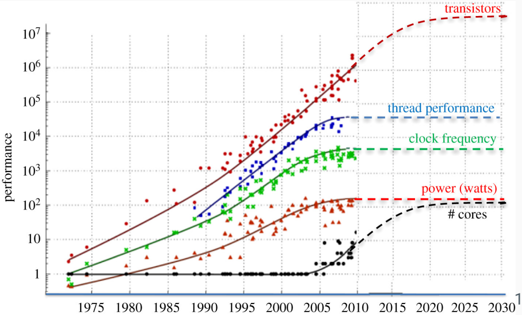
Reviewers: Ganesh GOPALAKRISHNAN
Eric GOUBAULT
David MONNIAUX

Examiners: Eva DARULOVA
Philippe LANGLOIS
Laura TITOLO

November 29, 2021

Introduction

- SW used to solve more & more complex tasks
- Thanks to HW whose performace double every 18 months $\Longrightarrow$ **Moore's law in** 1965

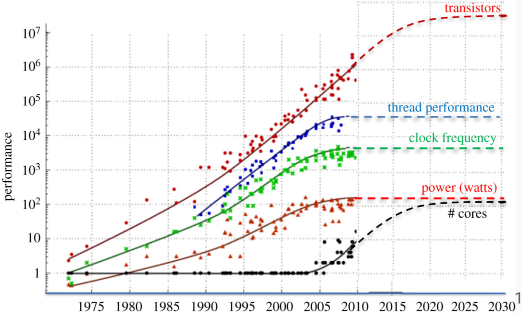


[1]

[1] Photo credits from https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0061

Introduction

- SW used to solve more & more complex tasks
- Thanks to HW whose performance double every 18 months ⟹ **Moore's law in** 1965



[1]

Issue

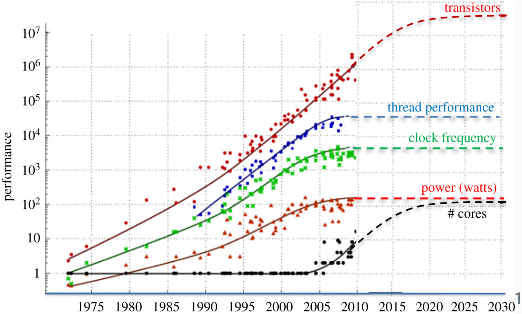
☞ Less perspectives to further performance improvements after 2025

☞ End of Moore's law

[1] Photo credits from https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0061

Introduction

- SW used to solve more & more complex tasks
- Thanks to HW whose performance double every 18 months ⟹ **Moore's law in** 1965



[1]

Issue

☞ Less perspectives to further performance improvements after 2025

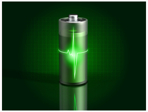
☞ End of Moore's law

☞ What is the future of computing?

[1] Photo credits from https://royalsocietypublishing.org/doi/10.1098/rsta.2019.0061

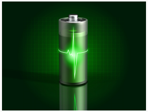
Motivation

- Developers use **highest precision** available (e.g. 64 bits, 128 bits) $\longrightarrow$ increases running time, memory footprint, energy consumption



Motivation

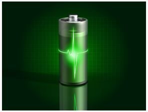
- Developers use **highest precision** available (e.g. 64 bits, 128 bits) $\longrightarrow$ increases running time, memory footprint, energy consumption

- Sometimes variables in **lower precision** (e.g. 16 bits, 32 bits) is enough

Motivation

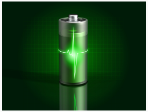
- Developers use **highest precision** available (e.g. 64 bits, 128 bits) $\longrightarrow$ increases running time, memory footprint, energy consumption



- Sometimes variables in **lower precision** (e.g. 16 bits, 32 bits) is enough
- Manually changing for optimized precision is fastidious

Motivation

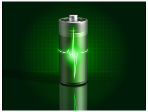
- Developers use **highest precision** available (e.g. 64 bits, 128 bits) ⟶ increases running time, memory footprint, energy consumption



- Sometimes variables in **lower precision** (e.g. 16 bits, 32 bits) is enough
- Manually changing for optimized precision is fastidious
- **Precision Tuning technique**
  - ☞ Not a Find-and-Replace button like in a text editor
  - ☞ More complex task with program semantics analysis

Motivation

- Developers use **highest precision** available (e.g. 64 bits, 128 bits) $\longrightarrow$ increases running time, memory footprint, energy consumption

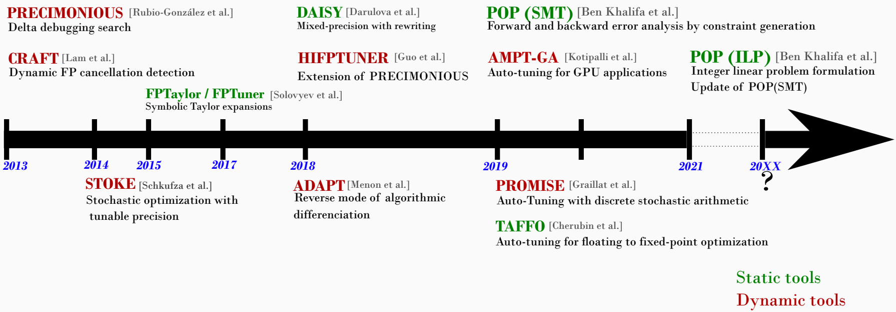


- Sometimes variables in **lower precision** (e.g. 16 bits, 32 bits) is enough
- Manually changing for optimized precision is fastidious
- **Precision Tuning technique**
  - ☞ Not a Find-and-Replace button like in a text editor
  - ☞ More complex task with program semantics analysis

Goal

- ☞ Develop automated techniques for precision tuning
- ☞ Best trade-offs between performance & user requirements

Precision Tuning A Survey[2]

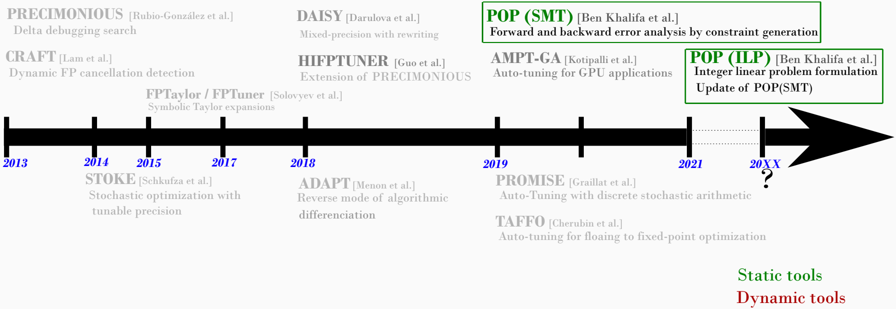


Weaknesses:

- Trial-and-Error strategy
- Search algorithm-based tools (CRAFT, PRECIMONIOUS, . . .)
- Static tools limited to straight-line programs (Daisy, FPTUNER, . . .) or uniform precision (Rosa,. . .)

☞ How to go beyond?

[2] A more exhaustive survey is given in Chapter 3 of the dissertation



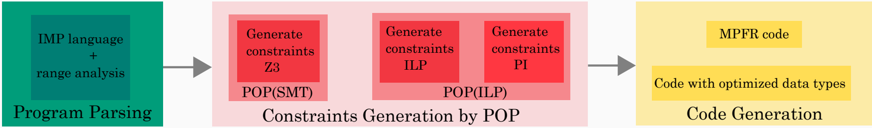
POP strengths:

- ✓ No trial-and-Error strategy
- ✓ Finds directly the minimal number of bits needed
- ✓ Complexity does not increase as number of data types increases
- ✓ Loops and conditionals

---

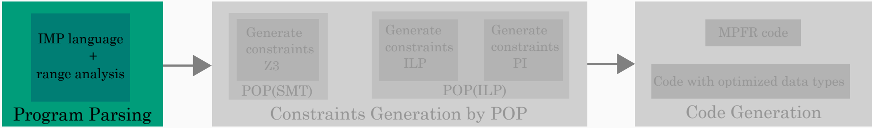
[2] A more exhaustive survey is given in Chapter 3 of the dissertation

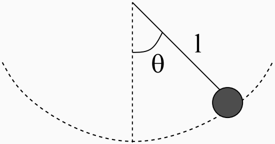
POP in One Slide



- Open-source tool
- Implementation in Java
- $\approx$ **10 000** lines of code
- ANTLR, Z3, GLPK libraries
- Bit-level, IEEE754, fixed and MPFR datatypes
- Publications [FTSCS'19, IINTEC'19, IoTaIS'20, ICICT'21, ICCSA'21, SAS'21]

Running Example From Parsing...





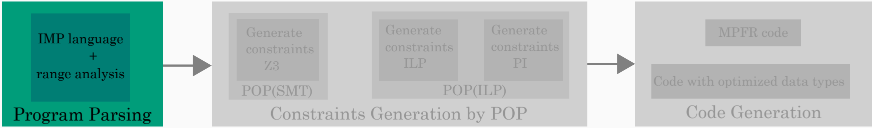
Pendulum ($\theta = \frac{\pi}{4}$)

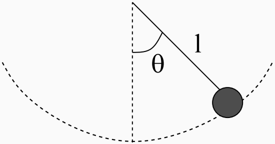
Second order differential equation

(E1): $m \cdot l \cdot \dfrac{d^2\theta}{dt^2} = -m \cdot g \cdot \sin\theta$

(E1)⇔(E2): $\dfrac{dy_1}{dt} = y_2$ and $\dfrac{dy_2}{dt} = -\dfrac{g}{l} \cdot \sin y_1$

Running Example From Parsing...





Pendulum ($\theta = \frac{\pi}{4}$)

Second order differential equation

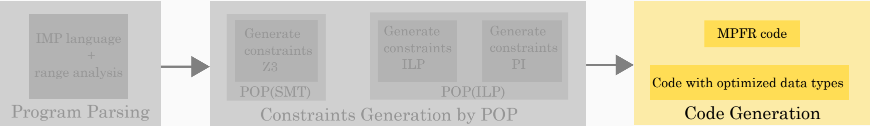
$$\textbf{(E1):}\quad m \cdot l \cdot \frac{d^2\theta}{dt^2} = -m \cdot g \cdot \sin\theta$$

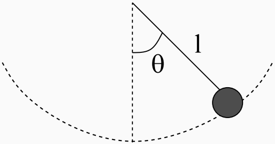
$$\textbf{(E1)}\Leftrightarrow\textbf{(E2):}\quad \frac{dy_1}{dt} = y_2 \quad \text{and} \quad \frac{dy_2}{dt} = -\frac{g}{l}\cdot \sin y_1$$

```
g^ℓ1 = 9.81^ℓ0 ; l^ℓ3 = 0.5^ℓ2 ;
y1^ℓ5 = 0.785398^ℓ4 ;
y2^ℓ7 = 0.785398^ℓ6 ;
h^ℓ9 = 0.1^ℓ8 ; t^ℓ11 = 0.0^ℓ10 ;
while (t^ℓ13 <^ℓ15 10.0^ℓ14)^ℓ59 {
  y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21 ;
  aux1^ℓ28 = sin(y1^ℓ26)^ℓ27 ;
  aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
   *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36 ;
  y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44 ;
  t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50 ;
  y1^ℓ55 = y1new^ℓ54 ;
  y2^ℓ58 = y2new^ℓ57 ; };
require_nsb(y2, 20)^ℓ61 ;
```

POP Label File

Running Example ... to Tuning





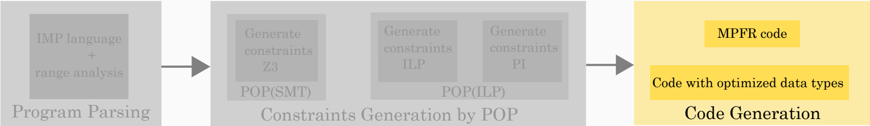
Pendulum ($\theta = \frac{\pi}{4}$)

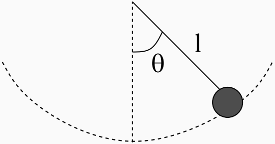
Second order differential equation

$$\textbf{(E1):}\quad m \cdot l \cdot \frac{d^2\theta}{dt^2} = -m \cdot g \cdot \sin\theta$$

$$\textbf{(E1)}\Leftrightarrow\textbf{(E2):}\quad \frac{dy_1}{dt} = y_2 \quad \text{and} \quad \frac{dy_2}{dt} = -\frac{g}{l} \cdot \sin y_1$$

Running Example ... to Tuning





Pendulum ($\theta = \frac{\pi}{4}$)

Second order differential equation

$$\textbf{(E1):}\quad m \cdot l \cdot \frac{d^2\theta}{dt^2} = -m \cdot g \cdot \sin\theta$$

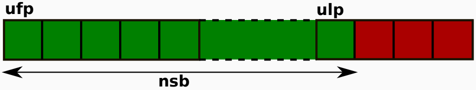
$$\textbf{(E1)}\Leftrightarrow\textbf{(E2):}\quad \frac{dy_1}{dt} = y_2 \quad \text{and} \quad \frac{dy_2}{dt} = -\frac{g}{l} \cdot \sin y_1$$

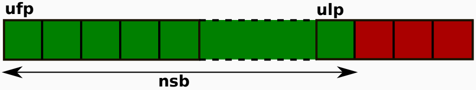
```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|21| = 0.0|21|;
h|21| = 0.1|21|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|21|*|22| h|21|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|19| *|20| h|18|*|19|g|17| /
        |18|l|17|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

POP Output File

Table of Contents

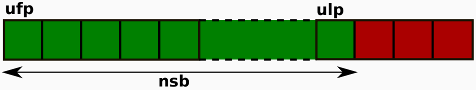
Preliminary Elements ufp, nsb, ulp





- The **unit in the first place** of a real number x

$$\mathrm{ufp}(x) = \begin{cases} \min\{i \in \mathbb{Z} : 2^{i+1} > |x|\} = \lfloor \log_2(|x|) \rfloor & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

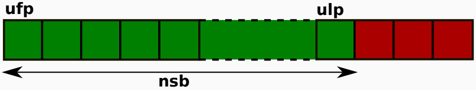


- The **unit in the first place** of a real number x

$$\mathrm{ufp}(x) = \begin{cases} \min\{i \in \mathbb{Z} : 2^{i+1} > |x|\} = \lfloor \log_2(|x|) \rfloor & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

- The **number of significant bits** nsb of x
  - $\hat{x}$: approximation of x in finite precision
  - Absolute error: $\varepsilon_x \leq 2^{\mathrm{ufp}(x) - \mathrm{nsb}(x) + 1}$

Preliminary Elements ufp, nsb, ulp



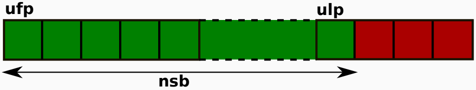
- The **unit in the first place** of a real number x

$$\mathrm{ufp}(x) = \begin{cases} \min\{i \in \mathbb{Z} : 2^{i+1} > |x|\} = \lfloor \log_2(|x|) \rfloor & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

- The **number of significant bits** nsb of x
  - $\hat{x}$: approximation of x in finite precision
  - Absolute error: $\varepsilon_x \leq 2^{\mathrm{ufp}(x)-\mathrm{nsb}(x)+1}$

- The **unit in the last place** of x

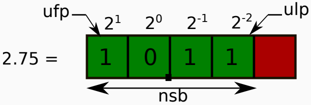
$$\mathrm{ulp}(x) = \mathrm{ufp}(x) - \mathrm{nsb}(x) + 1$$



- The **unit in the first place** of a real number x

$$\text{ufp}(x) = \begin{cases} \min\{i \in \mathbb{Z} : 2^{i+1} > |x|\} = \lfloor \log_2(|x|) \rfloor & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

- The **number of significant bits** nsb of x
  - $\hat{x}$: approximation of x in finite precision
  - Absolute error: $\varepsilon_x \leq 2^{\text{ufp}(x)-\text{nsb}(x)+1}$



- The **unit in the last place** of x

$$\text{ulp}(x) = \text{ufp}(x) - \text{nsb}(x) + 1$$

- $\text{ufp}(2.75) = 1$
- $\text{nsb}(2.75) = 4$
- $\text{ulp}(2.75) = -2$

Preliminary Elements — Range Determination

[...]

$z^{\ell_2} = x^{\ell_0} + y^{\ell_1};$

$\texttt{require\_nsb}(z,18)^{\ell_3};$

$$[\ldots]$$
$$z^{\ell_2} = x^{\ell_0} + y^{\ell_1};$$
$$\texttt{require\_nsb}(z,18)^{\ell_3};$$

- Preliminary range analysis for all the program variables

example $x \in [0.2, 0.7]$, $y \in [0.6, 0.8]$, $z \in [0.8, 1.5]$

Preliminary Elements — Range Determination

```
[...]
```

$$z^{\ell_2} = x^{\ell_0} + y^{\ell_1};$$

$$\texttt{require\_nsb}(z,18)^{\ell_3};$$

- Preliminary range analysis for all the program variables

example $x \in [0.2, 0.7]$, $y \in [0.6, 0.8]$, $z \in [0.8, 1.5]$

- Pre-computation of the ufp of each range

example $\mathrm{ufp}(0.2) = -3$, $\mathrm{ufp}(0.7) = -1 \Longrightarrow \mathrm{ufp}(x) = -1$, $ufp(y) = -1$, $ufp(z) = 0$

Preliminary Elements Range Determination

```
[...]
z^ℓ2 = x^ℓ0 + y^ℓ1;
require_nsb(z,18)^ℓ3;
```

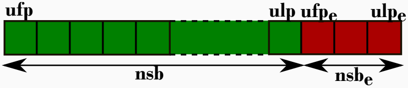
- Preliminary range analysis for all the program variables

example $x \in [0.2, 0.7]$, $y \in [0.6, 0.8]$, $z \in [0.8, 1.5]$

- Pre-computation of the ufp of each range

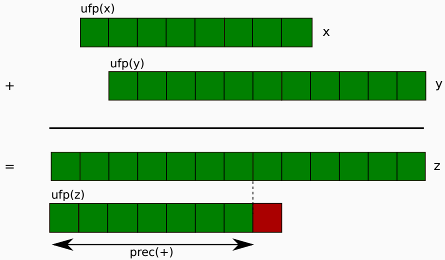
example $\mathrm{ufp}(0.2) = -3$, $\mathrm{ufp}(0.7) = -1 \Longrightarrow \mathrm{ufp}(x) = -1$, $ufp(y) = -1$, $ufp(z) = 0$

- **Goal: Computation of the minimal** nsb
  - $\mathrm{nsb}(z) = 18$ // thanks to the annotation
  - $\mathrm{nsb}(x) = ufp(x) - \mathrm{ufp}(\varepsilon(x))$
  - $\mathrm{nsb}(x) = -1$ - $\mathrm{ufp}(\varepsilon(x))$?



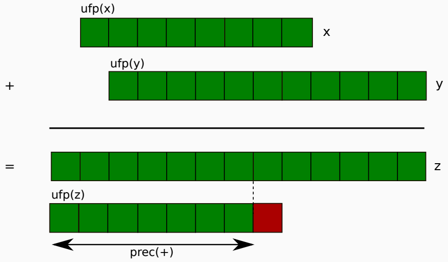
☞ How to compute errors?

Case 1: Initially values x and y are exact



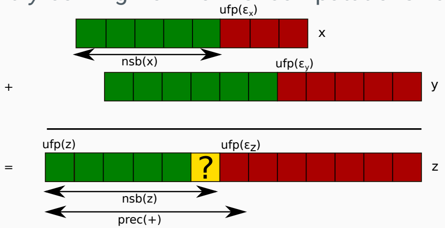
- Truncation error: $\varepsilon_+ \leq 2^{\mathrm{ufp}(z)-\mathrm{prec}(+)}$

Case 1: Initially values x and y are exact



- Truncation error: $\varepsilon_+ \leq 2^{\mathrm{ufp}(z)-\mathrm{prec}(+)}$

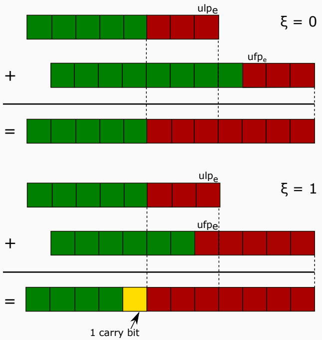
Case 2: Values x and y coming from former computations have errors



- $\mathrm{ufp}(\varepsilon_z)$ depends on carry

✗ Before this work, very costly over-approximation in [NFM'17]

- ✗ Before this work, very costly over-approximation in [NFM'17]
- ✓ Optimized carry bit function



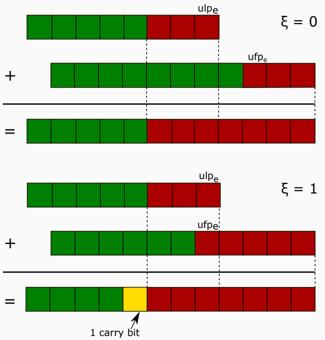
Optimized Carry Bit

Let $x^{\ell} = c_1^{\ell_1} \odot c_2^{\ell_2}$, $\ell \in Lab$ and $\odot \in \{+, -, \times, \div\}$

$$carry(\ell, \ell_1, \ell_2) = \begin{cases} 0 & \textbf{if no overlap errors} \\ 1 & \textbf{otherwise.} \end{cases}$$

✗ Before this work, very costly over-approximation in [NFM'17]

✓ Optimized carry bit function



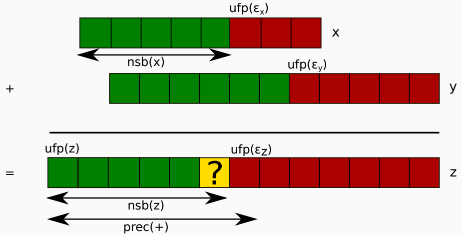
Optimized Carry Bit

Let $x^{\ell} = c_1^{\ell_1} \odot c_2^{\ell_2}$, $\ell \in Lab$ and $\odot \in \{+, -, \times, \div\}$

$$carry(\ell, \ell_1, \ell_2) = \begin{cases} 0 & \textbf{if no overlap errors} \\ 1 & \textbf{otherwise.} \end{cases}$$

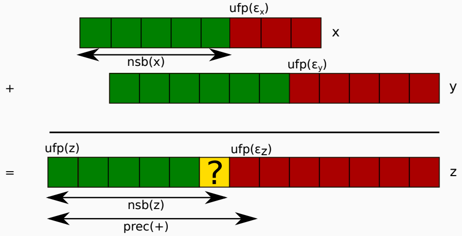
$$carry(\ell, \ell_1, \ell_2) = \min\left(\begin{array}{l} \max\big(\mathrm{ufp}(\ell_2) - \mathrm{ufp}(\ell_1) + \mathrm{nsb}(\ell_1) - \mathrm{nsb}(\ell_2) - \mathrm{nsb_e}(\ell_2), 0\big), \\ \max\big(\mathrm{ufp}(\ell_1) - \mathrm{ufp}(\ell_2) + \mathrm{nsb}(\ell_2) - \mathrm{nsb}(\ell_1) - \mathrm{nsb_e}(\ell_1), 0\big), 1 \end{array} \right)$$

Non-linear computations (min and max operators)



[3] Chapter 4 of the dissertation discusses the arithmetic expressions

Preliminary Elements — Transfer Functions



Forward addition

$$\overrightarrow{\oplus}(x, y) = z$$

$$\mathrm{nsb}(z) = \mathrm{ufp}(x + y) - \mathrm{ufp}(\varepsilon(x) + \varepsilon(y) + \varepsilon_+)$$

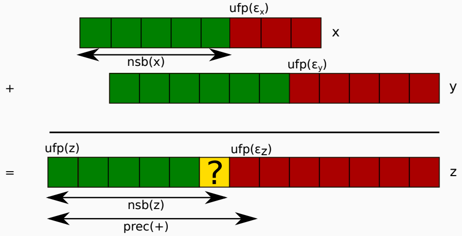
Backward addition

$$\overleftarrow{\oplus}(z, y) = (z - y)$$

$$\mathrm{nsb}(x) = \mathrm{ufp}(z - y) - \mathrm{ufp}(\varepsilon(z) - \varepsilon(y) - \varepsilon_+)$$

[3] Chapter 4 of the dissertation discusses the arithmetic expressions

Preliminary Elements — Transfer Functions



Forward addition

$$\overrightarrow{\oplus}(x, y) = z$$

$$\mathrm{nsb}(z) = \mathrm{ufp}(x + y) - \mathrm{ufp}(\varepsilon(x) + \varepsilon(y) + \varepsilon_{+})$$

Backward addition

$$\overleftarrow{\oplus}(z, y) = (z - y)$$

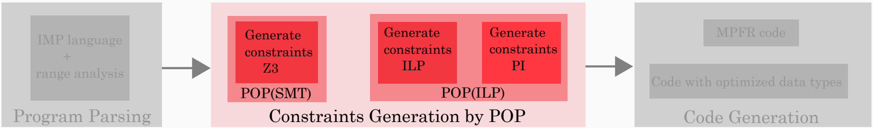
$$\mathrm{nsb}(x) = \mathrm{ufp}(z - y) - \mathrm{ufp}(\varepsilon(z) - \varepsilon(y) - \varepsilon_{+})$$

- Same for the multiplication, subtraction and division[3]
- Semantic of the commands in the dissertation
- Technique generalizable to sets of values

[3] Chapter 4 of the dissertation discusses the arithmetic expressions

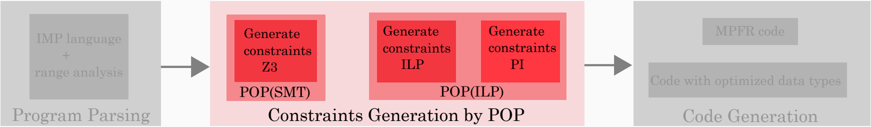
Table of Contents

Constraints Generation by POP



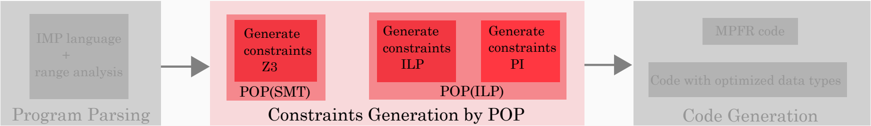
- **Approach:** static analysis of the error propagation

Constraints Generation by POP



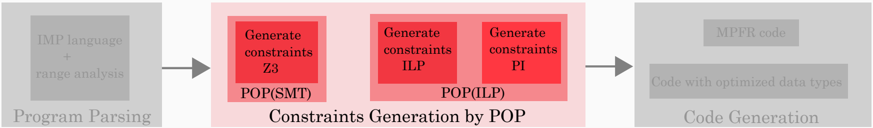
- **Approach:** static analysis of the error propagation
- **Formulation:** three systems of constraints

Constraints Generation by POP



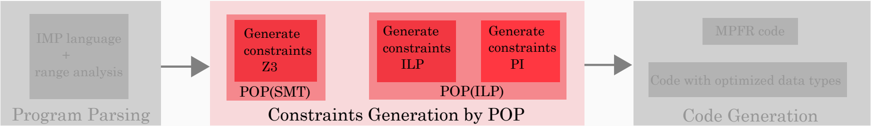
- **Approach:** static analysis of the error propagation
- **Formulation:** three systems of constraints
    1. **SMT-Based Method** checked by SMT solver
        - POP(SMT) version [FTSCS'19, IINTEC'19, IoTaIS'20, ICICT'21]

Constraints Generation by POP



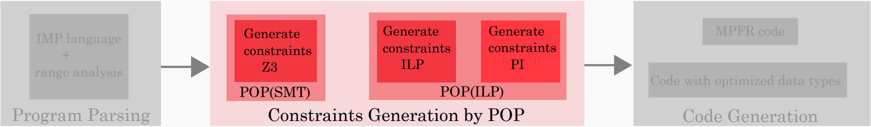
- **Approach:** static analysis of the error propagation
- **Formulation:** three systems of constraints
  1. **SMT-Based Method** checked by SMT solver
     - POP(SMT) version [FTSCS'19, IINTEC'19, IoTaIS'20, ICICT'21]
  2. **ILP-Based Method** solved by LP solver
     - Pessimistic carry bit
     - POP(ILP) version [ICCSA'21, SAS'21]

Constraints Generation by POP



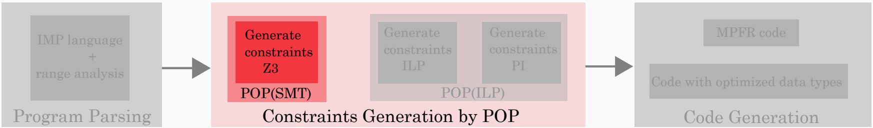
- **Approach:** static analysis of the error propagation
- **Formulation:** three systems of constraints
  1. **SMT-Based Method** checked by SMT solver
     - POP(SMT) version [FTSCS'19, IINTEC'19, IoTaIS'20, ICICT'21]
  2. **ILP-Based Method** solved by LP solver
     - Pessimistic carry bit
     - POP(ILP) version [ICCSA'21, SAS'21]
  3. **PI-Based Method** solved with policy iteration algorithm
     - Optimized carry bit
     - POP(ILP) version [ICCSA'21, SAS'21]

Constraints Generation by POP

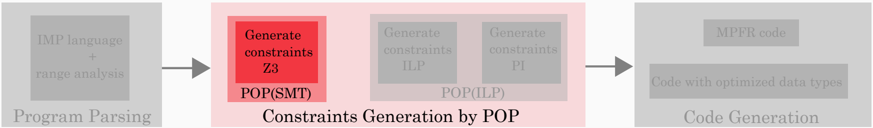


- **Approach:** static analysis of the error propagation
- **Formulation:** three systems of constraints
  1. **SMT-Based Method** checked by SMT solver
     - POP(SMT) version [FTSCS'19, IINTEC'19, IoTaIS'20, ICICT'21]
  2. **ILP-Based Method** solved by LP solver
     - Pessimistic carry bit
     - POP(ILP) version [ICCSA'21, SAS'21]
  3. **PI-Based Method** solved with policy iteration algorithm
     - Optimized carry bit
     - POP(ILP) version [ICCSA'21, SAS'21]
- **Output:** precision at bit-level translatable to IEEE754, fixed arithmetic, . . .

Table of Contents

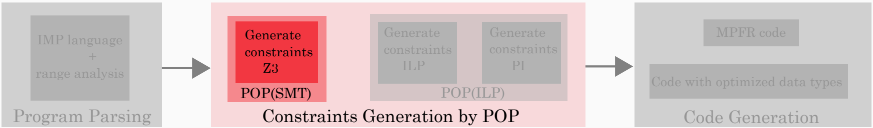


- First introduced in [NFM'17], improvement of carries in [FTSCS'19]



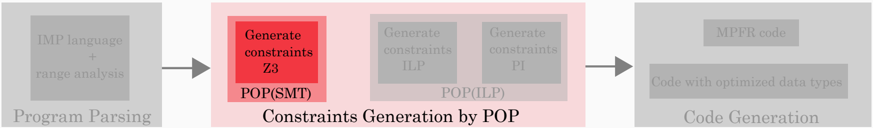
- First introduced in [NFM'17], improvement of carries in [FTSCS'19]
- **SMT-Based Method**[ab] **= forward analysis + backward analysis**

Formulation 1 SMT-Based Method

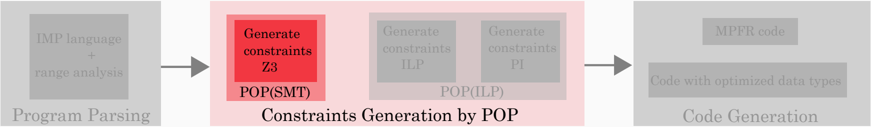


- First introduced in [NFM'17], improvement of carries in [FTSCS'19]
- **SMT-Based Method**[ab] = **forward analysis + backward analysis**
- Assign three parameters to each variable: nsb_F, nsb_B and nsb

Formulation 1 SMT-Based Method



- First introduced in [NFM'17], improvement of carries in [FTSCS'19]
- **SMT-Based Method**^{ab} = **forward analysis** + **backward analysis**
- Assign three parameters to each variable: nsb_F, nsb_B and nsb
- Constraints of **first order predicates** and with **linear integer relations** only



- First introduced in [NFM'17], improvement of carries in [FTSCS'19]
- **SMT-Based Method**[a b] = **forward analysis** + **backward analysis**
- Assign three parameters to each variable: nsb_F, nsb_B and nsb
- Constraints of **first order predicates** and with **linear integer relations** only
- Easy to solve for an SMT solver (Z3)

[a] M. Martel. *"Floating-Point Format Inference in Mixed-Precision"*. NFM'17

[b] D. Ben Khalifa, M. Martel and A. Adjé. *"POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations"*. FTSCS'19

- Recall that for the forward addition we have

$$\varepsilon(z) = \varepsilon(x) + \varepsilon(y) + \varepsilon_+$$

SMT Constraints Case of Addition $z = x + y$

- Recall that for the forward addition we have
$$\varepsilon(z) = \varepsilon(x) + \varepsilon(y) + \varepsilon_+$$
- First over-approximation
$$\varepsilon(z) \leq 2^{\mathsf{ufp}(x)-\mathsf{nsb}_F(x)+1} + 2^{\mathsf{ufp}(y)-\mathsf{nsb}_F(y)+1} + 2^{\mathsf{ufp}(z)-\mathsf{prec}(+)}$$

- Recall that for the forward addition we have

$$\varepsilon(z) = \varepsilon(x) + \varepsilon(y) + \varepsilon_{+}$$

- First over-approximation

$$\varepsilon(z) \leq 2^{\mathrm{ufp}(x)-\mathrm{nsb}_F(x)+1} + 2^{\mathrm{ufp}(y)-\mathrm{nsb}_F(y)+1} + 2^{\mathrm{ufp}(z)-\mathrm{prec}(+)}$$

- Second over-approximation

$$\varepsilon(z) \leq 2^{\max(\mathrm{ufp}(x)-\mathrm{nsb}_F(x),\mathrm{ufp}(y)-\mathrm{nsb}_F(y),\mathrm{ufp}(z)-\mathrm{prec}(+))+carry(z,x,y)}$$

- Recall that for the forward addition we have

$$\varepsilon(z) = \varepsilon(x) + \varepsilon(y) + \varepsilon_+$$

- First over-approximation

$$\varepsilon(z) \leq 2^{\mathrm{ufp}(x) - \mathrm{nsb}_F(x) + 1} + 2^{\mathrm{ufp}(y) - \mathrm{nsb}_F(y) + 1} + 2^{\mathrm{ufp}(z) - \mathrm{prec}(+)}$$

- Second over-approximation

$$\varepsilon(z) \leq 2^{\max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) + carry(z,x,y)}$$

- **Forward addition** $\overrightarrow{\oplus}$

$$\mathrm{nsb}_F(z) \geq \mathrm{ufp}(x+y) - \max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+))$$
$$- carry(z, x, y)$$

SMT Constraints Case of Addition $z = x + y$

- Recall that for the forward addition we have

$$\varepsilon(z) = \varepsilon(x) + \varepsilon(y) + \varepsilon_+$$

- First over-approximation

$$\varepsilon(z) \leq 2^{\mathrm{ufp}(x) - \mathrm{nsb}_F(x) + 1} + 2^{\mathrm{ufp}(y) - \mathrm{nsb}_F(y) + 1} + 2^{\mathrm{ufp}(z) - \mathrm{prec}(+)}$$

- Second over-approximation

$$\varepsilon(z) \leq 2^{\max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) + carry(z, x, y)}$$

- **Forward addition $\overrightarrow{\oplus}$**

$$\mathrm{nsb}_F(z) \geq \mathrm{ufp}(x+y) - \max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) - carry(z, x, y)$$

- **Backward addition $\overleftarrow{\oplus}$**

$$\mathrm{nsb}_B(x) \leq \mathrm{ufp}(z - y) - \mathrm{ufp}(z) + \mathrm{nsb}(z)$$

- Recall that for the forward addition we have

$$\varepsilon(z) = \varepsilon(x) + \varepsilon(y) + \varepsilon_{+}$$

- First over-approximation

$$\varepsilon(z) \leq 2^{\mathrm{ufp}(x) - \mathrm{nsb}_F(x)+1} + 2^{\mathrm{ufp}(y) - \mathrm{nsb}_F(y)+1} + 2^{\mathrm{ufp}(z) - \mathrm{prec}(+)}$$

- Second over-approximation

$$\varepsilon(z) \leq 2^{\max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) + \mathit{carry}(z,x,y)}$$

- **Forward addition $\overrightarrow{\oplus}$**

$$\mathrm{nsb}_F(z) \geq \mathrm{ufp}(x+y) - \max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) - \mathit{carry}(z,x,y)$$

- **Backward addition $\overleftarrow{\oplus}$**

$$\mathrm{nsb}_B(x) \leq \mathrm{ufp}(z-y) - \mathrm{ufp}(z) + \mathrm{nsb}(z)$$

- **Final precision**

$$0 \leq \mathrm{nsb}_B(x) \leq \mathrm{nsb}(x) \leq \mathrm{nsb}_F(x)$$

Elementary functions (sin, cos, tan, arcsin, log, . . .)

- Loss of precision of φ bits, where $\varphi \in \mathbb{N}$ a parameter of the analysis

example $x = 3.0$, $\varphi = 9$ and *require_nsb*$(\mathrm{sin}(x), 26) \Longrightarrow \overrightarrow{\mathrm{sin}}(x) = \mathrm{sin}(3.0|35|)|26|$

[4] Discussed in Chapter 5 of the dissertation

Elementary functions (sin, cos, tan, arcsin, log, ...)

- Loss of precision of φ bits, where $\varphi \in \mathbb{N}$ a parameter of the analysis

example $x = 3.0$, $\varphi = 9$ and $require\_nsb(\mathrm{sin}(x), 26) \Longrightarrow \overrightarrow{\mathrm{sin}}(x) = \mathrm{sin}(3.0|35|)|26|$

Loops

- Managed correctly thanks to range analysis
- Relate nsb at the end of the `body` to nsb of the same variables and the beginning of the loop

[4] Discussed in Chapter 5 of the dissertation

Elementary functions (sin, cos, tan, arcsin, log, ...)

- Loss of precision of φ bits, where $\varphi \in \mathbb{N}$ a parameter of the analysis

example $x = 3.0$, $\varphi = 9$ and *require_nsb*$(\mathrm{sin}(x), 26) \Longrightarrow \overrightarrow{\mathrm{sin}}(x) = \mathrm{sin}(3.0|35|)|26|$

Loops

- Managed correctly thanks to range analysis
- Relate nsb at the end of the `body` to nsb of the same variables and the beginning of the loop

Conditionals

- Analyze both `then` and `else` branches of the if statement without reducing the ranges of the variables
- ✘ Do not take care of the guards yet

[4] Discussed in Chapter 5 of the dissertation

```
g^{ℓ1} = 9.81^{ℓ0}; l^{ℓ3} = 0.5^{ℓ2};
y1^{ℓ5} = 0.785398^{ℓ4};
y2^{ℓ7} = 0.785398^{ℓ6};
h^{ℓ9} = 0.1^{ℓ8}; t^{ℓ11} = 0.0^{ℓ10};
while (t^{ℓ13} <^{ℓ15} 10.0^{ℓ14})^{ℓ59} {
 y1new^{ℓ24} = y1^{ℓ17} +^{ℓ23} y2^{ℓ19} *^{ℓ22} h^{ℓ21};
 aux1^{ℓ28} = sin(y1^{ℓ26})^{ℓ27};
 aux2^{ℓ40} = aux1^{ℓ30} *^{ℓ39} h^{ℓ32}
  *^{ℓ38} g^{ℓ34} /^{ℓ37} l^{ℓ36};
 y2new^{ℓ46} = y2^{ℓ42} -^{ℓ45} aux2^{ℓ44};
 t^{ℓ52} = t^{ℓ48} +^{ℓ51} h^{ℓ50};
 y1^{ℓ55} = y1new^{ℓ54};
 y2^{ℓ58} = y2new^{ℓ57}; };
require_nsb(y2, 20)^{ℓ61};
```

POP Label File

[5] D. Ben Khalifa, M. Martel and A. Adjé. *"POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations"*. FTSCS'19

```
g^ℓ1 = 9.81^ℓ0; l^ℓ3 = 0.5^ℓ2;
y1^ℓ5 = 0.785398^ℓ4;
y2^ℓ7 = 0.785398^ℓ6;
h^ℓ9 = 0.1^ℓ8; t^ℓ11 = 0.0^ℓ10;
while (t^ℓ13 <^ℓ15 10.0^ℓ14)^ℓ59 {
 y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21;
 aux1^ℓ28 = sin(y1^ℓ26)^ℓ27;
 aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
  *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36;
 y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44;
 t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50;
 y1^ℓ55 = y1new^ℓ54;
 y2^ℓ58 = y2new^ℓ57; };
require_nsb(y2, 20)^ℓ61;
```

POP Label File

$$
C_{Z3} = \left\{ \begin{array}{l}
//\textsc{Assign} \\
(assert(<= \mathrm{nsb}(\ell_{24})\ \mathrm{nsb}_F(\ell_{24}))) \\
(assert(<= \mathrm{nsb}_F(\ell_{56})\ \mathrm{nsb}_F(\ell_{24}))) \\
(assert(<= \mathrm{nsb}_B(\ell_{24})\ \mathrm{nsb}_B(\ell_{19}))) \\
(assert(>= \mathrm{nsb}_B(\ell_{24})\ \mathrm{nsb}_B(\ell_{13}))) \\
(assert(= \mathrm{nsb}_B(\ell_{56})\ \mathrm{nsb}_B(\ell_{24}))) \\
\ldots \\
//\textsc{Addition} \\
(assert(<= \mathrm{nsb}(\ell_{17})\ \mathrm{nsb}_F(\ell_{17}))) \\
(assert(<= \mathrm{nsb}_F(\ell_{17})\ \mathrm{nsb}_F(\ell_{5}))) \\
\ldots \\
//\textsc{Multiplication} \\
(assert(or(and(<= \mathrm{nsb}(\ell_{17})\ \mathrm{nsb}_F(\ell_{17}))(>= \mathrm{nsb}(\ell_{22}) \\
\ \mathrm{nsb}_B(\ell_{22})))(and(<= \mathrm{nsb}(\ell_{22}) \\
\mathrm{nsb}_F(\ell_{22}))(>= \mathrm{nsb}(\ell_{17})\ \mathrm{nsb}_B(\ell_{17}))))) \\
\ldots \\
//\textsc{Carry Bit} \\
(assert(= carry(\ell_{23}, \ell_{17}, \ell_{22})(ite(>\ (ite(>\ \mathrm{ulp_e}(\ell_0)(-0\ \ 56))\ 0\ 1) \\
(ite(>\ \mathrm{ulp_e}(\ell_0)\ (-\ 0\ 52))\ 0\ 1))(ite(>\ \mathrm{ulp_e}(\ell_0)(-\ 0\ \ 56))0\ \ 1) \\
(ite(>\ \mathrm{ulp_e}(\ell_0)(-\ 0\ \ 52))0\ \ 1)))) \\
\ldots \\
\ldots \\
\ldots
\end{array} \right\}
$$

☞ Linear number of constraints / variables in the size of the analyzed program

[5] D. Ben Khalifa, M. Martel and A. Adjé. *"POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations"*. FTSCS'19

```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|21| = 0.0|21|;
h|21| = 0.1|21|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|21|*|22| h|21|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|19| *|20| h|18|*|19|g|17| /|18|l|17|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_Z3

```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|21| = 0.0|21|;
h|21| = 0.1|21|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|21|*|22| h|21|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|19| *|20| h|18|*|19|g|17| /|18|l|17|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_Z3

Tool	#Bits	#Var Solver	#Cstr Solver	Time (s)
POP(SMT)	960	314	431	13.1

```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|21| = 0.0|21|;
h|21| = 0.1|21|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|21|*|22| h|21|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|19| *|20| h|18|*|19|g|17| /|18|l|17|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_Z3

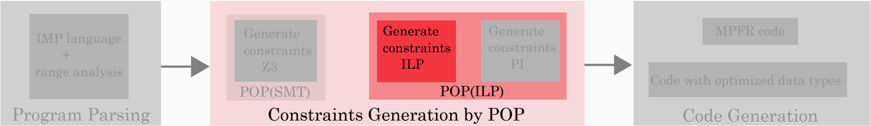
Tool	#Bits	#Var Solver	#Cstr Solver	Time (s)
POP(SMT)	960	314	431	13.1

Cons

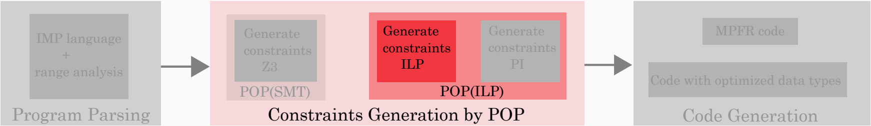
- ✗ Too complex system of constraints
- ✗ Non-optimizing solver coupled with binary search
- ✗ Is the forward analysis really useful?

Table of Contents

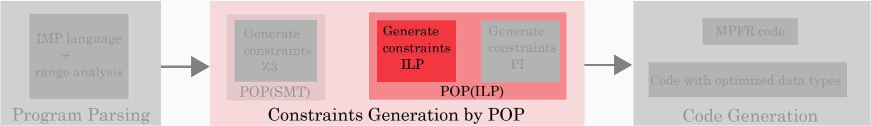
Formulation 2 ILP-Based Method



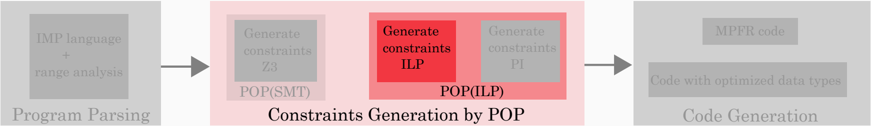
- Relaxation of the SMT-based method $\Longrightarrow$ backward analysis only



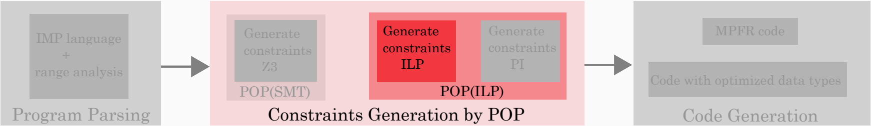
- Relaxation of the SMT-based method $\Longrightarrow$ backward analysis only
- Generate an ILP from the program source code



- Relaxation of the SMT-based method $\Longrightarrow$ backward analysis only
- Generate an ILP from the program source code
- Over-approximated carry bit function $\Longrightarrow$ $carry = 1$



- Relaxation of the SMT-based method $\Longrightarrow$ backward analysis only
- Generate an ILP from the program source code
- Over-approximated carry bit function $\Longrightarrow$ $carry = 1$
- Former constraints used to generate an ILP

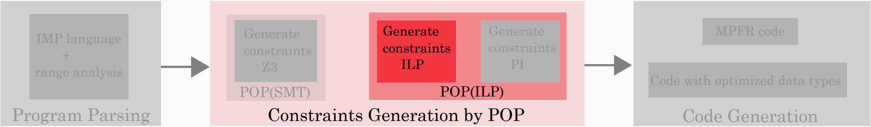


- Relaxation of the SMT-based method $\Longrightarrow$ backward analysis only
- Generate an ILP from the program source code
- Over-approximated carry bit function $\Longrightarrow$ $\mathit{carry} = 1$
- Former constraints used to generate an ILP

Case of Addition

☞ Since

$$\mathrm{nsb}(z) \leq \mathrm{ufp}(z) - \max\left(\mathrm{ufp}(x) - \mathrm{nsb}(x), \mathrm{ufp}(y) - \mathrm{nsb}(y)\right) - \mathit{carry}(z, y, x)$$



- Relaxation of the SMT-based method $\Longrightarrow$ backward analysis only
- Generate an ILP from the program source code
- Over-approximated carry bit function $\Longrightarrow$ $\mathit{carry} = 1$
- Former constraints used to generate an ILP

Case of Addition

☞ Since

$$\mathsf{nsb}(z) \leq \mathsf{ufp}(z) - \max\left(\mathsf{ufp}(x) - \mathsf{nsb}(x), \mathsf{ufp}(y) - \mathsf{nsb}(y)\right) - \mathit{carry}(z, y, x)$$

☞ We generate an ILP (with relaxation)

$$(S) = \begin{cases} \mathsf{nsb}(x) \geq \mathsf{nsb}(z) + \mathsf{ufp}(x) - \mathsf{ufp}(z) + 1 \\ \mathsf{nsb}(y) \geq \mathsf{nsb}(z) + \mathsf{ufp}(y) - \mathsf{ufp}(z) + 1 \end{cases}$$

Running Example ILP Constraints[7]

$g^{\ell 1} = 9.81^{\ell 0};\ l^{\ell 3} = 0.5^{\ell 2};$

$y1^{\ell 5} = 0.785398^{\ell 4};$

$y2^{\ell 7} = 0.785398^{\ell 6};$

$h^{\ell 9} = 0.1^{\ell 8};\ t^{\ell 11} = 0.0^{\ell 10};$

$\textbf{while}\ (t^{\ell 13} <^{\ell 15} 10.0^{\ell 14})^{\ell 59}\ \{$

$\quad \mathbf{y1new}^{\ell 24} = \mathbf{y1}^{\ell 17} +^{\ell 23} \mathbf{y2}^{\ell 19} *^{\ell 22} \mathbf{h}^{\ell 21};$

$\quad aux1^{\ell 28} = sin(y1^{\ell 26})^{\ell 27};$

$\quad aux2^{\ell 40} = aux1^{\ell 30} *^{\ell 39} h^{\ell 32}$

$\quad\ *^{\ell 38} g^{\ell 34} /^{\ell 37} l^{\ell 36};$

$\quad y2new^{\ell 46} = y2^{\ell 42} -^{\ell 45} aux2^{\ell 44};$

$\quad t^{\ell 52} = t^{\ell 48} +^{\ell 51} h^{\ell 50};$

$\quad y1^{\ell 55} = y1new^{\ell 54};$

$\quad y2^{\ell 58} = y2new^{\ell 57};\ \};$

$\mathbf{require\_nsb(y2, 20)}^{\ell 61};$

POP Label File

[6] ILP nature of the problem presented in Chapter 5 Theorem 5.2

[7] A. Adjé, D. Ben Khalifa and M. Martel. *"Fast and Efficient Bit-Level Precision Tuning"*. SAS'21

Running Example ILP Constraints[7]

```
g^ℓ1 = 9.81^ℓ0 ; l^ℓ3 = 0.5^ℓ2 ;
y1^ℓ5 = 0.785398^ℓ4 ;
y2^ℓ7 = 0.785398^ℓ6 ;
h^ℓ9 = 0.1^ℓ8 ; t^ℓ11 = 0.0^ℓ10 ;
while (t^ℓ13 <^ℓ15 10.0^ℓ14)^ℓ59 {
  y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21 ;
  aux1^ℓ28 = sin(y1^ℓ26)^ℓ27 ;
  aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
   *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36 ;
  y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44 ;
  t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50 ;
  y1^ℓ55 = y1new^ℓ54 ;
  y2^ℓ58 = y2new^ℓ57 ; };
require_nsb(y2, 20)^ℓ61 ;
```

POP Label File

$$C_{GLPK} = \begin{cases} \mathrm{nsb}(\ell_{17}) \geq \mathrm{nsb}(\ell_{23}) + (-1) + \mathit{carry}(\ell_{23})(\ell_{17}, \ell_{22}) - (-1) \; // \textsc{Add} \\ \mathrm{nsb}(\ell_{22}) \geq \mathrm{nsb}(\ell_{23}) + 0 + \mathit{carry}(\ell_{23})(\ell_{17}, \ell_{22}) - (1) \; // \textsc{Add} \\ \mathrm{nsb}(\ell_{19}) \geq \mathrm{nsb}(\ell_{22}) + \mathit{carry}(\ell_{22})(\ell_{19}, \ell_{21}) - 1 \; // \textsc{Mult} \\ \mathrm{nsb}(\ell_{21}) \geq \mathrm{nsb}(\ell_{22}) + \mathit{carry}(\ell_{22})(\ell_{19}, \ell_{21}) - 1 \; // \textsc{Mult} \\ \mathrm{nsb}(\ell_{23}) \geq \mathrm{nsb}(\ell_{24}) \; // \textsc{Assign} \\ \mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}) = 1, \mathit{carry}(\ell_{22}, \ell_{19}, \ell_{21}) = 1 \; // \textsc{Carry bit} \end{cases}$$

[6] ILP nature of the problem presented in Chapter 5 Theorem 5.2

[7] A. Adjé, D. Ben Khalifa and M. Martel. *"Fast and Efficient Bit-Level Precision Tuning"*. SAS'21

$$C_{GLPK} = \begin{cases} \mathrm{nsb}(\ell_{17}) \geq \mathrm{nsb}(\ell_{23}) + (-1) + \mathit{carry}(\ell_{23})(\ell_{17}, \ell_{22}) - (-1) \; //\textsc{Add} \\ \mathrm{nsb}(\ell_{22}) \geq \mathrm{nsb}(\ell_{23}) + 0 + \mathit{carry}(\ell_{23})(\ell_{17}, \ell_{22}) - (1) \; //\textsc{Add} \\ \mathrm{nsb}(\ell_{19}) \geq \mathrm{nsb}(\ell_{22}) + \mathit{carry}(\ell_{22})(\ell_{19}, \ell_{21}) - 1 \; //\textsc{Mult} \\ \mathrm{nsb}(\ell_{21}) \geq \mathrm{nsb}(\ell_{22}) + \mathit{carry}(\ell_{22})(\ell_{19}, \ell_{21}) - 1 \; //\textsc{Mult} \\ \mathrm{nsb}(\ell_{23}) \geq \mathrm{nsb}(\ell_{24}) \; //\textsc{Assign} \\ \boxed{\mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}) = 1, \mathit{carry}(\ell_{22}, \ell_{19}, \ell_{21}) = 1} \; //\textsc{Carry bit} \end{cases}$$

```
g^ℓ1 = 9.81^ℓ0; l^ℓ3 = 0.5^ℓ2;
y1^ℓ5 = 0.785398^ℓ4;
y2^ℓ7 = 0.785398^ℓ6;
h^ℓ9 = 0.1^ℓ8; t^ℓ11 = 0.0^ℓ10;
while (t^ℓ13 <^ℓ15 10.0^ℓ14)^ℓ59 {
  y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21;
  aux1^ℓ28 = sin(y1^ℓ26)^ℓ27;
  aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
   *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36;
  y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44;
  t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50;
  y1^ℓ55 = y1new^ℓ54;
  y2^ℓ58 = y2new^ℓ57; };
  require_nsb(y2, 20)^ℓ61;
```

POP Label File

POP(ILP)[6]

- Linear number of constraints / variables in the size of the analyzed program
- Integer solution computed in **polynomial time**
- LP solver among reals

[6] ILP nature of the problem presented in Chapter 5 Theorem 5.2

[7] A. Adjé, D. Ben Khalifa and M. Martel. *"Fast and Efficient Bit-Level Precision Tuning"*. SAS'21

```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|22| = 0.0|22|;
h|22| = 0.1|22|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|22|*|22| h|22|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|20| *|20| h|20|*|20| g|20|/|20|l|20|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_GLPK

Tool	#Bits	#Var Solver	#Cstr Solver	Time (s)
POP(SMT)	960	314	431	13.1
POP(ILP)	861	52	69	3.5

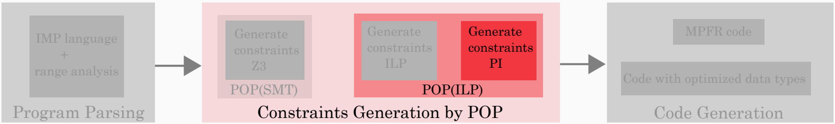
```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|22| = 0.0|22|;
h|22| = 0.1|22|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|22|*|22| h|22|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|20| *|20| h|20|*|20| g|20|/|20|l|20|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_GLPK

Tool	#Bits	#Var Solver	#Cstr Solver	Time (s)
POP(SMT)	960	314	431	13.1
POP(ILP)	861	52	69	3.5

☞ How to avoid the pessimistic carry bit?

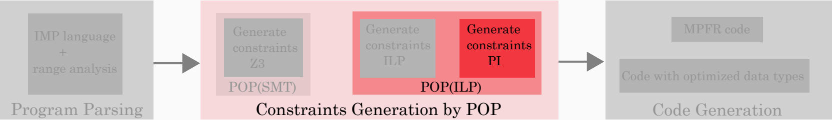
Table of Contents



- Policy iteration algorithm[8] to solve $\min - \max$ equations

[8] Algorithm 2 presented in Chapter 5 page 86

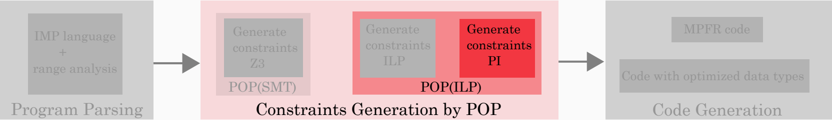
Formulation 3 PI-Based Method



- Policy iteration algorithm[8] to solve $\min - \max$ equations
- Back to Line 6 of the pendulum example

$$carry(\ell, \ell_1, \ell_2) = \min\left(\begin{array}{l} \max\left(\mathrm{ufp}(\ell_2) - \mathrm{ufp}(\ell_1) + \mathrm{nsb}(\ell_1) - \mathrm{nsb}(\ell_2) - \mathrm{nsb_e}(\ell_2), 0\right), \\ \max\left(\mathrm{ufp}(\ell_1) - \mathrm{ufp}(\ell_2) + \mathrm{nsb}(\ell_2) - \mathrm{nsb}(\ell_1) - \mathrm{nsb_e}(\ell_1), 0\right), 1 \end{array} \right)$$

[8] Algorithm 2 presented in Chapter 5 page 86



- Policy iteration algorithm[8] to solve $\min - \max$ equations
- Back to Line 6 of the pendulum example

$$carry(\ell, \ell_1, \ell_2) = \min \begin{pmatrix} \max\left(\mathsf{ufp}(\ell_2) - \mathsf{ufp}(\ell_1) + \mathsf{nsb}(\ell_1) - \mathsf{nsb}(\ell_2) - \mathsf{nsb_e}(\ell_2), 0\right), \\ \max\left(\mathsf{ufp}(\ell_1) - \mathsf{ufp}(\ell_2) + \mathsf{nsb}(\ell_2) - \mathsf{nsb}(\ell_1) - \mathsf{nsb_e}(\ell_1), 0\right), 1 \end{pmatrix}$$

Principle

- Choose a policy $\pi_0 \in \Pi$ by breaking the $\min$ operator
- Associate policy map f^{π_0} to π_0

example $f^{\pi_0} = \max\left(\mathsf{ufp}(\ell_2) - \mathsf{ufp}(\ell_1) + \mathsf{nsb}(\ell_1) - \mathsf{nsb}(\ell_2) - \mathsf{nsb_e}(\ell_2), 0\right)$

- Solve corresponding ILP problem
- If a more precise solution is found, PI algorithm iterates

[8] Algorithm 2 presented in Chapter 5 page 86

Running Example PI Constraints[9]

```
g^ℓ1 = 9.81^ℓ0 ; l^ℓ3 = 0.5^ℓ2 ;
y1^ℓ5 = 0.785398^ℓ4 ;
y2^ℓ7 = 0.785398^ℓ6 ;
h^ℓ9 = 0.1^ℓ8 ; t^ℓ11 = 0.0^ℓ10 ;
while (t^ℓ13 <^ℓ15 10.0^ℓ14)^ℓ59 {
 y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21 ;
 aux1^ℓ28 = sin(y1^ℓ26)^ℓ27 ;
 aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
  *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36 ;
 y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44 ;
 t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50 ;
 y1^ℓ55 = y1new^ℓ54 ;
 y2^ℓ58 = y2new^ℓ57 ; };
 require_nsb(y2, 20)^ℓ61 ;
```

POP Label File

[9] A. Adjé, D. Ben Khalifa and M. Martel. *"Fast and Efficient Bit-Level Precision Tuning"*. SAS'21

```
g^ℓ1 = 9.81^ℓ0 ; l^ℓ3 = 0.5^ℓ2 ;
y1^ℓ5 = 0.785398^ℓ4 ;
y2^ℓ7 = 0.785398^ℓ6 ;
h^ℓ9 = 0.1^ℓ8 ; t^ℓ11 = 0.0^ℓ10 ;
while (t^ℓ13 <^ℓ15 10.0^ℓ14 )^ℓ59 {
 y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21 ;
 aux1^ℓ28 = sin(y1^ℓ26 )^ℓ27 ;
 aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
  *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36 ;
 y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44 ;
 t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50 ;
 y1^ℓ55 = y1new^ℓ54 ;
 y2^ℓ58 = y2new^ℓ57 ; };
require_nsb(y2, 20)^ℓ61 ;
```

POP Label File

$$
C_{PI} = \left\{
\begin{array}{l}
\mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{17}), \ //\textsc{Add} \\
\mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{22}), \ //\textsc{Add} \\
\mathsf{nsb_e}(\ell_{23}) \geq -1 - 0 + \mathsf{nsb}(\ell_{22}) - \mathsf{nsb}(\ell_{17}) + \mathsf{nsb_e}(\ell_{22}) + \\
\mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}), \ //\textsc{Add} \\
\mathsf{nsb_e}(\ell_{23}) \geq 0 - (-1) + \mathsf{nsb}(\ell_{17}) - \mathsf{nsb}(\ell_{22}) + \mathsf{nsb_e}(\ell_{17}) + \\
\mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}), \ //\textsc{Add} \\
\mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{24}), \ //\textsc{Add} \\
\mathsf{nsb_e}(\ell_{22}) \geq \mathsf{nsb}(\ell_{19}) + \mathsf{nsb_e}(\ell_{19}) + \mathsf{nsb_e}(\ell_{21}) - 2, \ //\textsc{Mult} \\
\mathsf{nsb_e}(\ell_{22}) \geq \mathsf{nsb}(\ell_{21}) + \mathsf{nsb_e}(\ell_{21}) + \mathsf{nsb_e}(\ell_{19}) - 2, \ //\textsc{Mult} \\
\mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}) = \min \left(\begin{array}{l} \max\left(0 - 6 + \mathsf{nsb}(\ell_{17}) - \mathsf{nsb}(\ell_{22}) - \right. \\ \left. \mathsf{nsb_e}(\ell_{17}), 0\right), \\ \max\left(6 - 0 + \mathsf{nsb}(\ell_{22}) - \mathsf{nsb}(\ell_{17}) - \right. \\ \left. \mathsf{nsb_e}(\ell_{22}), 0\right), 1 \end{array} \right) \\
//\textsc{Carry}
\end{array}
\right.
$$

---

[9] A. Adjé, D. Ben Khalifa and M. Martel. *"Fast and Efficient Bit-Level Precision Tuning"*. SAS'21

Running Example PI Constraints[9]

$$C_{PI} = \begin{cases} \mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{17}), \quad \text{//ADD} \\ \mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{22}), \quad \text{//ADD} \\ \mathsf{nsb_e}(\ell_{23}) \geq -1 - 0 + \mathsf{nsb}(\ell_{22}) - \mathsf{nsb}(\ell_{17}) + \mathsf{nsb_e}(\ell_{22}) + \\ \mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}), \quad \text{//ADD} \\ \mathsf{nsb_e}(\ell_{23}) \geq 0 - (-1) + \mathsf{nsb}(\ell_{17}) - \mathsf{nsb}(\ell_{22}) + \mathsf{nsb_e}(\ell_{17}) + \\ \mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}), \quad \text{//ADD} \\ \mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{24}), \quad \text{//ADD} \\ \mathsf{nsb_e}(\ell_{22}) \geq \mathsf{nsb}(\ell_{19}) + \mathsf{nsb_e}(\ell_{19}) + \mathsf{nsb_e}(\ell_{21}) - 2, \quad \text{//MULT} \\ \mathsf{nsb_e}(\ell_{22}) \geq \mathsf{nsb}(\ell_{21}) + \mathsf{nsb_e}(\ell_{21}) + \mathsf{nsb_e}(\ell_{19}) - 2, \quad \text{//MULT} \\ \mathit{carry}(\ell_{23}, \ell_{17}, \ell_{22}) = \min \begin{pmatrix} \max\left(0 - 6 + \mathsf{nsb}(\ell_{17}) - \mathsf{nsb}(\ell_{22}) - \mathsf{nsb_e}(\ell_{17}), 0\right), \\ \max\left(6 - 0 + \mathsf{nsb}(\ell_{22}) - \mathsf{nsb}(\ell_{17}) - \mathsf{nsb_e}(\ell_{22}), 0\right), 1 \end{pmatrix} \\ \text{//CARRY} \end{cases}$$

```
g^ℓ1 = 9.81^ℓ0; l^ℓ3 = 0.5^ℓ2;
y1^ℓ5 = 0.785398^ℓ4;
y2^ℓ7 = 0.785398^ℓ6;
h^ℓ9 = 0.1^ℓ8; t^ℓ11 = 0.0^ℓ10;
while (t^ℓ13 <^ℓ15 10.0^ℓ14)^ℓ59 {
  y1new^ℓ24 = y1^ℓ17 +^ℓ23 y2^ℓ19 *^ℓ22 h^ℓ21;
  aux1^ℓ28 = sin(y1^ℓ26)^ℓ27;
  aux2^ℓ40 = aux1^ℓ30 *^ℓ39 h^ℓ32
   *^ℓ38 g^ℓ34 /^ℓ37 l^ℓ36;
  y2new^ℓ46 = y2^ℓ42 -^ℓ45 aux2^ℓ44;
  t^ℓ52 = t^ℓ48 +^ℓ51 h^ℓ50;
  y1^ℓ55 = y1new^ℓ54;
  y2^ℓ58 = y2new^ℓ57; };
  require_nsb(y2, 20)^ℓ61;
```

POP Label File

POP(ILP) with PI

- $C = C_{GLPK} \cup C_{PI}$: final set of constraints
- Activate optimized *carry*
- Feasible solution fastly computed

[9] A. Adjé, D. Ben Khalifa and M. Martel. *"Fast and Efficient Bit-Level Precision Tuning"*. SAS'21

Running Example Results with PI

```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|21| = 0.0|21|;
h|21| = 0.1|21|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|21|*|22| h|21|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|19| *|20| h|18|*|19|g|17| /|18|l|17|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_PI

Tool	#Bits	#Var Solver	#Cstr Solver	Time (s)
POP(SMT)	960	314	431	13.1
POP(ILP)	861	52	69	3.5
POP(ILP)/ PI	843	97	204 (1 policy)	4.1

Running Example — Results with PI

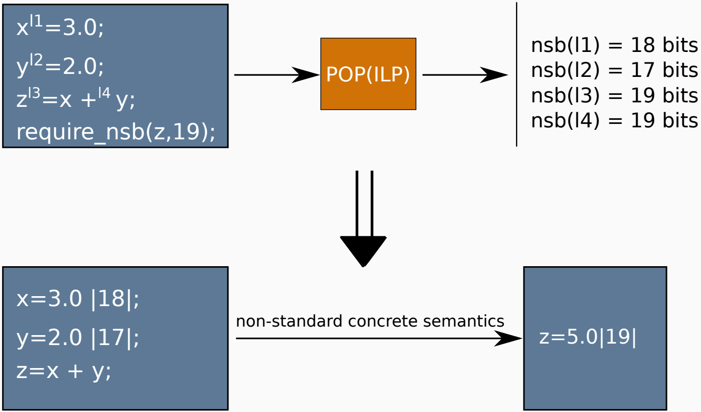
```
g|20| = 9.81|20|; l|20| = 1.5|20|;
y1|29| = 0.785398|29|;
y2|21| = 0.0|21|;
h|21| = 0.1|21|; t|21| = 0.0|21|;
while (t<1.0) {
  y1new|20| = y1|21| +|20| y2|21|*|22| h|21|;
  aux1|20| = sin(y1|29|)|20|;
  aux2|20| = aux1|19| *|20| h|18|*|19|g|17| /|18|l|17|;
  y2new|20| = y2|21| -|20| aux2|18|;
  t|20| = t|21| +|20| h|17|;
  y1|20| = y1new|20|;
  y2|20| = y2new|20|;
};
require_nsb(y2, 20);
```

File POP_output_PI

Tool	#Bits	#Var Solver	#Cstr Solver	Time (s)
POP(SMT)	960	314	431	13.1
POP(ILP)	861	52	69	3.5
POP(ILP)/ PI	843	97	204 (1 policy)	4.1

Correctness

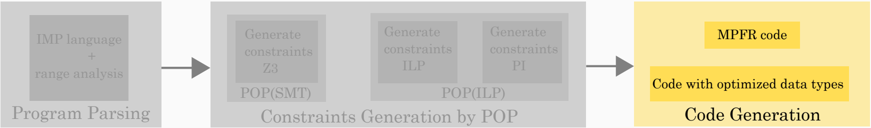
- Soundness of the constraint system of the ILP formulation[10]



[10] Presented and proved in Chapter 5 Theorem 5.1

Table of Contents

Methodology

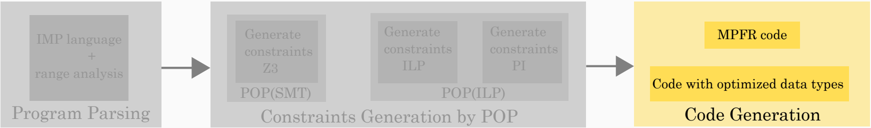


Code with mixed-precision at bit-level

- Translation of bit-level precision to the upper number of bits corresponding to an IEEE-754 format
- Fixed-point formats $\Longrightarrow$ soon!

[11] https://www.mpfr.org/

Methodology



Code with mixed-precision at bit-level

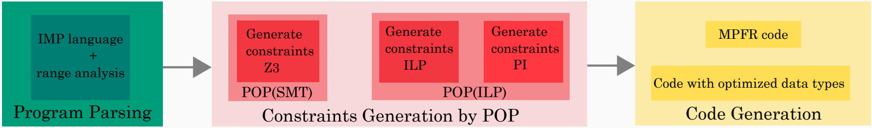
- Translation of bit-level precision to the upper number of bits corresponding to an IEEE-754 format
- Fixed-point formats $\Longrightarrow$ soon!

MPFR[11] code generation

- Generate MPFR program by assuming that the exact computations are performed in higher precision (300 bits)
- Generate MPFR program with precision returned by POP
- **Goal:** measure the difference between the two programs in fonction of the theoretical error given by the user

[11] https://www.mpfr.org/

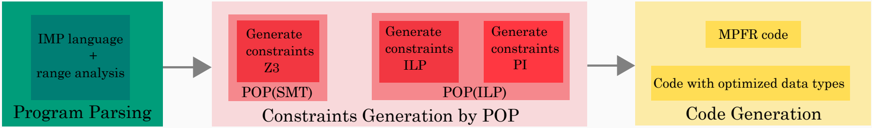
Evaluation Plan



Evaluation[12] is meant to answer questions

[12] Intel Core i5-8350U CPU cadenced at 1.7GHz on a Linux machine with 8 GB RAM
[13] https://www.gnu.org/software/gsl/
[14] https://fpbench.org/

Evaluation Plan

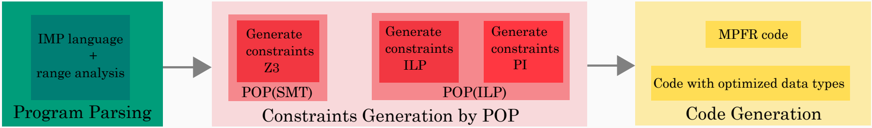


Evaluation[12] is meant to answer questions

- Which version of POP is more efficient?

[12] Intel Core i5-8350U CPU cadenced at 1.7GHz on a Linux machine with 8 GB RAM
[13] https://www.gnu.org/software/gsl/
[14] https://fpbench.org/

Evaluation Plan



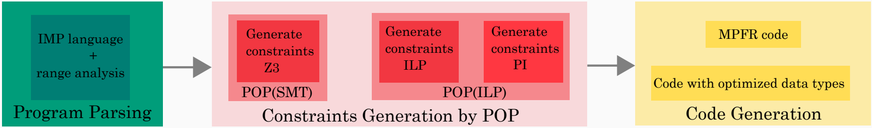
Evaluation[12] is meant to answer questions

- Which version of POP is more efficient?
- How the returned precision are validated?

---

[12] Intel Core i5-8350U CPU cadenced at 1.7GHz on a Linux machine with 8 GB RAM
[13] https://www.gnu.org/software/gsl/
[14] https://fpbench.org/

Evaluation Plan

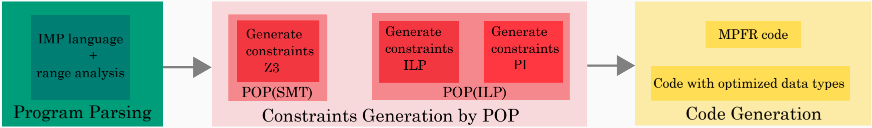


Evaluation[12] is meant to answer questions

- Which version of POP is more efficient?
- How the returned precision are validated?
- Which impact has the policy iteration (PI) method on POP(ILP)?

[12] Intel Core i5-8350U CPU cadenced at 1.7GHz on a Linux machine with 8 GB RAM
[13] https://www.gnu.org/software/gsl/
[14] https://fpbench.org/

Evaluation Plan

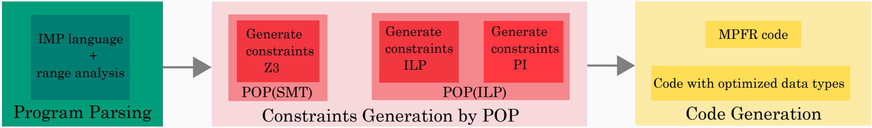


Evaluation[12] is meant to answer questions

- Which version of POP is more efficient?
- How the returned precision are validated?
- Which impact has the policy iteration (PI) method on POP(ILP)?
- POP(SMT) vs POP(ILP) vs Precimonious?

[12] Intel Core i5-8350U CPU cadenced at 1.7GHz on a Linux machine with 8 GB RAM
[13] https://www.gnu.org/software/gsl/
[14] https://fpbench.org/

Evaluation Plan



Evaluation[12] is meant to answer questions

- Which version of POP is more efficient?
- How the returned precision are validated?
- Which impact has the policy iteration (PI) method on POP(ILP)?
- POP(SMT) vs POP(ILP) vs Precimonious?

Target applications

- IoT, GNU scientific library [13] (arclength, simpson, . . .), physics (N-Body program $\approx$ 400 LOCs), FPBench[14] (Trapeze, Runge Kutta, PID, . . .), . . .

[12] Intel Core i5-8350U CPU cadenced at 1.7GHz on a Linux machine with 8 GB RAM
[13] https://www.gnu.org/software/gsl/
[14] https://fpbench.org/

Experimental Setup

Optimization parameters

- $100 \equiv$ all control points (CP) in **FP**64 precision (53 mantissa size)

example If we have 8 |CP| $\Longrightarrow$ $100 = 8 \times 53 = 424$ bits

Experimental Setup

Optimization parameters

- $100 \equiv$ all control points (CP) in **FP**64 precision (53 mantissa size)

example If we have 8 |CP| $\Longrightarrow$ $100 = 8 \times 53 = 424$ bits

Optimization in Bit-Level (BL)

$$\boxed{\mathrm{BL} = \frac{\sum_{l \in CP} \mathrm{nsb}(\ell) \times 100}{|CP| \times 53}}$$

Experimental Setup

Optimization parameters

- $100 \equiv$ all control points (CP) in **FP**64 precision (53 mantissa size)

example If we have 8 |CP| $\Longrightarrow$ $100 = 8 \times 53 = 424$ bits

Optimization in Bit-Level (BL)

$$\boxed{\mathrm{BL} = \frac{\sum_{l \in CP} \mathrm{nsb}(\ell) \times 100}{|CP| \times 53}}$$

From precision at bit-level to IEEE mode

- Round to the upper IEEE-754 format

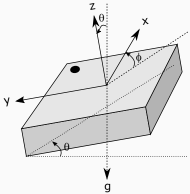
example nsb $= 8$ bits $\longrightarrow 10$ (FP16)

example nsb $= 20$ bits $\longrightarrow 23$ (FP32)

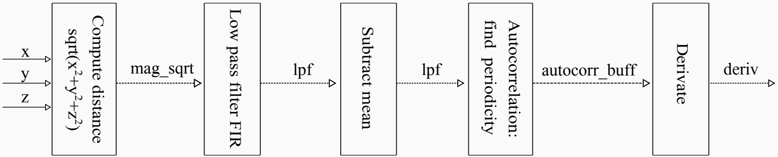
example nsb $= 25$ bits $\longrightarrow 53$ (FP64)

Table of Contents

- **Application 1: Tilt angle measurements by a 3 axis accelerometer**[15]

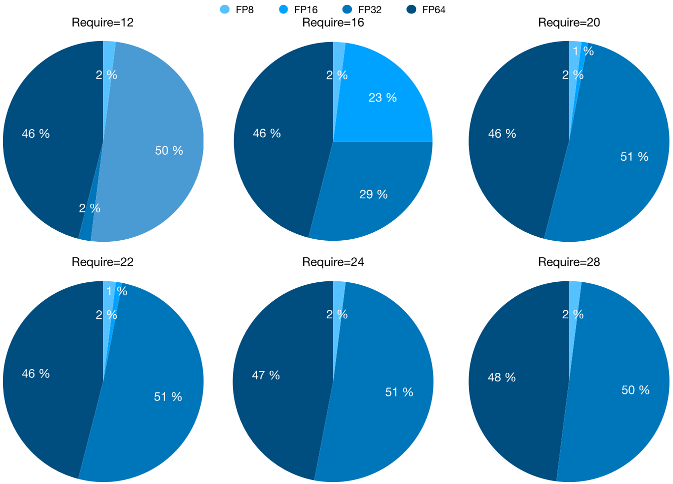


- **Application 2: Footstep counter program**[16]

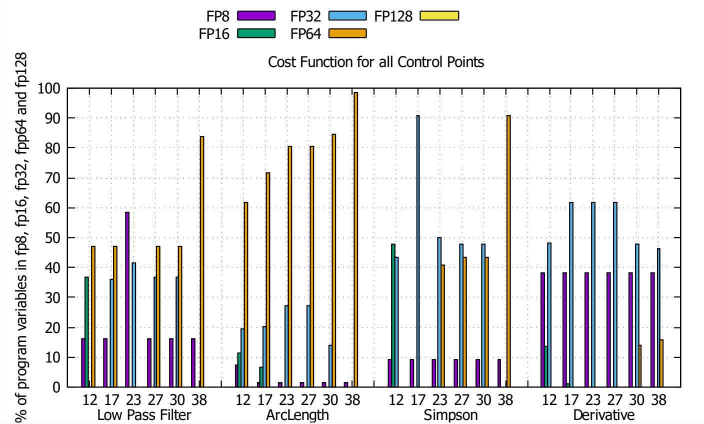


[15] D. Ben Khalifa and M.Martel. *"Precision Tuning and Internet of Things".IINTEC'19.*

[16] D.Ben Khalifa and M. Martel. *"Precision Tuning of an Accelerometer-Based Pedometer Algorithm for IoT Devices"*.IoTaIS'20



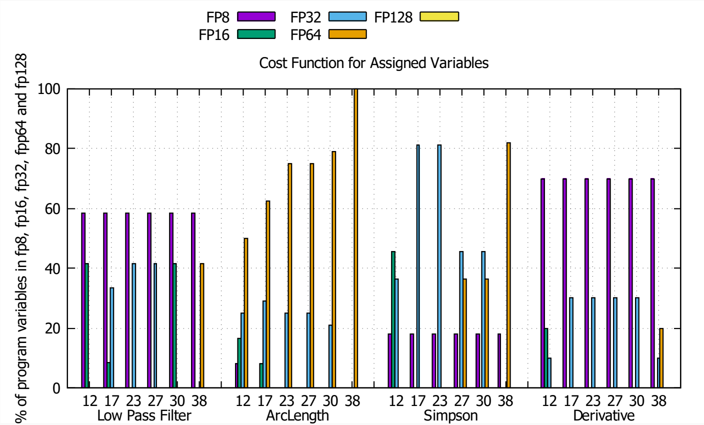
- **Goal:** measure the percentage of program variables in FP8, FP16, FP32 and FP64 after POP(SMT) analysis



- Cost function **with all control points**
- Results in mixed precision for nsb $= 12, 17, 23, 27, 30$ and 38 bits

[17] D. Ben Khalifa and M. Martel. *"An Evaluation of POP Performance for Tuning Numerical Programs in Floating-point Arithmetic".ICICT'21.*

Performance of POP(SMT) Z3 Cost Functions [17]

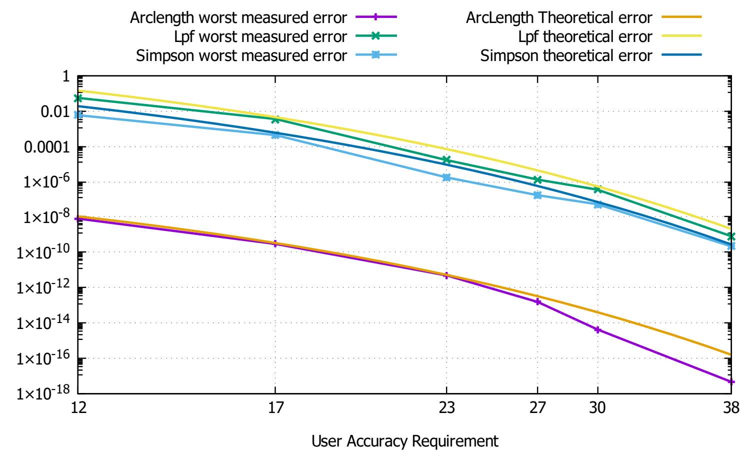


- More comparable cost function **with only assigned variables** $\Longrightarrow$ same as in Precimonious

☞ Better optimization of the number of bits of programs

[17] D. Ben Khalifa and M. Martel. *"An Evaluation of POP Performance for Tuning Numerical Programs in Floating-point Arithmetic".ICICT'21.*

POP(SMT) Generate MPFR Code



- **Goal:** Curve of the difference between the exact results (300 bits) and the precision of POP(SMT) **sticks** to the theoretical curve by below

Table of Contents

Precision Tuning Results for ILP and PI Methods[18]

Program	TH	BL	IEEE	ILP-time	BL	IEEE	PI-time	H	S	D	LD
accelerometer	10^{-4}	**73%**	**61%**	0.2s	**76%**	**62%**	1.0s	53	69	0	0
	10^{-6}	**62%**	**55%**	0.2s	**65%**	**55%**	1.0s	2	102	0	0
	10^{-8}	**49%**	**15%**	0.2s	**52%**	**18%**	1.0s	2	33	69	0
	10^{-10}	**36%**	**1%**	0.2s	**39%**	**1%**	1.0s	2	0	102	0
	10^{-12}	**25%**	**1%**	0.2s	**28%**	**1%**	1.0s	2	0	102	0
lowPassFilter	10^{-4}	**68%**	**46%**	1.8s	**69%**	**46%**	10.7s	260	581	0	0
	10^{-6}	**57%**	**38%**	1.8s	**58%**	**45%**	11.0s	258	580	3	0
	10^{-8}	**44%**	**-7%**	2.0s	**45%**	**-7%**	11.4s	258	2	581	0
	10^{-10}	**31%**	**-7%**	1.7s	**32%**	**-7%**	10.9s	258	0	583	0
	10^{-12}	**20%**	**-7%**	1.8s	**21%**	**-7%**	11.3s	258	0	583	0
2-Body	10^{-4}	**51%**	**41%**	0.81s	**51%**	**41%**	0.82s	5	39	5	0
	10^{-6}	**49%**	**18%**	0.78s	**49%**	**18%**	0.9s	0	44	5	0
	10^{-8}	**-7%**	**5%**	0.8s	**-7%**	**5%**	0.78s	0	5	44	0
	10^{-10}	**-34%**	**-2%**	0.8s	**-34%**	**-2%**	0.9	0	0	48	1
	10^{-12}	**-57%**	**-11%**	0.9s	**-57%**	**-11%**	1.0s	0	0	44	0

Parameters

- **TH**: Error threshold
- **IEEE** Optimization in IEEE754
- **H, S**: FP16, FP32 precision
- **BL**: Optimization at Bit level
- **ILP-time/PI-time**: Analysis time
- **D, LD**: FP64, FP128 precision

[18] More examples given in Chapter 9 Table 9.1

Precision Tuning Results for ILP and PI Methods[18]

Program	TH	BL	IEEE	ILP-time	BL	IEEE	PI-time	H	S	D	LD
accelerometer	10^{-4}	73%	61%	0.2s	76%	62%	1.0s	53	69	0	0
	10^{-6}	62%	55%	0.2s	65%	55%	1.0s	2	102	0	0
	10^{-8}	49%	15%	0.2s	52%	18%	1.0s	2	33	69	0
	10^{-10}	36%	1%	0.2s	39%	1%	1.0s	2	0	102	0
	10^{-12}	25%	1%	0.2s	28%	1%	1.0s	2	0	102	0
lowPassFilter	10^{-4}	68%	46%	1.8s	69%	46%	10.7s	260	581	0	0
	10^{-6}	57%	38%	1.8s	58%	45%	11.0s	258	580	3	0
	10^{-8}	44%	-7%	2.0s	45%	-7%	11.4s	258	2	581	0
	10^{-10}	31%	-7%	1.7s	32%	-7%	10.9s	258	0	583	0
	10^{-12}	20%	-7%	1.8s	21%	-7%	11.3s	258	0	583	0
2-Body	10^{-4}	51%	41%	0.81s	51%	41%	0.82s	5	39	5	0
	10^{-6}	49%	18%	0.78s	49%	18%	0.9s	0	44	5	0
	10^{-8}	-7%	5%	0.8s	-7%	5%	0.78s	0	5	44	0
	10^{-10}	-34%	-2%	0.8s	-34%	-2%	0.9	0	0	48	1
	10^{-12}	-57%	-11%	0.9s	-57%	-11%	1.0s	0	0	44	0

Main observations

- More important BL when coupling POP(ILP) with PI technique
- PI-time > ILP-Time $\longrightarrow$ more constraints to solve
- Ability of POP(ILP) to return new formats for any threshold

[18] More examples given in Chapter 9 Table 9.1

POP(ILP) vs POP(SMT) vs Precimonious[20]

Program	Tool (LOC)	#Bits saved - Time in seconds			
		Threshold 10^{-4}	Threshold 10^{-6}	Threshold 10^{-8}	Threshold 10^{-10}
arclength	POP(ILP) (28)	**2464b.** - 1.8s.	**2144b.** - 1.5s.	**1792b.** - 1.7s.	**1728b.** - 1.8s.
	POP(SMT) (22)	1488b. - 4.7s.	1472b. - 3.04s.	864b. - 3.09s.	384b. - 2.9s.
	Precimonious (9)	576b. - 146.4s.	576b. - 156.0s.	576b. - 145.8s.	576b. - 215.0s.
simpson	POP(ILP) (14)	**1344b.** - 0.4s.	**1152b.** - 0.5s.	**896b.** - 0.4s.	**896b.** - 0.4s.
	POP(SMT) (11)	896b. - 2.9s.	896b. - 1.9s.	704b. - 1.7s.	704b. - 1.8s.
	Precimonious (10)	704b. - 208.1s.	704b. - 213.7s.	704b. - 207.5s.	704b. - 200.3s.
rotation	POP(ILP) (25)	**2624b.** - 0.47s.	2464b. - 0.47s.	2048b. - 0.54s.	1600b. - 0.48s.
	POP(SMT) (22)	1584b. - 1.85s.	2208b. - 1.7s.	1776b. - 1.6s.	1600b. - 1.7s.
	Precimonious (27)	2400b. - 9.53s.	**2592b.** - 12.2s.	**2464b.** - 10.7s.	**2464b.** - 7.4s.
accel.	POP(ILP) (18)	**1776b.** - 1.05s.	**1728b.** - 1.05s.	**1248b.** - 1.04s.	**1152b.** - 1.03s.
	POP(SMT) (15)	1488b. - 2.6s.	1440b. - 2.6s.	1056b - 2.4s.	960b. - 2.4s.
	Precimonious (0)	-	-	-	-

Adjusting comparison criteria

- Consider **only** variables optimized by Precimonious
- Express all the error thresholds in base 10
- . . .[19]

[19] More criteria have been presented in Chapter 8 Section 8.3

[20] C. Rubio González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, D. Hough. *"Precimonious: tuning assistant for floating-point precision"*. SC'13.

POP(ILP) vs POP(SMT) vs Precimonious[19]

Program	Tool (LOC)	#Bits saved - Time in seconds			
		Threshold 10^{-4}	Threshold 10^{-6}	Threshold 10^{-8}	Threshold 10^{-10}
arclength	POP(ILP) (28)	**2464b.** - 1.8s.	**2144b.** - 1.5s.	**1792b.** - 1.7s.	**1728b.** - 1.8s.
	POP(SMT) (22)	1488b. - 4.7s.	1472b. - 3.04s.	864b. - 3.09s.	384b. - 2.9s.
	Precimonious (9)	576b. - 146.4s.	576b. - 156.0s.	576b. - 145.8s.	576b. - 215.0s.
simpson	POP(ILP) (14)	**1344b.** - 0.4s.	**1152b.** - 0.5s.	**896b.** - 0.4s.	**896b.** - 0.4s.
	POP(SMT) (11)	896b. - 2.9s.	896b. - 1.9s.	704b. - 1.7s.	704b. - 1.8s.
	Precimonious (10)	704b. - 208.1s.	704b. - 213.7s.	704b. - 207.5s.	704b. - 200.3s.
rotation	POP(ILP) (25)	**2624b.** - 0.47s.	2464b. - 0.47s.	2048b. - 0.54s.	1600b. - 0.48s.
	POP(SMT) (22)	1584b. - 1.85s.	2208b. - 1.7s.	1776b. - 1.6s.	1600b. - 1.7s.
	Precimonious (27)	2400b. - 9.53s.	**2592b.** - 12.2s.	**2464b.** - 10.7s.	**2464b.** - 7.4s.
accel.	POP(ILP) (18)	**1776b.** - 1.05s.	**1728b.** - 1.05s.	**1248b.** - 1.04s.	**1152b.** - 1.03s.
	POP(SMT) (15)	1488b. - 2.6s.	1440b. - 2.6s.	1056 - 2.4s.	960b. - 2.4s.
	Precimonious (0)	-	-	-	-

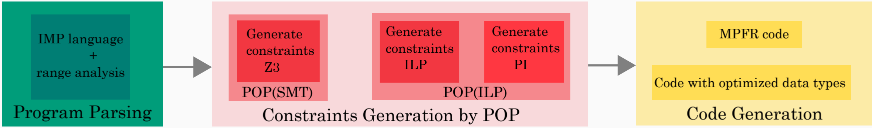
Main observations

- POP(ILP) saves more bits in fewer time
- Precimonious displays better results on the rotation example
- Precimonious fails to tune some benchmarks of POP

[19] C. Rubio González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, D. Hough. *"Precimonious: tuning assistant for floating-point precision"*. SC'13.

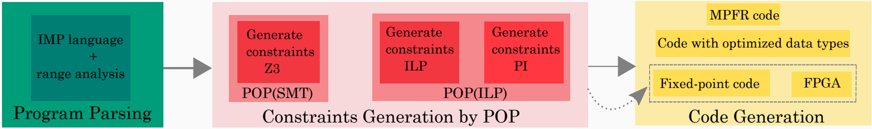
Table of Contents

Takeaways



- A new tool for precision tuning with three variants of method
  - Forward and backward error analysis checked by SMT solver
  - Pure ILP with an over-approximation of the carry functions
  - PI technique for more precise carry bit function
- Fast and efficient bit-level precision tuning
- There are still challenges to apply precision tuning **at scale**

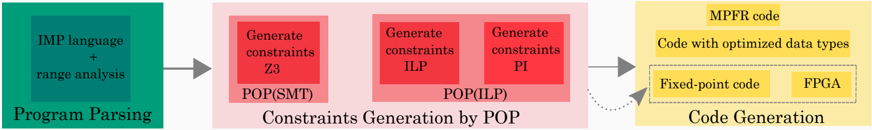
Source code available at
https://github.com/benkhelifadorra/POP-v2.0



First research direction

- Synthesis of fixed-point programs[20]
- Satisfy the accuracy guarantee on the results
- POP computes at each point of this program the pair $|m, s|$
  - m: weight of the most significant bits
  - s: number of significants (nsb)
  - Shifts ?
- Synthesis of VHDL code for FPGA

[20] Daniele Cattaneo et al. *"Fixed point exploitation via compiler analyses and transformations: POSTER"*. CF'19



Scalability

- Reduce number of variables and constraints
- Explore commercial LP solvers for larger ILP problems

Extension

- Include functions in POP
- Combining POP with rewriting tools to improve accuracy[21]
- Precision tuning for DNN's and GPU applications

[21] N. Damouche, M. Martel. *"Mixed Precision Tuning with Salsa"*. PECCS'18

Thank You...

Merci...

شكراً...

List of Publications

[1] Assalé Adjé, Dorra Ben Khalifa, and Matthieu Martel.
Fast and Efficient Bit-Level Precision Tuning.
In *Static Analysis - 28th International Symposium, 2021*.

[2] Dorra Ben Khalifa and Matthieu Martel.
An Evaluation of POP Performance for Tuning Numerical Programs in Floating-Point Arithmetic.
In *IEEE International Conference on Information and Computer Technologies, 2021*.

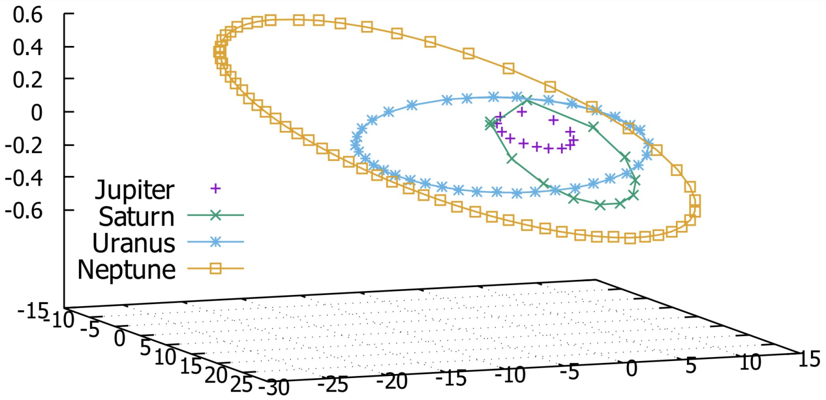
[3] Dorra Ben Khalifa and Matthieu Martel.
Precision Tuning and Internet of Things.
In *IEEE International Conference on Internet of Things, Embedded Systems and Communications, 2019*.

[4] Dorra Ben Khalifa and Matthieu Martel.
Precision Tuning of an Accelerometer-Based Pedometer Algorithm for IoT Devices.
In *IEEE International Conference on Internet of Things and Intelligence System, 2020*.

[5] Dorra Ben Khalifa and Matthieu Martel.
A Study of the Floating-Point Tuning Behaviour on the N-Body Problem.
In *International Conference on Computational Science and Its Applications, 2021*. Springer.

[6] Dorra Ben Khalifa, Matthieu Martel, and Assalé Adjé.
POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations.
In *Formal Techniques for Safety-Critical Systems - 7th International Workshop, 2019*.

A Study on the N-Body Problem by POP(ILP)[22]



- Simulation of the orbits of planets interacting with each other gravitationally

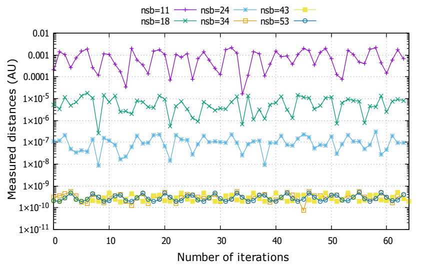
[22] D. Ben Khalifa and M. Martel. *"A study of the floating-point Tuning Behaviour on the N-body Problem."ICCSA'21*

$dist(x, \widehat{x})$ for a Single Position Planets

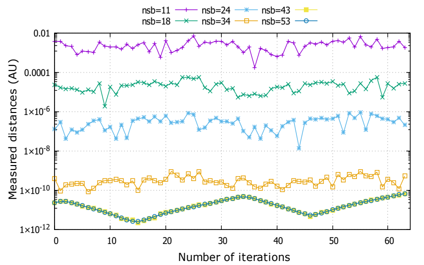
nsb	11	18	24	34	43	53
			Simulation time: 10 years			
Jupiter	$5.542 \cdot 10^{-4}$	$1.650 \cdot 10^{-6}$	$1.577 \cdot 10^{-7}$	$4.998 \cdot 10^{-10}$	$5.077 \cdot 10^{-10}$	$5.076 \cdot 10^{-10}$
Saturn	$1.571 \cdot 10^{-3}$	$2.111 \cdot 10^{-5}$	$1.326 \cdot 10^{-7}$	$4.427 \cdot 10^{-10}$	$3.119 \cdot 10^{-10}$	$3.117 \cdot 10^{-10}$
Uranus	$2.952 \cdot 10^{-3}$	$2.364 \cdot 10^{-5}$	$1.140 \cdot 10^{-7}$	$3.072 \cdot 10^{-10}$	$7.212 \cdot 10^{-11}$	$7.236 \cdot 10^{-11}$
Neptune	$2.360 \cdot 10^{-3}$	$3.807 \cdot 10^{-5}$	$2.206 \cdot 10^{-7}$	$5.578 \cdot 10^{-10}$	$1.751 \cdot 10^{-10}$	$1.757 \cdot 10^{-10}$
Runtime	2’59	2’52	2’57	2’56	3’10	2’59
POP Time	25”	22”	22”	24”	23”	24”

- ☞ Satisfactory results which respect the user defined error
- ☞ POP(ILP) analysis time $\approx$ 25 seconds

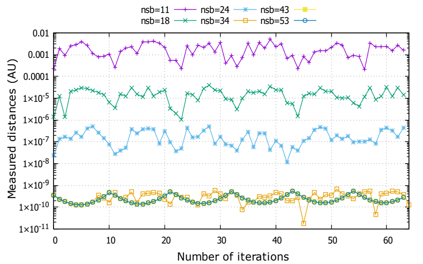
$dist(x, \widehat{x})$ at each Instant of Simulation



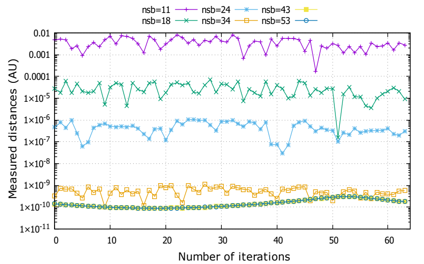
Jupiter



Uranus

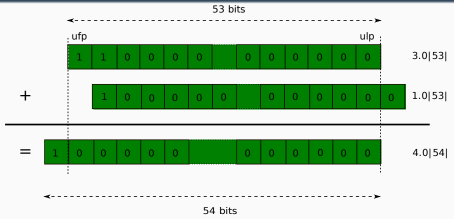


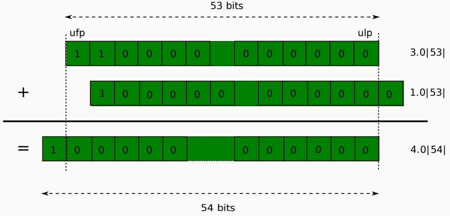
Saturn



Neptune

Transfer Functions Concrete Addition





Forward addition

$$\overrightarrow{\oplus}(x_p, y_q) = z_r$$

$$r = \text{ufp}(x + y) - \text{ufp}(\varepsilon_{x_p} + \varepsilon_{y_q} + \varepsilon_+)$$

example $\overrightarrow{\oplus}(3.0|53|, 1.0|53|) = 4.0|54|$

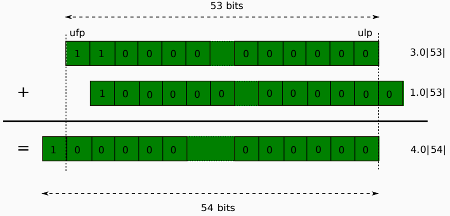
Backward addition

$$\overleftarrow{\oplus}(z_r, y_q) = (z - y)_p$$

$$p = \text{ufp}(z - y) - \text{ufp}(\varepsilon_{z_r} - \varepsilon_{y_q} - \varepsilon_+)$$

example $\overleftarrow{\oplus}(4.0|23|, 1.0|53|) = 3.0|25|$

Transfer Functions Concrete Addition



Forward addition

$$\overrightarrow{\oplus}(x_p, y_q) = z_r$$

$$r = \text{ufp}(x + y) - \text{ufp}(\varepsilon_{x_p} + \varepsilon_{y_q} + \varepsilon_+)$$

example $\overrightarrow{\oplus}(3.0|53|, 1.0|53|) = 4.0|54|$

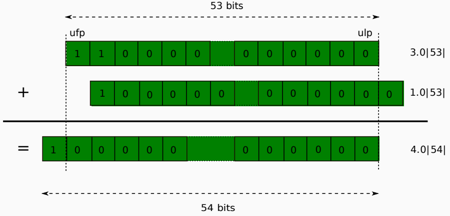
Backward addition

$$\overleftarrow{\oplus}(z_r, y_q) = (z - y)_p$$

$$p = \text{ufp}(z - y) - \text{ufp}(\varepsilon_{z_r} - \varepsilon_{y_q} - \varepsilon_+)$$

example $\overleftarrow{\oplus}(4.0|23|, 1.0|53|) = 3.0|25|$

- **Truncation error**: $\varepsilon_+ \leq 2^{\text{ufp}(z) - \text{prec}(+)}$ ($\text{prec}(+)$ precision of the operator $+$)



Forward addition

$$\overrightarrow{\oplus}(x_p, y_q) = z_r$$

$$r = \text{ufp}(x+y) - \text{ufp}(\varepsilon_{x_p} + \varepsilon_{y_q} + \varepsilon_+)$$

example $\overrightarrow{\oplus}(3.0|53|, 1.0|53|) = 4.0|54|$

Backward addition

$$\overleftarrow{\oplus}(z_r, y_q) = (z-y)_p$$

$$p = \text{ufp}(z-y) - \text{ufp}(\varepsilon_{z_r} - \varepsilon_{y_q} - \varepsilon_+)$$

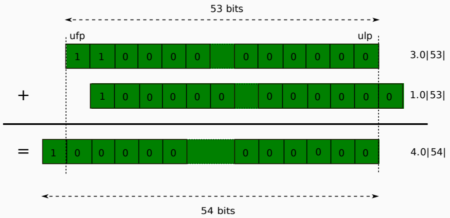
example $\overleftarrow{\oplus}(4.0|23|, 1.0|53|) = 3.0|25|$

- **Truncation error**: $\varepsilon_+ \leq 2^{\text{ufp}(z)-\text{prec}(+)}$ (prec$(+)$ precision of the operator $+$)

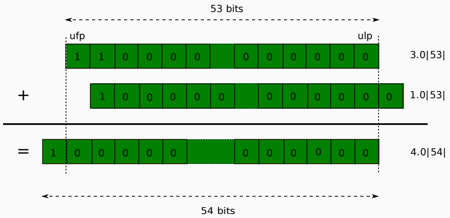
☞ Same for the multplication, subtraction and division

☞ Technique generalizable to set of values

Constraints SMT Forward Addition Case



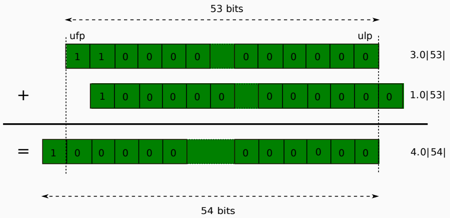
$$\varepsilon_z \leq 2^{\mathrm{ufp}(x)-\mathrm{nsb}_F(x)+1} + 2^{\mathrm{ufp}(y)-\mathrm{nsb}_F(y)+1} + 2^{\mathrm{ufp}(z)-\mathrm{prec}(+)}$$



$$\varepsilon_z \leq 2^{\mathrm{ufp}(x)-\mathrm{nsb}_F(x)+1} + 2^{\mathrm{ufp}(y)-\mathrm{nsb}_F(y)+1} + 2^{\mathrm{ufp}(z)-\mathrm{prec}(+)}$$

$$\mathrm{ufp}(\varepsilon_z) \leq \max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) + \mathit{carry}(z, x, y)$$

Constraints SMT Forward Addition Case



$$\varepsilon_z \leq 2^{\mathrm{ufp}(x)-\mathrm{nsb}_F(x)+1} + 2^{\mathrm{ufp}(y)-\mathrm{nsb}_F(y)+1} + 2^{\mathrm{ufp}(z)-\mathrm{prec}(+)}$$

$$\mathrm{ufp}(\varepsilon_z) \leq \max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) + \mathit{carry}(z, x, y)$$

$$\mathrm{nsb}_F(z) = \mathrm{ufp}(x+y) - \max(\mathrm{ufp}(x) - \mathrm{nsb}_F(x), \mathrm{ufp}(y) - \mathrm{nsb}_F(y), \mathrm{ufp}(z) - \mathrm{prec}(+)) - \mathit{carry}$$

example $\overrightarrow{\oplus}(3.0|53|, 1.0|53|) = 4.0|54|$

$$\mathrm{nsb}_B(x) = \mathrm{ufp}(z - y) - \mathrm{ufp}(z) + \mathrm{nsb}(z)$$

example $\overleftarrow{\oplus}(4.0|23|, 1.0|53|) = 3.0|25|$

$$0 \leq \mathrm{nsb}_B(\ell) \leq \mathrm{nsb}(\ell) \leq \mathrm{nsb}_F(\ell), \quad \forall \ell \in \mathit{Lab}$$

- **Forward multiplication** $\overrightarrow{\otimes}$ between x and y in z

  $\overrightarrow{\otimes}(x, y) = z$ where $\mathrm{nsb}_F(z) = \mathrm{ufp}(x \times y) - \mathrm{ufp}(y \cdot \varepsilon(x) + x \cdot \varepsilon(y) + \varepsilon(x) \cdot \varepsilon(x) + \varepsilon_\times)$

example $\overrightarrow{\otimes}(4.0|53|, 1.0|53|) = 4.0|53| \Longrightarrow \mathrm{nsb}_F(z) = 53$

[23] D. Ben Khalifa, M. Martel and A. Adjé. *"POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations"*. FTSCS'19

- **Forward multiplication** $\overrightarrow{\otimes}$ between x and y in z

  $\overrightarrow{\otimes}(x,y) = z$ where $\mathrm{nsb}_F(z) = \mathrm{ufp}(x \times y) - \mathrm{ufp}(y \cdot \varepsilon(x) + x \cdot \varepsilon(y) + \varepsilon(x) \cdot \varepsilon(x) + \varepsilon_\times)$

example $\overrightarrow{\otimes}(4.0|53|, 1.0|53|) = 4.0|53| \Longrightarrow \mathrm{nsb}_F(z) = 53$

- **Backward multiplication** $\overleftarrow{\otimes}$ between x and y in z

  $\overleftarrow{\otimes}(z,y) = (z \div y)$ where $\mathrm{nsb}(x) = \mathrm{ufp}(z \div y) - \mathrm{ufp}\left(\dfrac{y \cdot \varepsilon(z) - z \cdot \varepsilon(y)}{y \cdot (y + \varepsilon(y))} - \varepsilon_\times\right)$

example $\overleftarrow{\otimes}(4.0|23|, 1.0|53|) = 4.0|25| \Longrightarrow \mathrm{nsb}_B(z) = 53$

[23] D. Ben Khalifa, M. Martel and A. Adjé. *"POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations"*. FTSCS'19

- **Forward multiplication** $\overrightarrow{\otimes}$ between x and y in z

$$\overrightarrow{\otimes}(x,y) = z \text{ where } \mathrm{nsb}_F(z) = \mathrm{ufp}(x \times y) - \mathrm{ufp}(y \cdot \varepsilon(x) + x \cdot \varepsilon(y) + \varepsilon(x) \cdot \varepsilon(x) + \varepsilon_\times)$$

example $\overrightarrow{\otimes}(4.0|53|, 1.0|53|) = 4.0|53| \Longrightarrow \mathrm{nsb}_F(z) = 53$

- **Backward multiplication** $\overleftarrow{\otimes}$ between x and y in z

$$\overleftarrow{\otimes}(z,y) = (z \div y) \text{ where } \mathrm{nsb}(x) = \mathrm{ufp}(z \div y) - \mathrm{ufp}\left(\frac{y \cdot \varepsilon(z) - z \cdot \varepsilon(y)}{y \cdot (y + \varepsilon(y))} - \varepsilon_\times\right)$$

example $\overleftarrow{\otimes}(4.0|23|, 1.0|53|) = 4.0|25| \Longrightarrow \mathrm{nsb}_B(z) = 53$

Generalization

- ☞ Same for the subtraction and division
- ☞ Technique generalizable to set of values[23]

[23] D. Ben Khalifa, M. Martel and A. Adjé. *"POP: A Tuning Assistant for Mixed-Precision Floating-Point Computations"*. FTSCS'19

POP vs Precimonious[24]

POP (both versions)

- Constraint generation by static analysis
- Optimized formats given by solvers
- Mixed-precision: FP8, FP16, FP32, FP64, FPxx
- Programmer accuracy requirement
- Supports arrays, conditions, loops, no function yet

Precimonious

- Dynamic analysis by delta-debugging search
- Type configurations rely on inputs tested **only**
- FP32 and FP64
- Given programmer error threshold (10^{-4}, 10^{-6}, 10^{-8}, ...)
- C program input

[24] C. Rubio González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, D. Hough. *"Precimonious: tuning assistant for floating-point precision"*. SC'13.