Conference on Design and Architectures for Signal and Image Processing
Madrid, Spain, October 8-10, 2014

Toward the synthesis of fixed-point code for matrix inversion based on Cholesky decomposition

Matthieu Martel 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
  - → particular case of linear algebra basic blocks
  - → work done within the french ANR DEFIS project (`http://defis.lip6.fr`)
  - → targeting critical systems
- Tight code size
  - → targets embedded systems and FPGAs: constrained in terms of chip area
- Certified accuracy bounds using analytic approaches
  - → contrarily to simulation based approaches

Achievements

1. Formalization of two new fixed-point operators: square root and division
2. Approach for the synthesis of matrix inversion based on Cholesky decomposition
   - → code synthesis for 40×40 triangular matrix inversion in few seconds

A strategy to achieve matrix inversion

Let M be a symmetric positive definite matrix of fixed-point variables.
To generate certified code that inverts M, one needs to:

- Generate code to compute B a lower triangular s.t. $M = B \cdot B^T$
- Generate code to compute $N = B^{-1}$
- Generate code to compute $M^{-1} = N^T \cdot N$

The basic blocks we need to include in our tool-chain

- Fixed-point code synthesis for matrix multiplication *(PECCS '14)*
- Fixed-point code synthesis for triangular matrix inversion *(DASIP '14)*
- Fixed-point code synthesis for Cholesky decomposition *(DASIP '14)*

Outline of the talk

1. Fixed-point arithmetic model
2. Code synthesis for matrix inversion
3. Experimental results
4. Concluding remarks and future work

Outline of the talk

1. Fixed-point arithmetic model

2. Code synthesis for matrix inversion

3. Experimental results

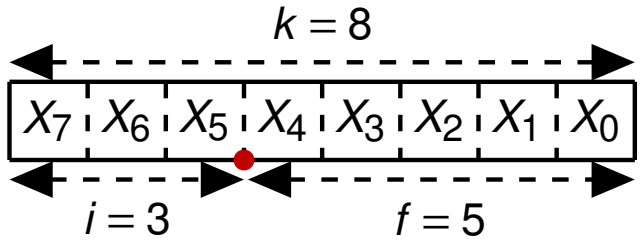
4. Concluding remarks and future work

Fixed-point arithmetic numbers

Definition and notation

A fixed-point number x is defined by two integers:

1. X the k-bit integer representation of x
2. f the implicit scaling factor of x



$\rightsquigarrow$ The value of x is given by $x = X \cdot 2^{-f}$

$\rightsquigarrow$ The variable x is in the $\mathbf{Q}_{i.f}$ format

Example

If x is in the format $\mathbf{Q}_{3.5}$ with $X = (1001\,1010)_2 = (154)_{10}$:

$$x = (100.11010)_2 = (4.8125)_{10}$$

Fixed-point arithmetic model

Arithmetic model to track errors in fixed-point computations

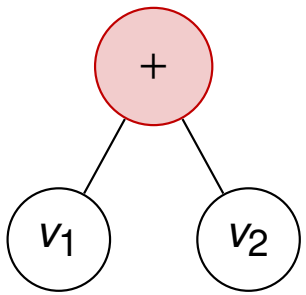
- For each variable v, we keep track of 2 intervals $\mathbf{Val}(v)$ and $\mathbf{Err}(v)$.
- For each basic operator, we have a rule that propagates these intervals.

Fixed-point arithmetic model

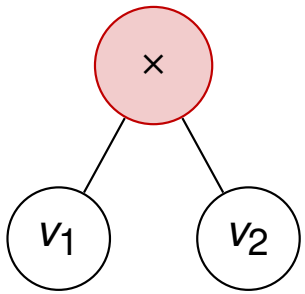
Arithmetic model to track errors in fixed-point computations

- For each variable v, we keep track of 2 intervals $\mathbf{Val}(v)$ and $\mathbf{Err}(v)$.
- For each basic operator, we have a rule that propagates these intervals.

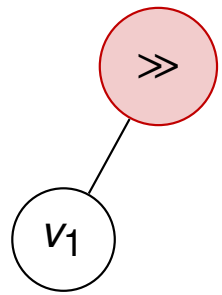
Propagation rules for $+$, $\times$ and $\gg$



$$\mathbf{Val}(v) = \mathbf{Val}(v_1) + \mathbf{Val}(v_2)$$
$$\mathbf{Err}(v) = \mathbf{Err}(v_1) + \mathbf{Err}(v_2)$$



$$\mathbf{Val}(v) = \mathbf{Val}(v_1) \times \mathbf{Val}(v_2) - \mathbf{Err}_{\times}$$
$$\mathbf{Err}(v) = \mathbf{Err}_{\times} + \mathbf{Err}(v_1) \times \mathbf{Err}(v_2) + \mathbf{Err}(v_2) \times \mathbf{Val}(v_1) + \mathbf{Val}(v_1) \times \mathbf{Err}(v_2)$$



$$\mathbf{Val}(v) = \mathbf{Val}(v_1) \gg \alpha - \mathbf{Err}_{\gg}$$
$$\mathbf{Err}(v) = \mathbf{Err}(v_1) + \mathbf{Err}_{\gg}$$

The CGPE software tool

- CGPE: a library to automate the synthesis of fast and certified fixed-point code
  - optimized for polynomial evaluation code synthesis
  - but also for summation and dot-product expressions
- CGPE uses interval arithmetic to compute certified accuracy bounds
- We use CGPE as a backend to synthesize code for linear algebra basic block
- CGPE is freely available for download under CeCILL v2 licence

`http://cgpe.gforge.inria.fr/`

Outline of the talk

1. Fixed-point arithmetic model

2. Code synthesis for matrix inversion

3. Experimental results

4. Concluding remarks and future work

Similar works

Previous works solving a similar problem

- **Frantz et al. (2007)**: *Design and Implementation of Numerical Linear Algebra Algorithms on Fixed Point DSPs*
- **Irturk et al. (2010)**: *GUSTO: An Automatic Generation and Optimization Tool for Matrix Inversion Architectures*

Recurring problems with existing works

- The tools are not available.
- Unclear arithmetic models.
- Sometimes, only toys examples are treated.
- Code generation is slow since it is based on simulation.
- Numerical accuracy is estimated a posteriori by comparing to floating-point.

Statement of the problems

Input

- A size-n matrix of interval fixed-point variables
  - triangular matrix inversion $\rightsquigarrow$ a lower triangular matrix B

    $$B \in \mathbb{F}\mathrm{ix}^{n \times n}$$
  - Cholesky decomposition $\rightsquigarrow$ a symmetric positive definite matrix M

    $$M \in \mathbb{F}\mathrm{ix}^{n \times n}$$

Output

- A fixed-point C code
  - to evaluate the inverse

    $$N' = (B')^{-1}, \quad \text{where} \quad B' \in B \quad \text{and} \quad B' \text{ is lower triangular}$$
  - to compute the decomposition

    $$B' = chol(M'), \quad \text{where} \quad M' \in M \quad \text{and} \quad M' \text{ is symmetric positive definite}$$
- An accuracy certificate verifiable by a formal proof checker

Missing basic blocks

Triangular matrix inversion

$$n_{i,j} = \begin{cases} \dfrac{1}{b_{i,i}} & \text{if } i = j \\ \dfrac{-c_{i,j}}{b_{i,i}} & \text{if } i \neq j \end{cases}$$

$$\text{where } c_{i,j} = \sum_{k=j}^{i-1} b_{i,k} \cdot n_{k,j}$$

Cholesky decomposition

$$b_{i,j} = \begin{cases} \sqrt{c_{i,i}} & \text{if } i = j \\ \dfrac{c_{i,j}}{b_{j,j}} & \text{if } i \neq j \end{cases}$$

$$\text{with } c_{i,j} = m_{i,j} - \sum_{k=0}^{j-1} b_{i,k} \cdot b_{j,k}$$

Missing basic blocks

Triangular matrix inversion

$$n_{i,j} = \begin{cases} \dfrac{1}{b_{i,i}} & \text{if } i = j \\ \dfrac{-c_{i,j}}{b_{i,i}} & \text{if } i \neq j \end{cases}$$

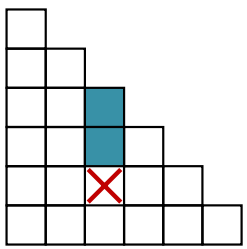
$$\text{where } c_{i,j} = \sum_{k=j}^{i-1} b_{i,k} \cdot n_{k,j}$$

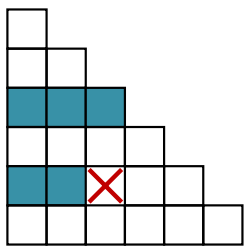
Cholesky decomposition

$$b_{i,j} = \begin{cases} \sqrt{c_{i,i}} & \text{if } i = j \\ \dfrac{c_{i,j}}{b_{j,j}} & \text{if } i \neq j \end{cases}$$

$$\text{with } c_{i,j} = m_{i,j} - \sum_{k=0}^{j-1} b_{i,k} \cdot b_{j,k}$$

- Two main difficulties of the synthesis process
  1. compared to matrix multiplication: the format of a given matrix coefficient depends directly upon the ones of previous computed coefficients





Missing basic blocks

Triangular matrix inversion

$$n_{i,j} = \begin{cases} \dfrac{1}{b_{i,i}} & \text{if } i = j \\ \dfrac{-c_{i,j}}{b_{i,i}} & \text{if } i \neq j \end{cases}$$

$$\text{where } c_{i,j} = \sum_{k=j}^{i-1} b_{i,k} \cdot n_{k,j}$$

Cholesky decomposition

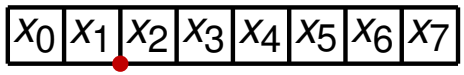
$$b_{i,j} = \begin{cases} \sqrt{c_{i,i}} & \text{if } i = j \\ \dfrac{c_{i,j}}{b_{j,j}} & \text{if } i \neq j \end{cases}$$

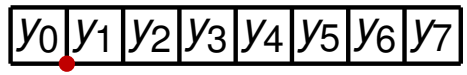
$$\text{with } c_{i,j} = m_{i,j} - \sum_{k=0}^{j-1} b_{i,k} \cdot b_{j,k}$$

- Two main difficulties of the synthesis process
  1. compared to matrix multiplication: the format of a given matrix coefficient depends directly upon the ones of previous computed coefficients
  2. some arithmetic problems may arise when dealing with division or square root

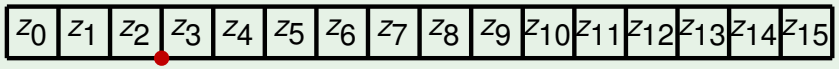
The dilemma of the division output format

- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:



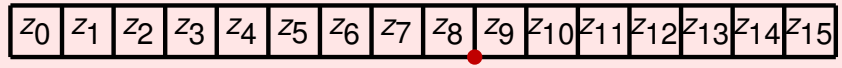


Multiplication



- Doubling the word-length
- $\mathbf{Err}_{\times} \in [0, 0]$

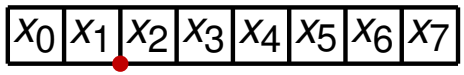
Division

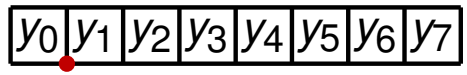


- Doubling the word-length.
- $\mathbf{Err}_{/} \in [-2^{-7}, 2^{-7}]$

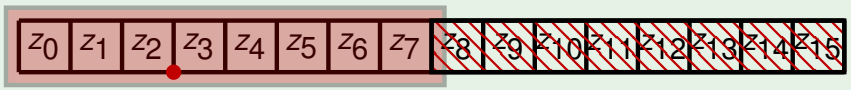
The dilemma of the division output format

- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:



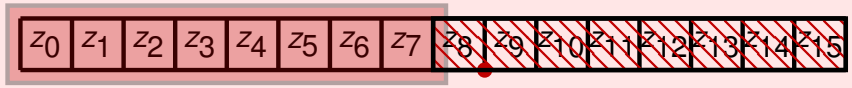


Multiplication



- Keeping the upper half of the result
- $\mathbf{Err}_{\times} \in [-2^{-5}, 2^{-5}]$

Division

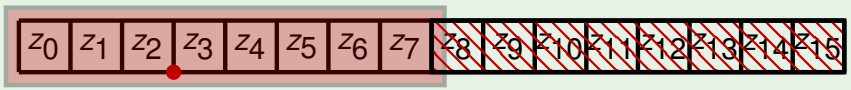


- Keeping the upper half of the result
- $\mathbf{Err}_{/} \in [-2, 2]$

The dilemma of the division output format

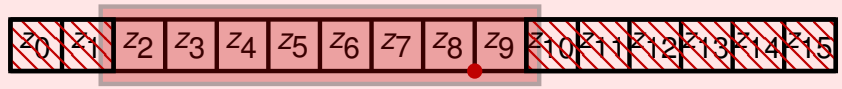
- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:

Multiplication



- Keeping the upper half of the result
- $\mathbf{Err}_{\times} \in [-2^{-5}, 2^{-5}]$

Division

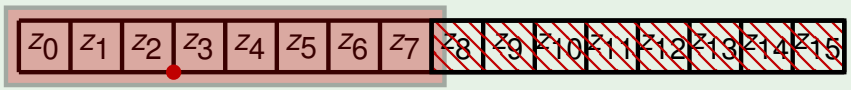


- Taking some risk of overflow!
- $\mathbf{Err}_{/} \in [-2^{-1}, 2^{-1}]$

The dilemma of the division output format

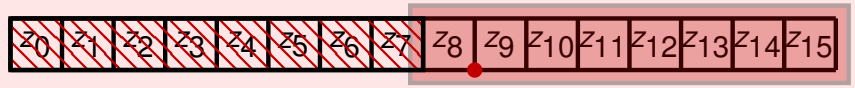
- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:

Multiplication



- Keeping the upper half of the result
- $\mathbf{Err}_{\times} \in [-2^{-5}, 2^{-5}]$

Division

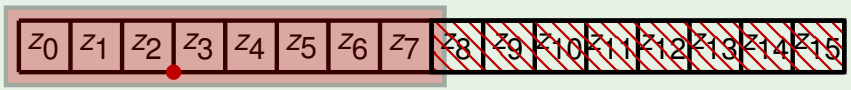


- Taking more risk of overflow!!
- $\mathbf{Err}_{/} \in [-2^{-6}, 2^{-6}]$

The dilemma of the division output format

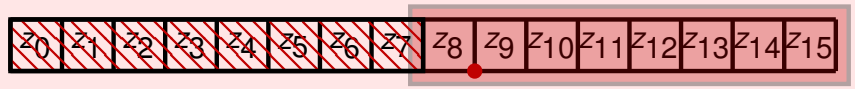
- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:

Multiplication



- Keeping the upper half of the result
- $\mathbf{Err}_{\times} \in [-2^{-5}, 2^{-5}]$

Division

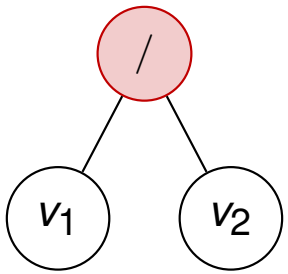


- Taking more risk of overflow!!
- $\mathbf{Err}_{/} \in [-2^{-6}, 2^{-6}]$

How to decide the output format of division?

- Keeping a large integer part
  - ✓ Prevents overflow
  - ✗ Leads to a loss of precision and loose error bounds
- Keeping a tight integer part
  - ✓ Leads to more precision and sharper error bounds
  - ✗ May cause overflow

Fixed-point division



$$\mathbf{Val}(v) = \frac{\mathbf{Val}(v_1)}{\mathbf{Val}(v_2)} - \mathbf{Err}_{/}$$

$$\mathbf{Err}(v) = \frac{\mathbf{Val}(v_2)\cdot\mathbf{Err}(v_1) - \mathbf{Val}(v_1)\cdot\mathbf{Err}(v_2)}{\mathbf{Val}(v_2)\cdot(\mathbf{Val}(v_2)+\mathbf{Err}(v_2))} + \mathbf{Err}_{/}$$

Our approach

1. Fix the output format $\rightsquigarrow (i_s, f_s)$
2. Compute

$$\frac{v_1}{v_2} = \frac{V_1\cdot 2^{-f_1}}{V_2\cdot 2^{-f_2}} = \frac{V_1\cdot 2^{f_s-f_1+f_2}}{V_2}\cdot 2^{-f_s}$$

3. Put the output result on a finite number of bits

$\rightsquigarrow$ $\mathbf{Err}_{/} = [-2^{-f_s}, 2^{-f_s}]$

Outline of the talk

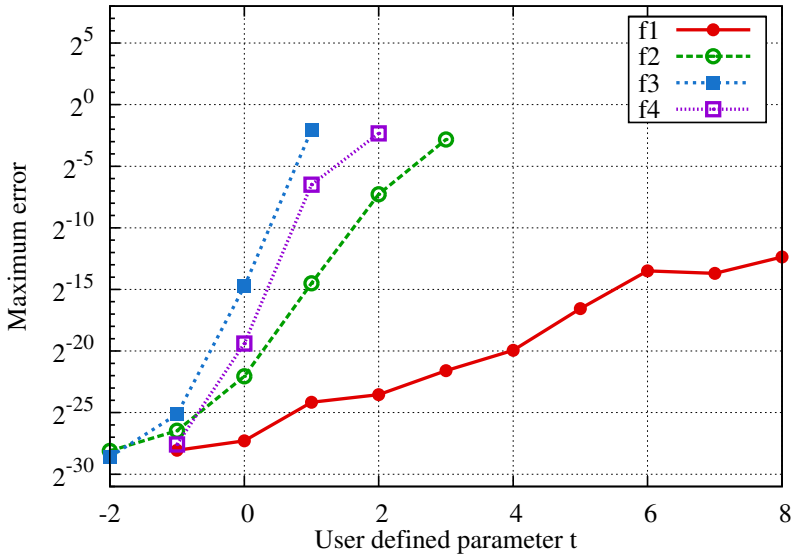
1. Fixed-point arithmetic model

2. Code synthesis for matrix inversion

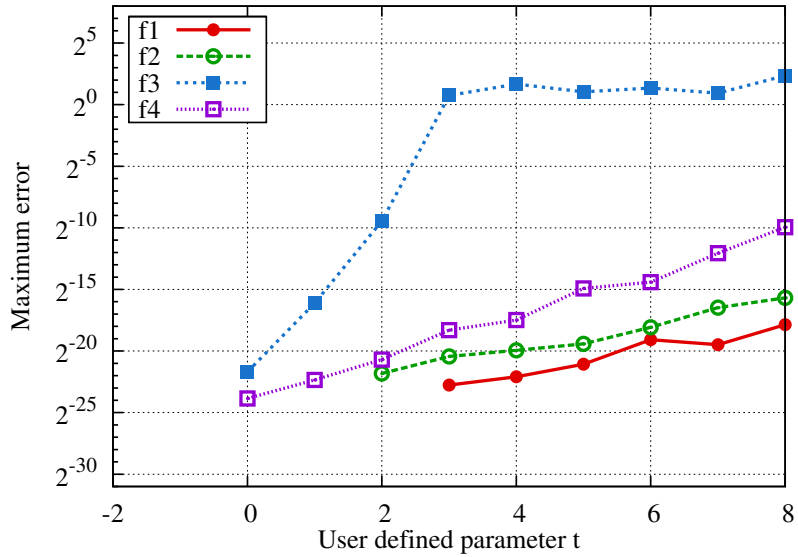
3. Experimental results

4. Concluding remarks and future work

Impact of the output format of division



(a) Cholesky decomposition 5×5.



(b) Triangular inversion 10×10.

Figure: Maximum error of Cholesky decomposition and triangular inversion with various functions used to determine the output formats of division.

- $f_1(i_1, i_2) = t$
- $f_2(i_1, i_2) = \min(i_1, i_2) + t$
- $f_3(i_1, i_2) = \max(i_1, i_2) + t$
- $f_4(i_1, i_2) = \lfloor (i_1 + i_2)/2 \rfloor + t$

where $t \in \mathbb{Z}$ is a user defined parameter, and i_1 and i_2 are the formats of the operands

How fast is generating triangular matrix inversion codes?

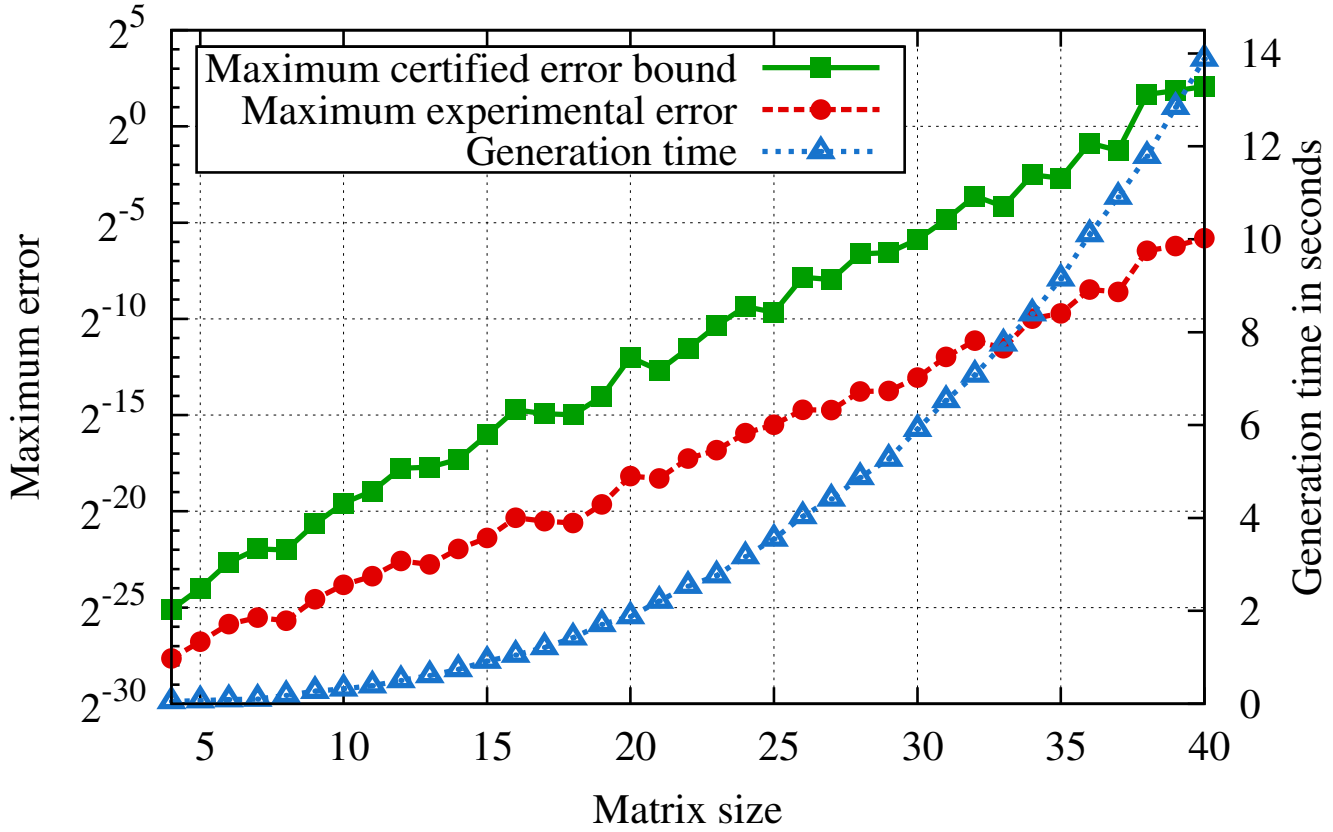
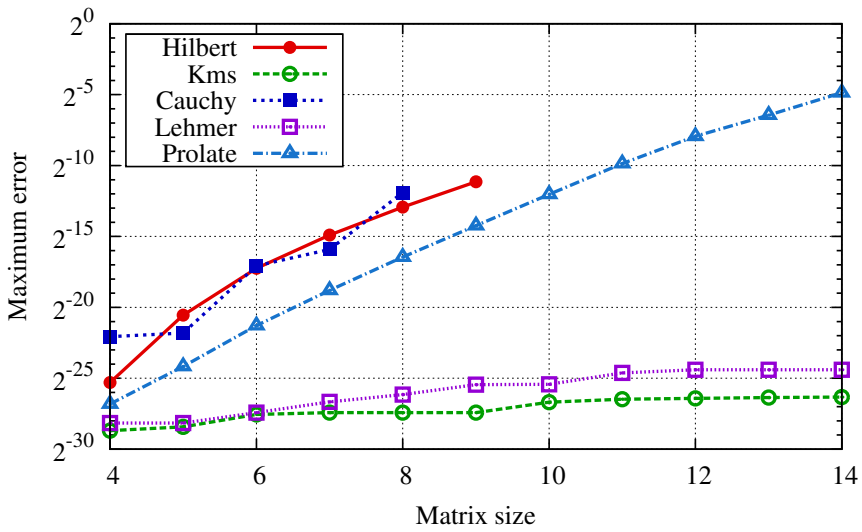
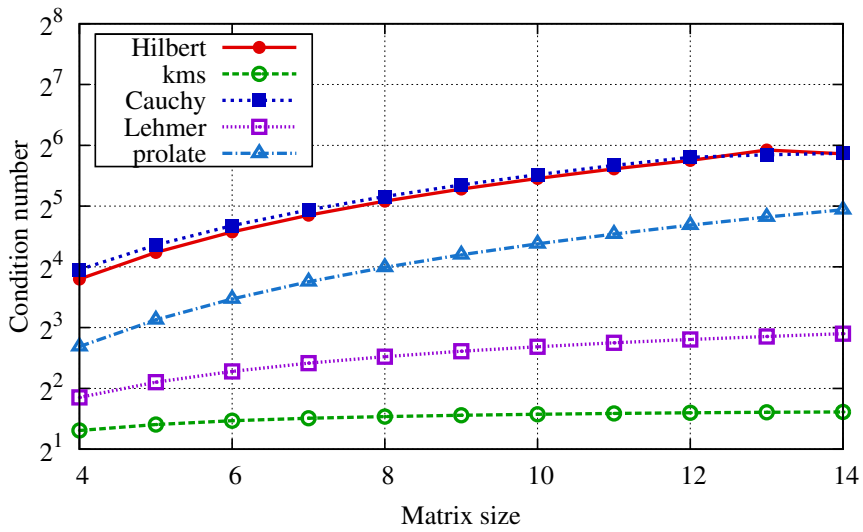


Figure: Comparison of the error bounds and experimental errors together with generation time, for the inversion of triangular matrices of size 4 to 40.

Decomposing some well known matrices



(a) Maximum experimental errors.



(b) Condition numbers.

Figure: Maximum errors measured when computing the Cholesky decomposition of various kinds of matrices for sizes varying from 4 to 14.

Outline of the talk

1. Fixed-point arithmetic model

2. Code synthesis for matrix inversion

3. Experimental results

4. Concluding remarks and future work

Conclusion remarks and future work

Work done so far

- Formalization and implementation of fixed-point square root and division
- Approach for the synthesis of triangular matrix inversion and Cholesky decomposition
  - matrices of size up to 40 in few seconds
- These algorithms are implemented in the FPLA tool

  `http://perso.univ-perp.fr/mohamedamine.najahi/fpla/`

Future work is twofold

- Further works on the arithmetic model:
  - understand better the role of the output format of division
  - derive sharper error bounds for square root
- Further works on the flow for matrix inversion:
  - integrate all the blocks to automate code generation for matrix inversion
  - handle alternative flows, based on LU or QR decomposition