16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing
Timisoara, Romania, September 22nd 2014

Automated Synthesis of Target-Dependent Programs for Polynomial Evaluation in Fixed-Point Arithmetic

Christophe Mouilleron   Amine Najahi   Guillaume Revy

Univ. Perpignan Via Domitia, DALI project-team
Univ. Montpellier 2, LIRMM, UMR 5506
CNRS, LIRMM, UMR 5506

Summary

Context and objectives

- Automated synthesis of fixed-point programs
  - → work done within the french ANR DEFIS project (`http://defis.lip6.fr`)
  - → targeting critical systems
- Optimized code
  - → use of the advanced instructions available on the target embedded systems
- Certified accuracy bounds using analytic approaches
  - → contrarily to simulation based approaches

Achievements

1. A new arithmetic model for the synthesis of signed fixed-point programs
2. A modular approach to optimize the generated code based on instruction selection
   - → the generated code can be optimized for accuracy, for latency, or for both
   - → more than 10% of speedup and almost up to 1 bit of precision gained

Motivation

- In this talk, we will focus on polynomial evaluation
  - it frequently appears as a building block of some mathematical operator implementation $\rightsquigarrow$ floating-point support emulation
  - it can be used to convert calls to floating-point operators into fixed-point code $\rightsquigarrow$ fixed-point conversion
- Remark: There is a huge number of schemes to evaluate a given polynomial, even for small degree
  - degree-5 univariate polynomial $\rightsquigarrow$ 2334244 different schemes

There is a need for the automation of the design of polynomial evaluation codes $\rightsquigarrow$ CGPE.

Outline of the talk

1. The CGPE tool

2. Code optimization through instruction selection

3. Conclusion and perspectives

Outline of the talk

Overview of CGPE

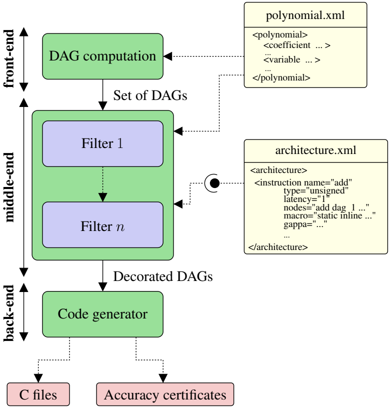
- Goal of CGPE: automate the design of fast and certified C codes for evaluating univariate or bivariate polynomials in fixed-point arithmetic
  - by using unsigned fixed-point arithmetic only
  - by using the target architecture features (as much as possible)

- Remarks on CGPE
  - fast $\rightsquigarrow$ that reduce the evaluation latency on a given target
  - certified $\rightsquigarrow$ for which we can bound the error entailed by the evaluation within the given target's arithmetic

Global architecture of CGPE

- Architecture of CGPE ≈ architecture of a compiler
  - it proceeds in three main steps

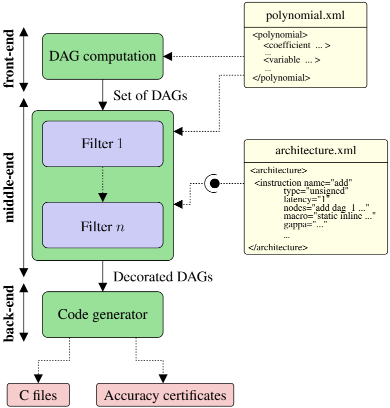
1. Computation step ⇝ front-end
   - computes schemes reducing the evaluation latency on unbounded parallelism ⇝ DAG
   - considers only the cost of ⊕ and ⊗



Global architecture of CGPE

- Architecture of CGPE $\approx$ architecture of a compiler
  - it proceeds in three main steps

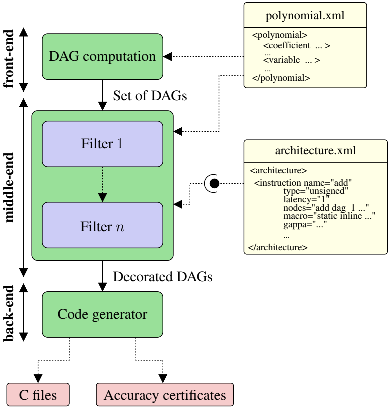
1. Computation step $\rightsquigarrow$ front-end
   - computes schemes reducing the evaluation latency on unbounded parallelism $\rightsquigarrow$ DAG
   - considers only the cost of $\oplus$ and $\otimes$
2. Filtering step $\rightsquigarrow$ middle-end
   - prunes the DAGs that do not satisfy different criteria:
     - latency $\rightsquigarrow$ scheduling filter,
     - accuracy $\rightsquigarrow$ numerical filter, ...



Global architecture of CGPE

- Architecture of CGPE ≈ architecture of a compiler
  - it proceeds in three main steps

1. Computation step ⇝ front-end
   - computes schemes reducing the evaluation latency on unbounded parallelism ⇝ DAG
   - considers only the cost of ⊕ and ⊗
2. Filtering step ⇝ middle-end
   - prunes the DAGs that do not satisfy different criteria:
     - latency ⇝ scheduling filter,
     - accuracy ⇝ numerical filter, ...
3. Generation step ⇝ back-end
   - generates C codes and Gappa accuracy certificates



Recent contributions to CGPE

- no support for signed fixed-point arithmetic
  - handling of variables of constants sign

  ⇝ problem: CGPE fails in evaluating polynomials around one of its roots

- hypotheses are made on the format of the inputs
  - no shift operators are allowed during the evaluation

  ⇝ problem: CGPE fails in evaluating polynomials with inputs having incorrect formats

- simple description of the target architecture
  - no handling of advanced operators

  ⇝ problem: CGPE fails in making the most out of any advanced instructions

Recent contributions to CGPE

- no support for signed fixed-point arithmetic *extension of the arithmetic model*
  - handling of variables of constants sign
  - ⇝ problem: CGPE fails in evaluating polynomials around one of its roots
- hypotheses are made on the format of the inputs *shift handling*
  - no shift operators are allowed during the evaluation
  - ⇝ problem: CGPE fails in evaluating polynomials with inputs having incorrect formats
- simple description of the target architecture *filter based on instruction selection*
  - no handling of advanced operators
  - ⇝ problem: CGPE fails in making the most out of any advanced instructions

Recent contributions to CGPE

- no support for signed fixed-point arithmetic *extension of the arithmetic model*
  - handling of variables of constants sign
  - ⇝ problem: CGPE fails in evaluating polynomials around one of its roots

- hypotheses are made on the format of the inputs *shift handling*
  - no shift operators are allowed during the evaluation
  - ⇝ problem: CGPE fails in evaluating polynomials with inputs having incorrect formats

- simple description of the target architecture *filter based on instruction selection*
  - no handling of advanced operators
  - ⇝ problem: CGPE fails in making the most out of any advanced instructions
  - ⇝ main motivation: it may absorb shifts appearing in the DAG, eventually in the critical path

Outline of the talk

Introduction to instruction selection

- It is a well known problem in compilation $\rightsquigarrow$ proven to be NP-complete on DAGs
- Usually solved using a tiling algorithm:
  - input:
    - a DAG representing an arithmetic expression,
    - a set of tiles, with a cost for each,
    - a function that associates a cost to a DAG.
  - output: a set of covering tiles that minimize the cost function.
- Examples of advanced instructions
  - `fma` on IEEE processors $\rightsquigarrow$ `a * b + c` with only one final rounding
  - `mulacc` on some DSP $\rightsquigarrow$ `a * b + c`
  - shift-and-add instruction on the ST231 $\rightsquigarrow$ `a << b + c` in 1 cycle, with $b \in \{1, \cdots, 4\}$

Motivation of using instruction selection inside CGPE

- Related work: Voronenko and Püschel from the Spiral group
  - Automatic Generation of Implementations for DSP Transforms on Fused Multiply-Add Architectures (2004)
  - Mechanical Derivation of Fused Multiply-Add Algorithms for Linear Transforms (2007)

Motivation of using instruction selection inside CGPE

- Related work: Voronenko and Püschel from the Spiral group
  - Automatic Generation of Implementations for DSP Transforms on Fused Multiply-Add Architectures (2004)
  - Mechanical Derivation of Fused Multiply-Add Algorithms for Linear Transforms (2007)

✓ they provide a short proof of optimality in the case of trees

✗ their method handles `fma` in DAGs but is not generic

Motivation of using instruction selection inside CGPE

- Related work: Voronenko and Püschel from the Spiral group
  - Automatic Generation of Implementations for DSP Transforms on Fused Multiply-Add Architectures (2004)
  - Mechanical Derivation of Fused Multiply-Add Algorithms for Linear Transforms (2007)

✓ they provide a short proof of optimality in the case of trees

✗ their method handles `fma` in DAGs but is not generic

- Our goal is twofold:
  1. to handle any advanced instruction $\rightsquigarrow$ described in an external XML file
  2. to integrate a numerical verification step in the process of instruction selection

XML architecture description file

```
<instruction
  name="add" type="unsigned" inputs="32 32" output="32"
  nodes="add dag 1 dag 2" latency="1"
  macro="static uint32_t __name__(uint32_t a, uint32_t b){
           return (a + b);
        }"
  gappa="_r_ fixed<_Fr_,dn>= _1_ + _2_;
        _Mr_ = _M1_ + _M2_;"
/>
```

- For each instruction, the XML architecture description file contains:
  - the name, the type (signed or unsigned), the latency (# cycles),
  - a description of the pattern matched by the instruction,
  - a C macro for emulating the instruction in software,
  - and a piece of Gappa script for computing the error entailed by the instruction evaluation in fixed-point arithmetic.

The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

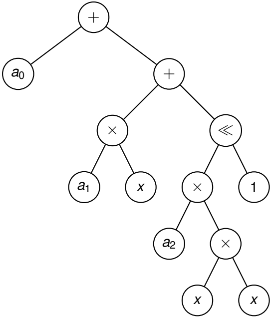
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?

addition / shift ⇝ 1 cycle

shift-and-add ⇝ 1 cycle

multiplication ⇝ 3 cycles



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

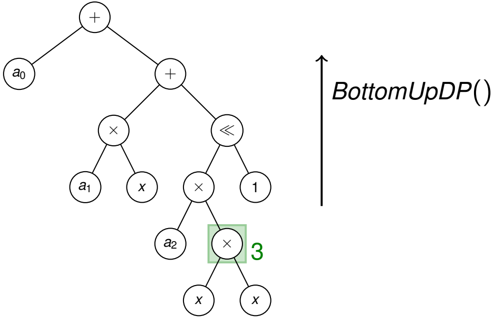
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1 \big) \Big)$?

addition / shift ⇝ 1 cycle

shift-and-add ⇝ 1 cycle

multiplication ⇝ 3 cycles



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

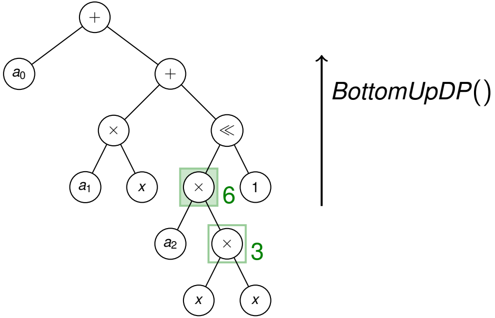
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?

addition / shift $\rightsquigarrow$ 1 cycle

shift-and-add $\rightsquigarrow$ 1 cycle

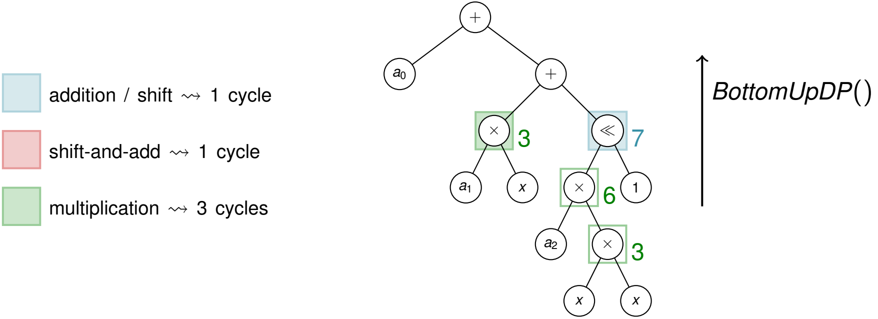
multiplication $\rightsquigarrow$ 3 cycles



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

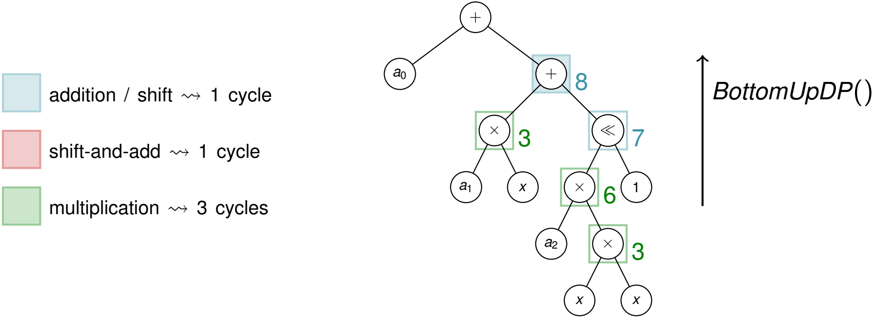
- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

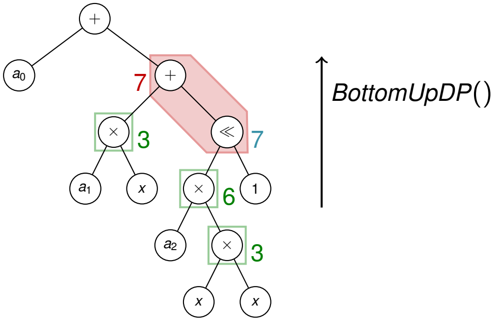
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?

addition / shift $\rightsquigarrow$ 1 cycle

shift-and-add $\rightsquigarrow$ 1 cycle

multiplication $\rightsquigarrow$ 3 cycles

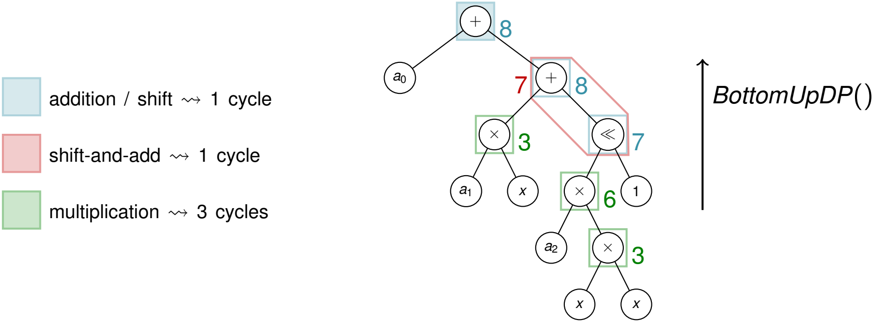


The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

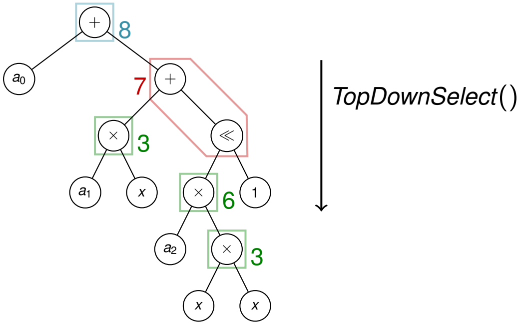
1: BottomUpDP() + TopDownSelect()
2: ImproveCSEDecision()
3: BottomUpDP() + TopDownSelect()

- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?

addition / shift ⇝ 1 cycle

shift-and-add ⇝ 1 cycle

multiplication ⇝ 3 cycles



The NOLTIS tiling algorithm

Near-Optimal Instruction Selection algorithm (Koes and Goldstein in CGO-2008)

1: BottomUpDP() + TopDownSelect()
2: ~~ImproveCSEDecision()~~
3: ~~BottomUpDP() + TopDownSelect()~~

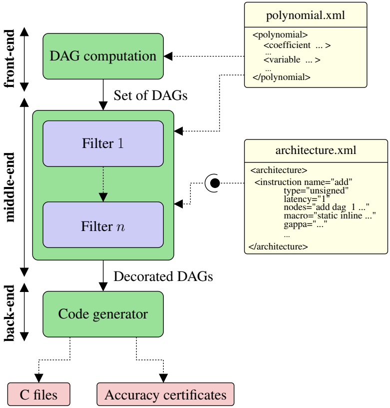
- Example: how to evaluate $a_0 + \Big((a_1 \cdot x) + \big((a_2 \cdot (x \cdot x)) \ll 1\big)\Big)$?

- In our case, only the first step of NOLTIS is valuable.

- NOLTIS algorithm mainly relies on the evaluation of a cost function. We have implemented three different cost functions:
  - ⇝ number of operator (regardless commun subexpressions)
  - ⇝ evaluation latency on unbounded parallelism
  - ⇝ evaluation accuracy, computed by using the piece of Gappa script for each instruction

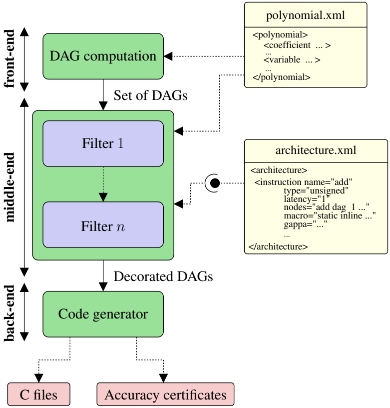
Remarks on instruction selection in CGPE

- A separation is achieved between the computation of the intermediate representation and the code generation process
  - we can generate codes according different criteria
  - we can generate target-dependent codes without writing new computation algorithms each time a new instruction is available
  - this general approach allows to tackle other problems (sum, dot-product, ...)



Remarks on instruction selection in CGPE

- A separation is achieved between the computation of the intermediate representation and the code generation process
  - we can generate codes according different criteria
  - we can generate target-dependent codes without writing new computation algorithms each time a new instruction is available
  - this general approach allows to tackle other problems (sum, dot-product, ...)



- We are not bounded to basic instructions
  - we can add many others advanced instructions or basic blocks
  - this general approach allows to give some feedback on the eventual need of some new instructions

Impact on the number of instructions

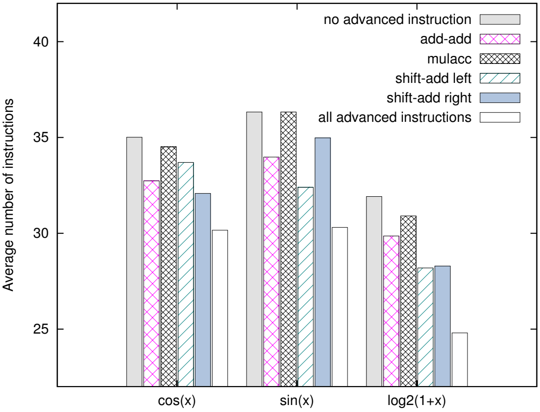


Figure: Average number of instructions in 50 synthesized codes, for the evaluation of polynomials of degree 5 up to 12 for various elementary functions.

- Remark 1: average reduction of 8.7 % up to 13.75 %
- Remark 2: interest of ST231 shift-and-add for $\sin(x)$ implementation
  $\rightsquigarrow$ reduction of 8.7 %
- Remark 3: interest of shift-and-add with right shift for $\cos(x)$ and $\log_2(1+x)$ implementation
  $\rightsquigarrow$ reduction of 12.8 % and 13.75 %, respectively

Impact on the accuracy of some functions

$f(x)$	I	d	Accuracy	
			not optimized	optimized
$\exp(x)-1$	$[-0.25, 0.25]$	7	-26.98	-27.34
$\exp(x)$	$[0, 1]$	7	-13.94	-14.90
$\sin(x)$	$[-0.5, 0.5]$	9	-18.95	-19.91
$\cos(x)$	$[-0.5, 0.25]$	5	-27.01	-27.26
$\tan(x)$	$[0.25, 0.5]$	9	-18.81	-19.64
$\log_2(1+x)/x$	$[2^{-23}, 1]$	7	-13.94	-14.89
$\sqrt{1+x}$	$[2^{-23}, 1]$	7	-13.94	-14.90

Table: Impact of the accuracy based selection step on the certified accuracy of the generated code for various functions.

- Remark 1: with a `mulacc` that computes `(a * b) + (c >> n)` with `n` $\in \{1, \cdots, 31\}$ with one final rounding
- Remark 2: a gain of precision is obtained in all the cases, almost up to 1 bit of accuracy for *exp*, *sin*, $\log_2$ and $\sqrt{1+x}$

Impact on the latency

- Polynomial: degree-7 polynomial approximating the function $\cos(x)$ over $[0, 2]$
- Architecture:
  - 1 cycle addition/subtraction and shift-and-add
  - 3-cycle multiplication and `mulacc`

	Without tiling	With tiling	Speed-up
Horner's rule	41	34	$\approx$ 17.07 %
Estrin's rule	16	14	$\approx$ 12.5 %
Best scheme	15	13	$\approx$ 13.33 %

Table: Latency in # cycles on unbounded parallelism, for various schemes, with and without tiling.

Outline of the talk

Conclusion and perspectives

- Target-dependent code generation for fast and certified polynomial evaluation
  - in signed and unsigned fixed point arithmetic
  - using filter based on instruction selection, so as to make the most out of advanced instructions
  - selection according to different criteria: operator count, latency on unbounded parallelism, accuracy

- Further extensions of CGPE
  - this work has already been extended to sums and dot-products
    `http://cgpe.gforge.inria.fr/`
  - it has been used in higher level tools to generate fixed-point code for linear algebra programs FPLA
    `http://perso.univ-perp.fr/mohamedamine.najahi/fpla/`
  - to handle other arithmetics like the floating-point arithmetic, where the `fma` instruction is more and more ubiquitous

16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing
Timisoara, Romania, September 22nd 2014

Automated Synthesis of Target-Dependent Programs for Polynomial Evaluation in Fixed-Point Arithmetic

Christophe Mouilleron   Amine Najahi   Guillaume Revy

Univ. Perpignan Via Domitia, DALI project-team
Univ. Montpellier 2, LIRMM, UMR 5506
CNRS, LIRMM, UMR 5506