University of Perpignan Via Domitia
LAMPS Laboratory, France

Managing Performance vs. Accuracy Trade-offs with an Improved Bit-Level Precision Tuning

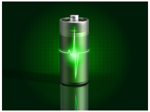
CMMSE 2021

Assalé Adjé **Dorra Ben Khalifa** **Matthieu Martel**
dorra.ben-khalifa@univ-perp.fr

July 25, 2021

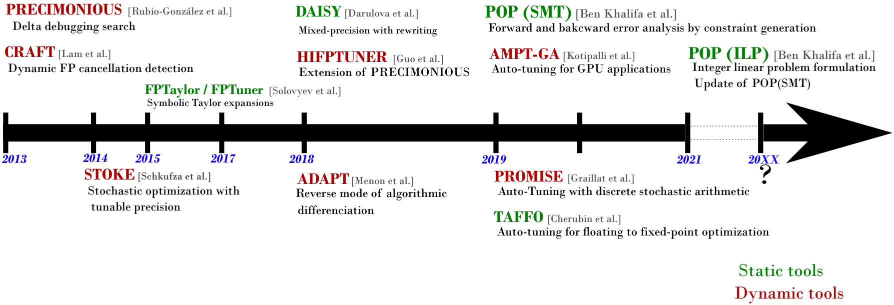
Motivation

- Higher-precision data formats, *e.g. binary*64 *and binary*128
  - \+ More accurate/robust
  - – Higher execution time, increased memory / energy consumption



- **Challenge:** Best trade-offs between accuracy and performance
- **Precision Tuning:** lowering a computational kernel's precision up to a user accuracy requirement
- **Idea:** develop automated techniques to assist in precision tuning

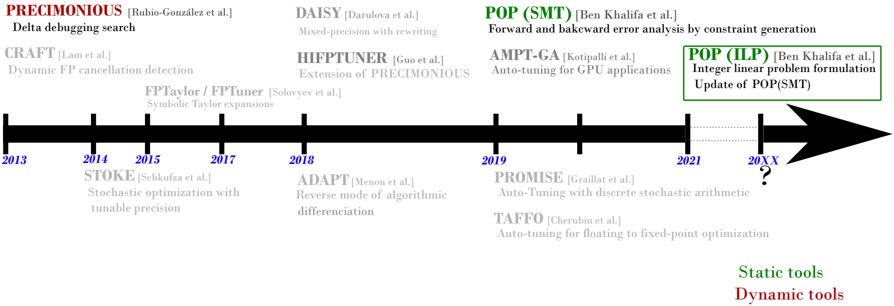
Precision Tuning: A Survey



- Try and fail strategies: change more or less randomly the data types and run the program

[S. Cherubin and G. Agosta. Tools for Reduced Precision Computation: A Survey. In ACM Computing Surveys'20]

Precision Tuning: A Survey

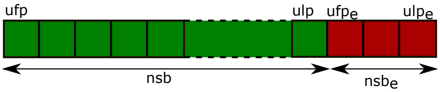


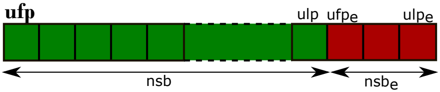
[D. Ben Khalifa, M. Martel, and A. Adjé. POP : A Tuning Assistant for Mixed-Precision Floating-Point Computations (FTSCS'19)]

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

Preliminary Notations and Definitions

ufp, nsb, ulp and computation errors



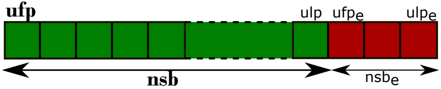


- The **unit in the first place** of a real number x

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

Preliminary Notations and Definitions

ufp, nsb, ulp and computation errors



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

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

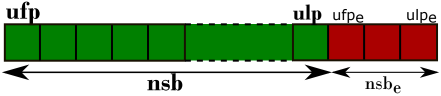
- $\mathrm{nsb}(x)$: number of significant bits of x
  - $\hat{x}$: approximation of x in finite precision
  - $\varepsilon(x) = |x - \hat{x}|$: the absolute error

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

[Parker, D.S.: Monte carlo arithmetic: exploiting randomness in floating-point arithmetic. Tech. Rep. CSD-970002, University of California'97]

Preliminary Notations and Definitions

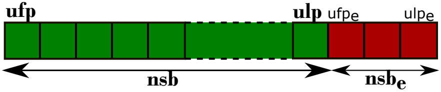
ufp, nsb, ulp and computation errors



- The **unit in the last place** of x: $\text{ulp}(x) = \text{ufp}(x) - \text{nsb}(x) + 1$
- $\text{ufp}_e(x) = \text{ufp}(x) - \text{nsb}(x)$

Preliminary Notations and Definitions

ufp, nsb, ulp and computation errors

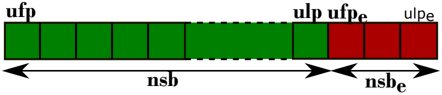


- The **unit in the last place** of x: $\mathrm{ulp}(x) = \mathrm{ufp}(x) - \mathrm{nsb}(x) + 1$
- $\mathrm{ufp_e}(x) = \mathrm{ufp}(x) - \mathrm{nsb}(x)$
- $\mathrm{nsb_e}$: number of significant bits of the computation error on x
- $\mathrm{nsb_e}(x)$ is computed as follows:
  - For a constant c, $\mathrm{nsb_e}(c) = 0$ (constants assumed exact)
  - $x^{\ell} = c_1^{\ell_1} \odot c_2^{\ell_2}$ with $\mathrm{ufp_e}(c_1) \geq \mathrm{ufp_e}(c_2)$, $\odot \in \{+, -, \times, \div\}$

$$\mathrm{nsb_e}(x) = \mathrm{ufp_e}(c_1) - (\mathrm{ufp_e}(c_2) - \mathrm{nsb_e}(c_2) + 1)$$

Preliminary Notations and Definitions

ufp, nsb, ulp and computation errors

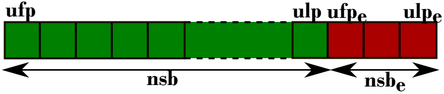


- The **unit in the last place** of x: $\text{ulp}(x) = \text{ufp}(x) - \text{nsb}(x) + 1$
- $\text{ufp}_e(x) = \text{ufp}(x) - \text{nsb}(x)$
- nsb_e: number of significant bits of the computation error on x
- $\text{nsb}_e(x)$ is computed as follows:
  - For a constant c, $\text{nsb}_e(c) = 0$ (constants assumed exact)
  - $x^{\ell} = c_1^{\ell_1} \odot c_2^{\ell_2}$ with $\text{ufp}_e(c_1) \geq \text{ufp}_e(c_2)$, $\odot \in \{+, -, \times, \div\}$

$$\text{nsb}_e(x) = \text{ufp}_e(c_1) - (\text{ufp}_e(c_2) - \text{nsb}_e(c_2) + 1)$$

Preliminary Notations and Definitions

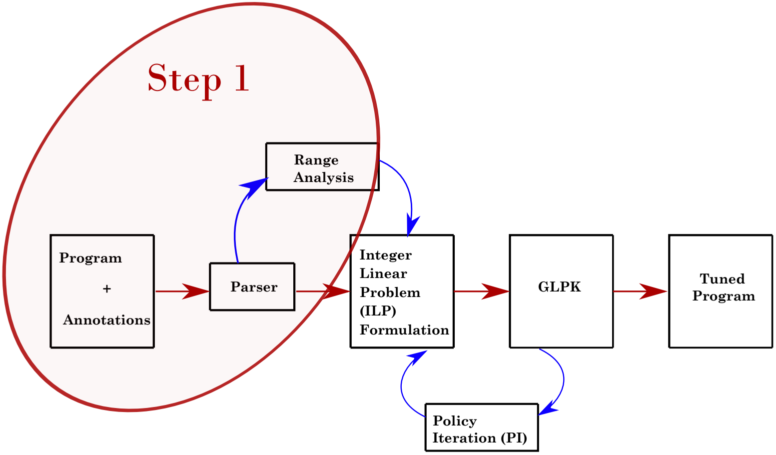
ufp, nsb, ulp and computation errors



- The **unit in the last place** of x: $\mathrm{ulp}(x) = \mathrm{ufp}(x) - \mathrm{nsb}(x) + 1$
- $\mathrm{ufp_e}(x) = \mathrm{ufp}(x) - \mathrm{nsb}(x)$
- $\mathrm{nsb_e}$: number of significant bits of the computation error on x
- $\mathrm{nsb_e}(x)$ is computed as follows:
  - For a constant c, $\mathrm{nsb_e}(c) = 0$ (constants assumed exact)
  - $x^{\ell} = c_1^{\ell_1} \odot c_2^{\ell_2}$ with $\mathrm{ufp_e}(c_1) \geq \mathrm{ufp_e}(c_2)$, $\odot \in \{+, -, \times, \div\}$

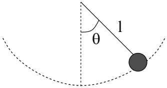
$$\mathrm{nsb_e}(x) = \mathrm{ufp_e}(c_1) - (\mathrm{ufp_e}(c_2) - \mathrm{nsb_e}(c_2) + 1)$$

- $\mathrm{ulp_e}(x) = \mathrm{ufp_e}(x) - \mathrm{nsb_e}(x) + 1$


Step 1
Range Analysis
Program + Annotations
Parser
Integer Linear Problem (ILP) Formulation
GLPK
Tuned Program
Policy Iteration (PI)

Parsing and Range Determination Phase

Pendulum Example

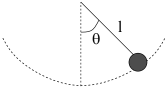



```
g = 9.81; l = 0.5;
y1 = 0.785398;
y2 =  0.785398;
h = 0.1; t = 0.0;
while (t<10.0) {
 y1new = y1 + y2 * h ;
 aux1 = sin(y1) ;
 aux2 = aux1 * h * g / l;
 y2new = y2 - aux2;
  t = t + h;
 y1 = y1new;
 y2 = y2new;
};
require_nsb(y2, 20);
```

POP Source File

Parsing and Range Determination Phase

Pendulum Example




```
g = 9.81; l = 0.5;
y1 = 0.785398;
y2 =  0.785398;
h = 0.1; t = 0.0;
while (t<10.0) {
 y1new = y1 + y2 * h ;
 aux1 = sin(y1) ;
 aux2 = aux1 * h * g / l;
 y2new = y2 - aux2;
 t = t + h;
 y1 = y1new;
 y2 = y2new;
};
require_nsb(y2, 20);
```

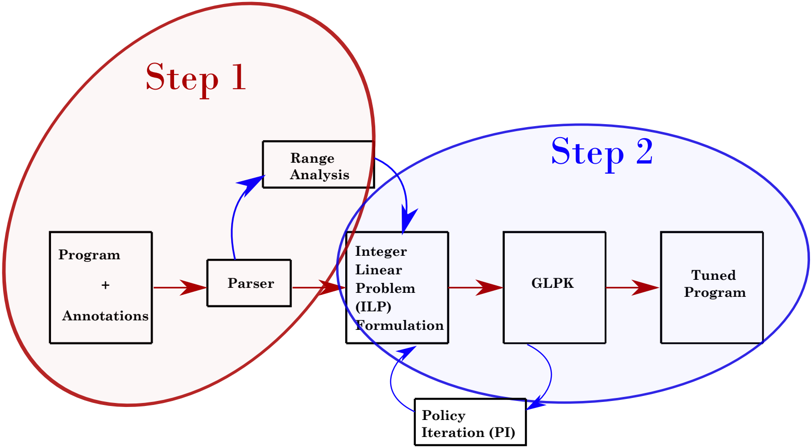
POP Source File

```
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

Constraints Generation for Bit-Level Tuning

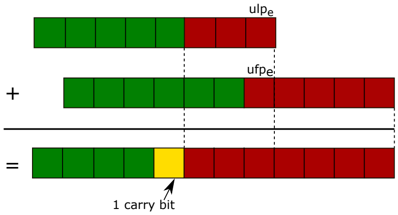
Step 2: ILP Formulation



Constraints Generation for Bit-Level Tuning

ILP Formulation with Pessimistic Carry Bit Propagation

- Static technique relying on semantical modelling of the propagation of numerical errors
- Reasoning on ufp (known at constraint generation time) and nsb (unknown) of the program values
- Generate an ILP from the program source code
- Optimally solved by a classical LP solver in **polynomial time**
- Pessimistic carry bit function: $\xi = 1$



Constraints Generation for Bit-Level Tuning

Pendulum Program (ILP)


```
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

Constraints Generation for Bit-Level Tuning

Pendulum Program (ILP)


```
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

$$C_1 = \begin{cases} \mathrm{nsb}(\ell_{17}) \geq \mathrm{nsb}(\ell_{23}) + (-1) + \xi(\ell_{23})(\ell_{17}, \ell_{22}) - (-1) & //\textsc{Add} \\ \mathrm{nsb}(\ell_{22}) \geq \mathrm{nsb}(\ell_{23}) + 0 + \xi(\ell_{23})(\ell_{17}, \ell_{22}) - (1) & //\textsc{Add} \\ \mathrm{nsb}(\ell_{19}) \geq \mathrm{nsb}(\ell_{22}) + \xi(\ell_{22})(\ell_{19}, \ell_{21}) - 1 & //\textsc{Mult} \\ \mathrm{nsb}(\ell_{21}) \geq \mathrm{nsb}(\ell_{22}) + \xi(\ell_{22})(\ell_{19}, \ell_{21}) - 1 & //\textsc{Mult} \\ \mathrm{nsb}(\ell_{23}) \geq \mathrm{nsb}(\ell_{24}) & //\textsc{Assign} \\ \xi(\ell_{23})(\ell_{17}, \ell_{22}) \geq 1, \xi(\ell_{22})(\ell_{19}, \ell_{21}) \geq 1 & //\textsc{Carry bit} \end{cases}$$

Constraints Generation for Bit-Level Tuning

Pendulum Program (ILP)

$g^{\ell_1} = 9.81^{\ell_0};\ l^{\ell_3} = 0.5^{\ell_2};$

$y1^{\ell_5} = 0.785398^{\ell_4};$

$y2^{\ell_7} = 0.785398^{\ell_6};$

$h^{\ell_9} = 0.1^{\ell_8};\ t^{\ell_{11}} = 0.0^{\ell_{10}};$

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

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

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

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

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

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

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

$y1^{\ell_{55}} = y1new^{\ell_{54}};$

$y2^{\ell_{58}} = y2new^{\ell_{57}};\ \};$

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

POP Label File

$$C_1 = \begin{cases} \mathrm{nsb}(\ell_{17}) \geq \mathrm{nsb}(\ell_{23}) + (-1) + \xi(\ell_{23})(\ell_{17},\ell_{22}) - (-1) & //\textsc{Add} \\ \mathrm{nsb}(\ell_{22}) \geq \mathrm{nsb}(\ell_{23}) + 0 + \xi(\ell_{23})(\ell_{17},\ell_{22}) - (1) & //\textsc{Add} \\ \mathrm{nsb}(\ell_{19}) \geq \mathrm{nsb}(\ell_{22}) + \xi(\ell_{22})(\ell_{19},\ell_{21}) - 1 & //\textsc{Mult} \\ \mathrm{nsb}(\ell_{21}) \geq \mathrm{nsb}(\ell_{22}) + \xi(\ell_{22})(\ell_{19},\ell_{21}) - 1 & //\textsc{Mult} \\ \mathrm{nsb}(\ell_{23}) \geq \mathrm{nsb}(\ell_{24}) & //\textsc{Assign} \\ \xi(\ell_{23})(\ell_{17},\ell_{22}) \geq 1, \xi(\ell_{22})(\ell_{19},\ell_{21}) \geq 1 & //\textsc{Carry bit} \end{cases}$$

- Constraints generated for **all the statements** of POP programs: `if`, `while`, `for`, `array`, `sqrt`, `sin`, etc.
- Linear number of constraints / variables in the size of the analyzed program

[Nielson, F. and Nielson, H. R., and Hankin, C. Principles of Program Analysis. Springer-Verlag, Berlin, Heidelberg'99]

Tuned Pendulum Program (ILP)


```
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

- 2385 bits (originally) VS 861 bits at bit level after POP analysis

Tuned Pendulum Program (ILP)

```
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);
```

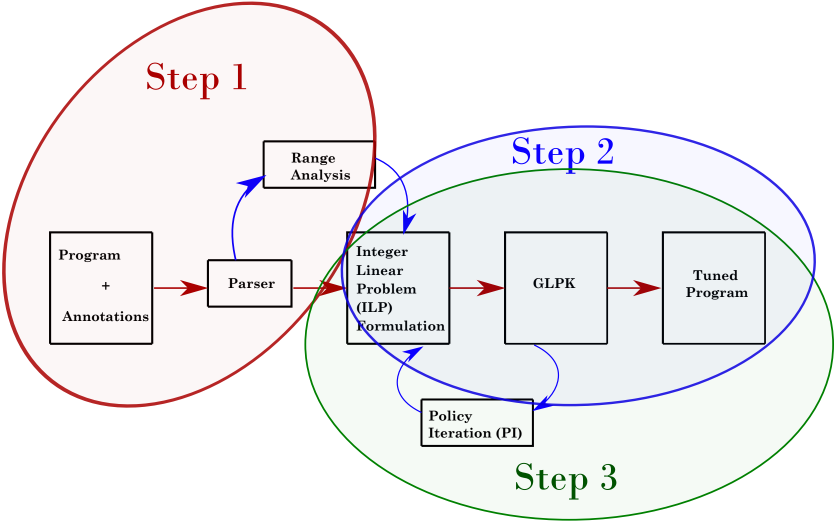
File pop_output

- 2385 bits (originally) VS 861 bits at bit level after POP analysis

How to be less pessimistic on carries propagation?

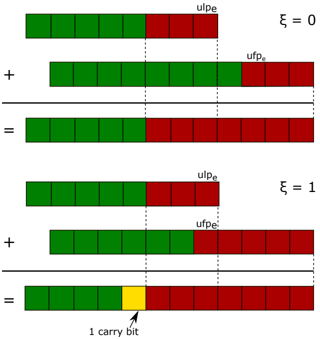
Constraints Generation for Bit-Level Tuning

Step 3: PI for Optimized Carry Bit Propagation



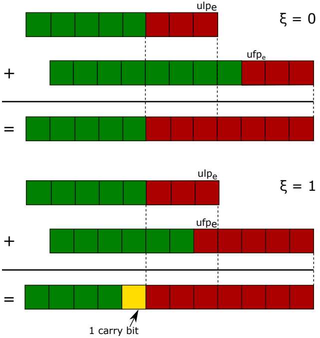
Constraints Generation for Bit-Level Tuning

Step 3: PI for Optimized Carry Bit Propagation



Constraints Generation for Bit-Level Tuning

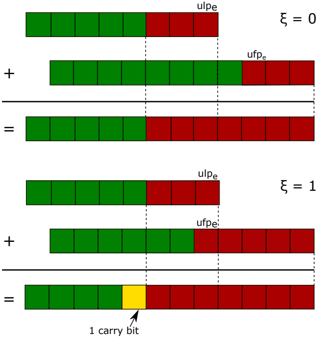
Step 3: PI for Optimized Carry Bit Propagation



- Very costly over-approximation of ξ in ILP formulation

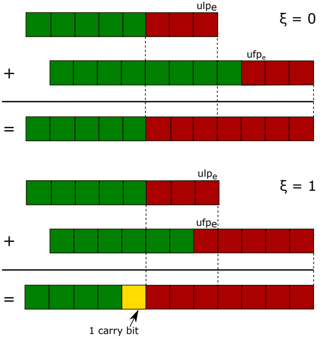
Constraints Generation for Bit-Level Tuning

Step 3: PI for Optimized Carry Bit Propagation



- Very costly over-approximation of ξ in ILP formulation
- $x^{\ell} = c_1^{\ell_1} + c_2^{\ell_2}$

$$\xi(\ell, \ell_1, \ell_2) = \begin{cases} 0 & \mathrm{ulp_e}(\ell_1) > \mathrm{ufp_e}(\ell_2) \\ 0 & \mathrm{ulp_e}(\ell_2) > \mathrm{ufp_e}(\ell_1) \\ 1 & \text{otherwise} \end{cases}$$



- Very costly over-approximation of ξ in ILP formulation
- $x^{\ell} = c_1^{\ell_1} + c_2^{\ell_2}$

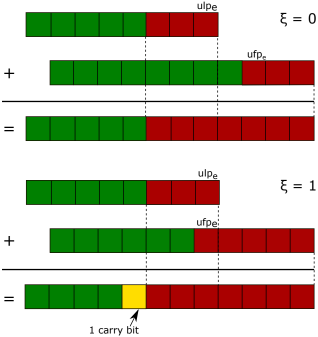
$$\xi(\ell, \ell_1, \ell_2) = \begin{cases} 0 & \mathrm{ulp_e}(\ell_1) > \mathrm{ufp_e}(\ell_2) \\ 0 & \mathrm{ulp_e}(\ell_2) > \mathrm{ufp_e}(\ell_1) \\ 1 & \text{otherwise} \end{cases}$$

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

- Need to estimate the integer quantity $\mathrm{nsb_e}$

Constraints Generation for Bit-Level Tuning

Step 3: PI for Optimized Carry Bit Propagation



- Very costly over-approximation of ξ in ILP formulation
- $x^{\ell} = c_1^{\ell_1} + c_2^{\ell_2}$

$$\xi(\ell, \ell_1, \ell_2) = \begin{cases} 0 & \mathsf{ulp_e}(\ell_1) > \mathsf{ufp_e}(\ell_2) \\ 0 & \mathsf{ulp_e}(\ell_2) > \mathsf{ufp_e}(\ell_1) \\ 1 & \text{otherwise} \end{cases}$$

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

- Need to estimate the integer quantity $\mathsf{nsb_e}$

No longer ILP formulation!

Constraints Generation for Bit-Level Tuning

Policy Iteration to Refine Carry Bit Propagation

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

$$\xi(\ell)(\ell_1,\ell_2) = \min\left(\begin{array}{l}\max\left(\pi_{11},\pi_{12}\right),\\ \max\left(\pi_{21},\pi_{22}\right),\pi_{31}\end{array}\right)$$

- **Principle:**
  - Choose a policy π_0 by breaking the $\min - \max$ equations
    - e.g. $\pi_0 = (\pi_{11},\pi_{22}),\ldots.$
  - Associate f^{π_0} for π_0
    - e.g. $f^{\pi_0} = (\pi_{11},\pi_{22})$
  - Solve corresponding ILP problem
  - If no fixed point is reached, POP iterates until a solution is found
    - e.g. $\pi_1 = (\pi_{12},\pi_{21}),\ldots$

$\longrightarrow$ A policy cannot be selected twice in the running of the algorithm

$\longrightarrow$ The number of iterations is bounded by the number of policies

Constraints Generation for Bit-Level Tuning

Pendulum Program (PI)

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

$$C_2 = \left\{ \begin{array}{l} \mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{17}), \\ \mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{22}), \\ \mathsf{nsb}(\ell_{23}) \geq -1 - 0 + \mathsf{nsb}(\ell_{22}) - \mathsf{nsb}(\ell_{17}) + \mathsf{nsb_e}(\ell_{22}) + \xi(\ell_{23}, \ell_{17}, \ell_{22}), \\ \mathsf{nsb_e}(\ell_{23}) \geq 0 - (-1) + \mathsf{nsb}(\ell_{17}) - \mathsf{nsb}(\ell_{22}) + \mathsf{nsb_e}(\ell_{17}) + \xi(\ell_{23}, \ell_{17}, \ell_{22}), \\ \mathsf{nsb_e}(\ell_{23}) \geq \mathsf{nsb_e}(\ell_{24}), \\ \mathsf{nsb_e}(\ell_{22}) \geq \mathsf{nsb}(\ell_{19}) + \mathsf{nsb_e}(\ell_{19}) + \mathsf{nsb_e}(\ell_{21}) - 2, \\ \mathsf{nsb_e}(\ell_{22}) \geq \mathsf{nsb}(\ell_{21}) + \mathsf{nsb_e}(\ell_{21}) + \mathsf{nsb_e}(\ell_{19}) - 2, \\ \xi(\boldsymbol{\ell_{23}})(\boldsymbol{\ell_{17}}, \boldsymbol{\ell_{22}}) = \min \left(\begin{array}{l} \max\left(0 - 6 + \mathsf{nsb}(\ell_{17}) - \mathsf{nsb}(\ell_{22}) - \mathsf{nsb_e}(\ell_{17}), 0\right), \\ \max\left(6 - 0 + \mathsf{nsb}(\ell_{22}) - \mathsf{nsb}(\ell_{17}) - \mathsf{nsb_e}(\ell_{22}), 0\right), 1 \end{array} \right) \end{array} \right\}$$

- $C = C_1 \cup C_2$: global set of constraints
- Activate optimized ξ instead of its over-approximation in pure ILP
- Optimal solution found in few seconds

Tuned Pendulum Program (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

- 861 bits (ILP) VS 843 bits with PI method

POP(ILP)[1, 2], POP(SMT)[3, 4, 5, 6] and Precimonious

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)	-	-	-	-

- **Adjusting comparison criteria**
  - Consider **only** the variables optimized by Precimonious to estimate the quality of the optimization.
  - Express all the error thresholds in base 10

[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.]

Takeaways/Future Directions

- New approach for precision tuning with two variants of methods
  - Pure ILP with an over-approximation of the carry functions
  - PI technique for more precise carry bit function
- Limitation is the size of the problem accepted by the solver
- Reduce number of variables by assigning the same precision to the whole piece of code
- Technique adaptable for DNN's
- Code synthesis for the fixed-point arithmetic

Main References

[1] Assalé Adjé, Dorra Ben Khalifa, and Matthieu Martel.
Fast and efficient bit-level precision tuning.
Submitted.

[2] Dorra Ben Khalifa and Matthieu Martel.
A study of the floating-point tuning behaviour on the n-body problem.
Submitted.

[3] Dorra Ben Khalifa and Matthieu Martel.
Precision tuning and internet of things.
In *International Conference on Internet of Things, Embedded Systems and Communications, IINTEC 2019*, pages 80–85. IEEE, 2019.

[4] Dorra Ben Khalifa and Matthieu Martel.
Precision tuning of an accelerometer-based pedometer algorithm for iot devices.
In *International Conference on Internet of Things and Intelligence System, IOTAIS 2020*, pages 113–119. IEEE, 2020.

[5] Dorra Ben Khalifa and Matthieu Martel.
An evaluation of pop performance for tuning numerical programs in floating-point arithmetic.
In *International Conference on Information and Computer Technologies, ICICT 2021*. IEEE, 2021.

[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, FTSCS 2019*, volume 1165 of *Communications in Computer and Information Science*, pages 77–94. Springer, 2019.

Thank you! Questions?