Journée *Arithmétique pour le traitement de signal et de l'image* du GDR ISIS
Paris, July 3rd, 2014

Code synthesis for linear algebra basic blocks in fixed-point arithmetic

The cases of matrix multiplication and inversion

Amine Najahi
Advisors: Matthieu Martel and Guillaume Revy

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

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. Novel trade-off algorithm for the synthesis of matrix multiplication
   - → up to 50% code size reduction while satisfying the accuracy criterion
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.
- Fixed-point code synthesis for triangular matrix inversion.
- Fixed-point code synthesis for Cholesky decomposition.

Outline of the talk

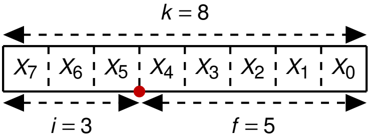
1. Our fixed-point arithmetic model
2. A novel tradeoff algorithm for code synthesis for matrix multiplication
3. Toward code synthesis for matrix inversion
4. Concluding remarks and future work

Outline of the talk

1. Our fixed-point arithmetic model
2. A novel tradeoff algorithm for code synthesis for matrix multiplication
3. Toward code synthesis for matrix inversion
4. Concluding remarks and future work

Fixed-point arithmetic numbers

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

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

Notation

A fixed-point number with i bits of integer part and f bits of fraction part is in the $\mathbf{Q}_{i.f}$ format.

Example:

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

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

- A fixed-point variable v in $\mathbf{Q}_{3.5}$ holds values in the discrete interval $[0, 7.96875]$

Fixed-point arithmetic model

Arithmetic model to track errors in fixed-point computations

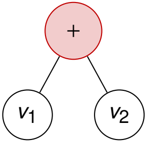
- For each variable v, we keep track of 3 intervals $\mathbf{Math}(v)$, $\mathbf{Val}(v)$ and $\mathbf{Err}(v)$.
  - They are related by the formula $\mathbf{Err}(v) = \mathbf{Math}(v) - \mathbf{Val}(v)$.
- For each basic operator, we have a rule that propagates these intervals.

Fixed-point arithmetic model

Arithmetic model to track errors in fixed-point computations

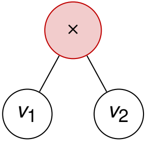
- For each variable v, we keep track of 3 intervals $\mathbf{Math}(v)$, $\mathbf{Val}(v)$ and $\mathbf{Err}(v)$.
  - They are related by the formula $\mathbf{Err}(v) = \mathbf{Math}(v) - \mathbf{Val}(v)$.
- For each basic operator, we have a rule that propagates these intervals.

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



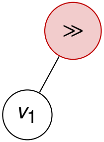
$\mathbf{Val}(v) = \mathbf{Val}(v_1) + \mathbf{Val}(v_2)$

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



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

$$\begin{aligned}\mathbf{Err}(v) = \mathbf{Err}_{\times} &+ \mathbf{Err}(v_1) \times \mathbf{Err}(v_2)\\ &+ \mathbf{Err}(v_2) \times \mathbf{Val}(v_1)\\ &+ \mathbf{Val}(v_1) \times \mathbf{Err}(v_2)\end{aligned}$$



$\mathbf{Val}(v) = \mathbf{Val}(v_1) \gg \alpha$

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

The CGPE software tool

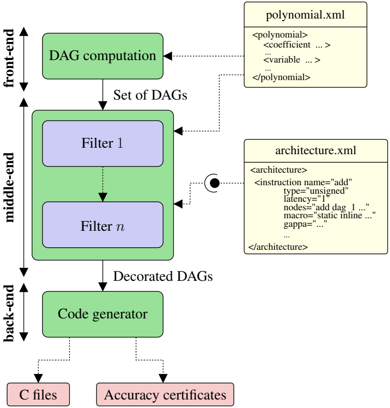
- CGPE: a library to automate the synthesis of fast and certified fixed-point code
  - optimized for polynomial evaluation code synthesis
  - but also for summation and dot-product expressions
- We use CGPE as a backend to synthesize code for linear algebra basic block
- CGPE is freely available for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

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

Focus on the CGPE software tool

- Architecture of CGPE ≈ architecture of a compiler

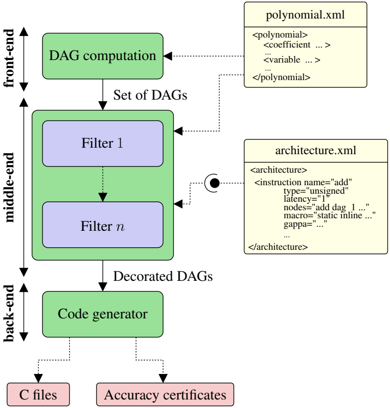
  1. Computation step ⇝ front-end
     - computes schemes reducing the evaluation latency on unbounded parallelism ⇝ DAG
     - considers only the cost of ⊕ and ⊗



Focus on the CGPE software tool

- Architecture of CGPE ≈ architecture of a compiler

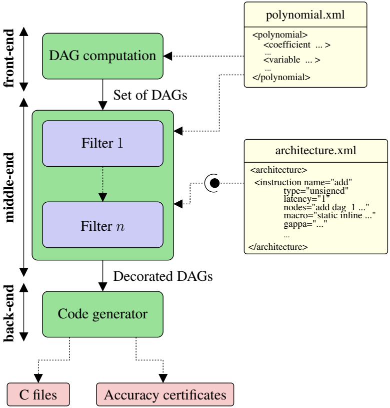
  1. Computation step ⇝ front-end
     - computes schemes reducing the evaluation latency on unbounded parallelism ⇝ DAG
     - considers only the cost of ⊕ and ⊗
  2. Filtering step ⇝ middle-end
     - prunes the DAGs that do not satisfy different criteria:
       - latency ⇝ scheduling filter,
     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



Focus on the CGPE software tool

- Architecture of CGPE $\approx$ architecture of a compiler

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, ...
3. Generation step $\rightsquigarrow$ back-end
   - generates C codes and Gappa accuracy certificates



Outline of the talk

Similar works

Previous works on linear algebra primitives in fixed-point

- **Lee et al. (2006)**: *Accuracy-Guaranteed Bit-Width Optimization.*
- **Frantz et al. (2007)**: *Design and Implementation of Numerical Linear Algebra Algorithms on Fixed Point DSPs.*

Recurring problems with existing works

- The tools are not available.
- Only toys examples are treated.
- Code generation is slow and is based on simulation.
- Numerical accuracy is estimated a posteriori by comparing to floating-point.

Statement of the problem

Inputs

- Two matrices A and B of interval fixed-point variables

$$A \in \mathbb{F}\mathrm{ix}^{m\times n} \quad \text{and} \quad B \in \mathbb{F}\mathrm{ix}^{n\times p}$$

- A bound $\mathscr{C}_1$ on the roundoff error
- A bound $\mathscr{C}_2$ on the code size

Statement of the problem

Inputs

- Two matrices A and B of interval fixed-point variables

$$A \in \mathbb{F}\mathrm{ix}^{m \times n} \quad \text{and} \quad B \in \mathbb{F}\mathrm{ix}^{n \times p}$$

- A bound $\mathscr{C}_1$ on the roundoff error
- A bound $\mathscr{C}_2$ on the code size

Output

- Fixed-point code (C, VHDL, ...) that evaluates the product

$$C' = A' \cdot B', \quad \text{where} \quad A' \in A \quad \text{and} \quad B' \in B$$

  that satisfy both $\mathscr{C}_1$ and $\mathscr{C}_2$
- Accuracy certificate (verifiable by a formal proof checker)

How to implement matrix multiplication?

Using floating-point numbers (C like syntax)

```
int main()
{
  int i,j,k;
  float A[N][N]={...}, B[N][N]={...}, C[N][N]={0,...,0};
  for (i = 0; i < N ; i++)
    for (j = 0; j < N ; j++)
      for (k = 0; k < N ; k++)
        C[i][j]+=A[i][k]*B[k][j];     /* This inner loop computes the dot-product of row i and column j */
}
```

What makes the problem harder in fixed-point?

- Intermediate computations depend on the input variables range and computation scheme
- Contrarily to the floating-point arithmetic, the programmer is in charge of:
  - overflow prevention, alignments, optimization of integer part lengths
    - $\rightsquigarrow$ requires the estimation of the dynamic range of intermediate variables

Straightforward algorithms

Accurate algorithm

- Main idea: a dot product code for each coefficient of the resulting matrix

Accurate algorithm

Inputs:

Two matrices $A \in \mathbb{F}ix^{m \times n}$ and $B \in \mathbb{F}ix^{n \times p}$

Outputs:

C code to compute the product $A \cdot B$

$m \cdot p$ accuracy certificates

Steps:

for $1 < i \leq m$ **do**
    for $1 < j \leq p$ **do**
        DPSynthesis$(A_{i,:}, B_{:,j})$
    end for
end for
Check $\mathscr{C}_1$ *and* $\mathscr{C}_2$

Straightforward algorithms

Accurate algorithm

- Main idea: a dot product code for each coefficient of the resulting matrix

Accurate algorithm

Inputs:

Two matrices $A \in \mathbb{F}ix^{m \times n}$ and $B \in \mathbb{F}ix^{n \times p}$

Outputs:

C code to compute the product $A \cdot B$

$m \cdot p$ accuracy certificates

Steps:

1: **for** $1 < i \leq m$ **do**
2: **for** $1 < j \leq p$ **do**
3: *DPSynthesis*$(A_{i,:}, B_{:,j})$
4: **end for**
5: **end for**
6: *Check* $\mathscr{C}_1$ *and* $\mathscr{C}_2$

Compact algorithm

- Main idea: a unique dot product code for all the coefficient of the resulting matrix

Compact algorithm

Inputs:

Two matrices $A \in \mathbb{F}ix^{m \times n}$ and $B \in \mathbb{F}ix^{n \times p}$

Outputs:

C code to compute the product $A \cdot B$

1 accuracy certificate

Steps:

1: $\mathscr{U} = A_{1,:} \cup A_{2,:} \cup \cdots \cup A_{m,:}$, with $\mathscr{U} \in \mathbb{F}ix^{1 \times n}$
2: $\mathscr{V} = B_{:,1} \cup B_{:,2} \cup \cdots \cup B_{:,p}$, with $\mathscr{V} \in \mathbb{F}ix^{n \times 1}$
3: *DPSynthesis*$(\mathscr{U}, \mathscr{V})$
4: *Check* $\mathscr{C}_1$ *and* $\mathscr{C}_2$

Illustration through a toy example

Consider the product of the following two fixed-point matrices:

$$A = \begin{pmatrix} [-1000,1000] & [-3000,3000] \\ [-1,1] & [-1,1] \end{pmatrix} \text{ and } B = \begin{pmatrix} [-2000,2000] & [-2,2] \\ [-4000,4000] & [-10,10] \end{pmatrix}$$

Coefficient	$A_{1,1}$	$A_{1,2}$	$A_{2,1}$	$A_{2,2}$	$B_{1,1}$	$B_{1,2}$	$B_{2,1}$	$B_{2,2}$
Fixed-point format	$\mathbf{Q}_{11.21}$	$\mathbf{Q}_{12.20}$	$\mathbf{Q}_{2.30}$	$\mathbf{Q}_{2.30}$	$\mathbf{Q}_{11.21}$	$\mathbf{Q}_{3.29}$	$\mathbf{Q}_{2.30}$	$\mathbf{Q}_{5.27}$

Illustration through a toy example

Consider the product of the following two fixed-point matrices:

$$A = \begin{pmatrix} [-1000, 1000] & [-3000, 3000] \\ [-1, 1] & [-1, 1] \end{pmatrix} \text{ and } B = \begin{pmatrix} [-2000, 2000] & [-2, 2] \\ [-4000, 4000] & [-10, 10] \end{pmatrix}$$

Coefficient	$A_{1,1}$	$A_{1,2}$	$A_{2,1}$	$A_{2,2}$	$B_{1,1}$	$B_{1,2}$	$B_{2,1}$	$B_{2,2}$
Fixed-point format	$\mathbf{Q}_{11.21}$	$\mathbf{Q}_{12.20}$	$\mathbf{Q}_{2.30}$	$\mathbf{Q}_{2.30}$	$\mathbf{Q}_{11.21}$	$\mathbf{Q}_{3.29}$	$\mathbf{Q}_{2.30}$	$\mathbf{Q}_{5.27}$

Accurate algorithm

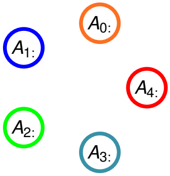
Dot-product	$A_{1,:} \cdot B_{:,1}$	$A_{1,:} \cdot B_{:,2}$	$A_{2,:} \cdot B_{:,1}$	$A_{2,:} \cdot B_{:,2}$
Evaluated using	$\text{DPCode}_{1,1}$	$\text{DPCode}_{1,2}$	$\text{DPCode}_{2,1}$	$\text{DPCode}_{2,2}$
Output format	$Q_{26,6}$	$Q_{18,14}$	$Q_{15,17}$	$Q_{7,25}$
Certified error	$\approx 2^{-5}$	$\approx 2^{-14}$	$\approx 2^{-16}$	$\approx 2^{-24}$
Maximum error	$\approx 2^{-5}$			
Average error	$\approx 2^{-7}$			

Compact algorithm

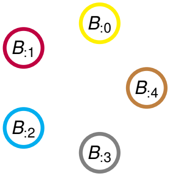
Dot-product	$A_{1,:} \cdot B_{:,1}$	$A_{1,:} \cdot B_{:,2}$	$A_{2,:} \cdot B_{:,1}$	$A_{2,:} \cdot B_{:,2}$
Evaluated using	$\text{DPCode}_{\mathcal{U},\mathcal{V}}$			
Output format	$Q_{26,6}$			
Certified error	$\approx 2^{-5}$			
Maximum error	$\approx 2^{-5}$			
Average error	$\approx 2^{-5}$			

Tradeoff algorithms

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \qquad B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}$$



Accurate algorithm:
(25 dot-product codes)

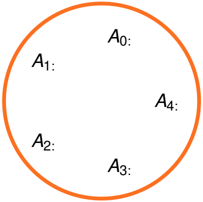


$$C = A \cdot B = \begin{pmatrix} \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,0}) & \mathrm{DPCode}_{0,1}(A_{0,:}, B_{:,1}) & \mathrm{DPCode}_{0,2}(A_{0,:}, B_{:,2}) & \mathrm{DPCode}_{0,3}(A_{0,:}, B_{:,3}) & \mathrm{DPCode}_{0,4}(A_{0,:}, B_{:,4}) \\ \mathrm{DPCode}_{1,0}(A_{1,:}, B_{:,0}) & \mathrm{DPCode}_{1,1}(A_{1,:}, B_{:,1}) & \mathrm{DPCode}_{1,2}(A_{1,:}, B_{:,2}) & \mathrm{DPCode}_{1,3}(A_{1,:}, B_{:,3}) & \mathrm{DPCode}_{1,4}(A_{1,:}, B_{:,4}) \\ \mathrm{DPCode}_{2,0}(A_{2,:}, B_{:,0}) & \mathrm{DPCode}_{2,1}(A_{2,:}, B_{:,1}) & \mathrm{DPCode}_{2,2}(A_{2,:}, B_{:,2}) & \mathrm{DPCode}_{2,3}(A_{2,:}, B_{:,3}) & \mathrm{DPCode}_{2,4}(A_{2,:}, B_{:,4}) \\ \mathrm{DPCode}_{3,0}(A_{3,:}, B_{:,0}) & \mathrm{DPCode}_{3,1}(A_{3,:}, B_{:,1}) & \mathrm{DPCode}_{3,2}(A_{3,:}, B_{:,2}) & \mathrm{DPCode}_{3,3}(A_{3,:}, B_{:,3}) & \mathrm{DPCode}_{3,4}(A_{3,:}, B_{:,4}) \\ \mathrm{DPCode}_{4,0}(A_{4,:}, B_{:,0}) & \mathrm{DPCode}_{4,1}(A_{4,:}, B_{:,1}) & \mathrm{DPCode}_{4,2}(A_{4,:}, B_{:,2}) & \mathrm{DPCode}_{4,3}(A_{4,:}, B_{:,3}) & \mathrm{DPCode}_{4,4}(A_{4,:}, B_{:,4}) \end{pmatrix}$$

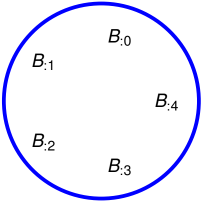
Tradeoff algorithms

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}$$



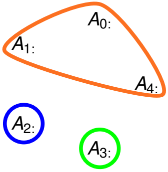
Compact algorithm:
(1 dot-product code)



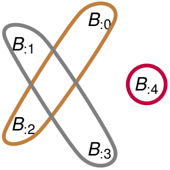
$$C = A \cdot B = \begin{pmatrix} \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,0}) & \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,2}) & \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,3}) & \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,4}) \\ \mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,0}) & \mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,2}) & \mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,3}) & \mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,4}) \\ \mathrm{DPCode}_{0,0}(A_{2,:}, B_{:,0}) & \mathrm{DPCode}_{0,0}(A_{2,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{2,:}, B_{:,2}) & \mathrm{DPCode}_{0,0}(A_{2,:}, B_{:,3}) & \mathrm{DPCode}_{0,0}(A_{2,:}, B_{:,4}) \\ \mathrm{DPCode}_{0,0}(A_{3,:}, B_{:,0}) & \mathrm{DPCode}_{0,0}(A_{3,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{3,:}, B_{:,2}) & \mathrm{DPCode}_{0,0}(A_{3,:}, B_{:,3}) & \mathrm{DPCode}_{0,0}(A_{3,:}, B_{:,4}) \\ \mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,0}) & \mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,2}) & \mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,3}) & \mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,4}) \end{pmatrix}$$

Tradeoff algorithms

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \qquad B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}$$



Tradeoff algorithm:
(9 dot-product codes)



$$C = A \cdot B = \begin{pmatrix}
\mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,0}) & \mathrm{DPCode}_{0,1}(A_{0,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{0,:}, B_{:,2}) & \mathrm{DPCode}_{0,1}(A_{0,:}, B_{:,3}) & \mathrm{DPCode}_{0,4}(A_{0,:}, B_{:,4}) \\
\mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,0}) & \mathrm{DPCode}_{0,1}(A_{1,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{1,:}, B_{:,2}) & \mathrm{DPCode}_{0,1}(A_{1,:}, B_{:,3}) & \mathrm{DPCode}_{0,4}(A_{1,:}, B_{:,4}) \\
\mathrm{DPCode}_{2,0}(A_{2,:}, B_{:,0}) & \mathrm{DPCode}_{2,1}(A_{2,:}, B_{:,1}) & \mathrm{DPCode}_{2,0}(A_{2,:}, B_{:,2}) & \mathrm{DPCode}_{2,1}(A_{2,:}, B_{:,3}) & \mathrm{DPCode}_{2,4}(A_{2,:}, B_{:,4}) \\
\mathrm{DPCode}_{3,0}(A_{3,:}, B_{:,0}) & \mathrm{DPCode}_{3,1}(A_{3,:}, B_{:,1}) & \mathrm{DPCode}_{3,0}(A_{3,:}, B_{:,2}) & \mathrm{DPCode}_{3,1}(A_{3,:}, B_{:,3}) & \mathrm{DPCode}_{3,4}(A_{3,:}, B_{:,4}) \\
\mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,0}) & \mathrm{DPCode}_{0,1}(A_{4,:}, B_{:,1}) & \mathrm{DPCode}_{0,0}(A_{4,:}, B_{:,2}) & \mathrm{DPCode}_{0,1}(A_{4,:}, B_{:,3}) & \mathrm{DPCode}_{0,4}(A_{4,:}, B_{:,4})
\end{pmatrix}$$

Tradeoff algorithms

$$A = \begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} \\ a_{40} & a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \qquad B = \begin{pmatrix} b_{00} & b_{01} & b_{02} & b_{03} & b_{04} \\ b_{10} & b_{11} & b_{12} & b_{13} & b_{14} \\ b_{20} & b_{21} & b_{22} & b_{23} & b_{24} \\ b_{30} & b_{31} & b_{32} & b_{33} & b_{34} \\ b_{40} & b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix}$$

Number of possible tradeoff algorithms

- The number of ways to merge k vectors is given by the Bell number $\mathscr{B}(k)$

Number of vectors k	3	5	10	16	20	$\cdots$
Bell number $\mathscr{B}(k)$	5	52	$115975 \approx 2^{17}$	$10480142147 \approx 2^{33}$	$51724158235372 \approx 2^{46}$	$\cdots$

$\hookrightarrow$ The total numbers of algorithms is given by $\mathscr{B}(m) \cdot \mathscr{B}(p)$

(m,p)	$(5,5)$	$(6,6)$	$(10,10)$	$(16,16)$	$(25,25)$	$(64,64)$	$\cdots$
Number of algorithms	2704	41 209	$\approx 2^{34}$	$\approx 2^{66}$	$\approx 2^{124}$	$\approx 2^{433}$	$\cdots$

Distances

The Hausdorff distance d_H

$$d_H : \mathbb{F}ix \times \mathbb{F}ix \rightarrow \mathbb{R}^+$$
$$d_H(I_1, I_2) = max\left\{\left|\underline{I_1} - \underline{I_2}\right|, \left|\overline{I_1} - \overline{I_2}\right|\right\}$$

Fixed-point distance

$$d_F : \mathbb{F}ix \times \mathbb{F}ix \rightarrow \mathbb{N}$$
$$d_F(I_1, I_2) = \left|IntegerPart(I_1) - IntegerPart(I_2)\right|$$

Width criterion

$$d_W : \mathbb{F}ix \times \mathbb{F}ix \rightarrow \mathbb{R}^+$$
$$d_W(I_1, I_2) = \left(\overline{I_1 \cup I_2} - \underline{I_1 \cup I_2}\right)$$

Distances

The Hausdorff distance d_H

$$d_H : \mathbb{F}ix \times \mathbb{F}ix \to \mathbb{R}^+$$
$$d_H(I_1, I_2) = max\left\{\left|\underline{I_1} - \underline{I_2}\right|, \left|\overline{I_1} - \overline{I_2}\right|\right\}$$

Fixed-point distance

$$d_F : \mathbb{F}ix \times \mathbb{F}ix \to \mathbb{N}$$
$$d_F(I_1, I_2) = \left|IntegerPart(I_1) - IntegerPart(I_2)\right|$$

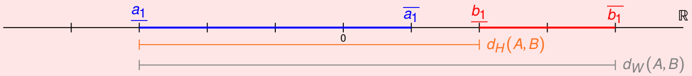
Width criterion

$$d_W : \mathbb{F}ix \times \mathbb{F}ix \to \mathbb{R}^+$$
$$d_W(I_1, I_2) = \left(\overline{I_1 \cup I_2} - \underline{I_1 \cup I_2}\right)$$

Example

Let $A = [-3, 1]$ and $B = [2, 4]$ with A in the fixed-point format $Q_{3,29}$ and B in $Q_{4,28}$, we have:

- $d_H(A, B) = 5$
- $d_F(A, B) = |3 - 4| = 1$
- $d_W(A, B) = 7$



Closest pair strategy

Input:

Two matrices $A \in \mathbb{F}\mathrm{ix}^{m\times p}$ and $B \in \mathbb{F}\mathrm{ix}^{p\times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

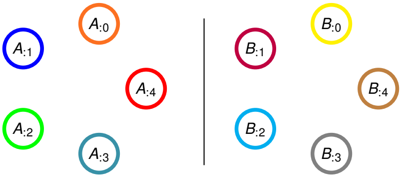
Output:

Code to compute $A \cdot B$ s.t. $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfied,
or no code otherwise

Algorithm:

$\mathscr{S}_A \leftarrow \{A_{0,:}, \ldots, A_{m-1,:}\}$
$\mathscr{S}_B \leftarrow \{B_{:,0}, \ldots, B_{:,n-1}\}$
while $\mathscr{C}_1$ is satisfied **do**
  $(u_A, v_A), d_A \leftarrow \mathit{findClosestPair}(\mathscr{S}_A, d)$
  $(u_B, v_B), d_B \leftarrow \mathit{findClosestPair}(\mathscr{S}_B, d)$
  if $d_A \leq d_B$ **then**
    $\mathit{remove}(u_A, v_A, \mathscr{S}_A)$
    $\mathit{insert}(u_A \cup v_A, \mathscr{S}_A)$
  else
    $\mathit{remove}(u_B, v_B, \mathscr{S}_B)$
    $\mathit{insert}(u_B \cup v_B, \mathscr{S}_B)$
  end if
  for $(A_i, B_j) \in \mathscr{S}_A \times \mathscr{S}_B$ **do**
    $\mathit{DPSynthesis}(A_i, B_j)$
  end for
end while
/* Revert the last merging step, and check the bound $\mathscr{C}_2$. */

Accurate algorithm



25 DPcodes

$\mathscr{C}_1$ is satisfied

Closest pair strategy

Input:

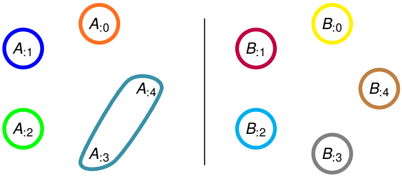
Two matrices $A \in \mathbb{F}\mathrm{ix}^{m\times p}$ and $B \in \mathbb{F}\mathrm{ix}^{p\times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

Code to compute $A \cdot B$ s.t. $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfied,
or no code otherwise

Algorithm:

$\mathscr{S}_A \leftarrow \{A_{0,:}, \ldots, A_{m-1,:}\}$
$\mathscr{S}_B \leftarrow \{B_{:,0}, \ldots, B_{:,n-1}\}$
while $\mathscr{C}_1$ is satisfied **do**
  $(u_A, v_A), d_A \leftarrow \mathit{findClosestPair}(\mathscr{S}_A, d)$
  $(u_B, v_B), d_B \leftarrow \mathit{findClosestPair}(\mathscr{S}_B, d)$
  if $d_A \le d_B$ **then**
    $\mathit{remove}(u_A, v_A, \mathscr{S}_A)$
    $\mathit{insert}(u_A \cup v_A, \mathscr{S}_A)$
  else
    $\mathit{remove}(u_B, v_B, \mathscr{S}_B)$
    $\mathit{insert}(u_B \cup v_B, \mathscr{S}_B)$
  end if
  for $(A_i, B_j) \in \mathscr{S}_A \times \mathscr{S}_B$ **do**
    $\mathit{DPSynthesis}(A_i, B_j)$
  end for
end while
/* Revert the last merging step, and check the bound $\mathscr{C}_2$. */



20 DPcodes

$\mathscr{C}_1$ is satisfied

Closest pair strategy

Input:

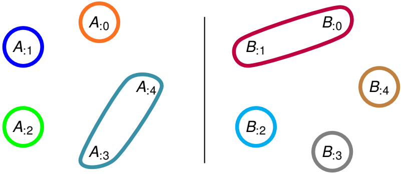
Two matrices $A \in \mathbb{F}\mathrm{ix}^{m \times p}$ and $B \in \mathbb{F}\mathrm{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

Code to compute $A \cdot B$ s.t. $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfied,
or no code otherwise

Algorithm:

$\mathscr{S}_A \leftarrow \{A_{0,:}, \ldots, A_{m-1,:}\}$
$\mathscr{S}_B \leftarrow \{B_{:,0}, \ldots, B_{:,n-1}\}$
while $\mathscr{C}_1$ is satisfied **do**
    $(u_A, v_A), d_A \leftarrow \mathit{findClosestPair}(\mathscr{S}_A, d)$
    $(u_B, v_B), d_B \leftarrow \mathit{findClosestPair}(\mathscr{S}_B, d)$
    if $d_A \leq d_B$ **then**
        $\mathit{remove}(u_A, v_A, \mathscr{S}_A)$
        $\mathit{insert}(u_A \cup v_A, \mathscr{S}_A)$
    else
        $\mathit{remove}(u_B, v_B, \mathscr{S}_B)$
        $\mathit{insert}(u_B \cup v_B, \mathscr{S}_B)$
    end if
    for $(A_i, B_j) \in \mathscr{S}_A \times \mathscr{S}_B$ **do**
        $\mathit{DPSynthesis}(A_i, B_j)$
    end for
end while
/* Revert the last merging step, and check the bound $\mathscr{C}_2$. */



16 `DPcodes`

$\mathscr{C}_1$ is satisfied

Closest pair strategy

Input:

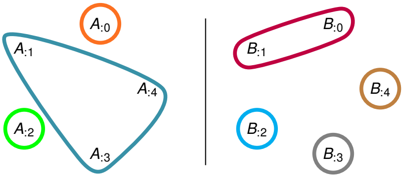
Two matrices $A \in \mathbb{F}\mathrm{ix}^{m \times p}$ and $B \in \mathbb{F}\mathrm{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

Code to compute $A \cdot B$ s.t. $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfied,
or no code otherwise

Algorithm:

$\mathscr{S}_A \leftarrow \{A_{0,:}, \ldots, A_{m-1,:}\}$
$\mathscr{S}_B \leftarrow \{B_{:,0}, \ldots, B_{:,n-1}\}$
while $\mathscr{C}_1$ is satisfied **do**
  $(u_A, v_A), d_A \leftarrow \mathit{findClosestPair}(\mathscr{S}_A, d)$
  $(u_B, v_B), d_B \leftarrow \mathit{findClosestPair}(\mathscr{S}_B, d)$
  if $d_A \le d_B$ **then**
    $\mathit{remove}(u_A, v_A, \mathscr{S}_A)$
    $\mathit{insert}(u_A \cup v_A, \mathscr{S}_A)$
  else
    $\mathit{remove}(u_B, v_B, \mathscr{S}_B)$
    $\mathit{insert}(u_B \cup v_B, \mathscr{S}_B)$
  end if
  for $(A_i, B_j) \in \mathscr{S}_A \times \mathscr{S}_B$ **do**
    $\mathit{DPSynthesis}(A_i, B_j)$
  end for
end while
/* Revert the last merging step, and check the bound $\mathscr{C}_2$. */



12 `DPcodes`

$\mathscr{C}_1$ is satisfied

Closest pair strategy

Input:

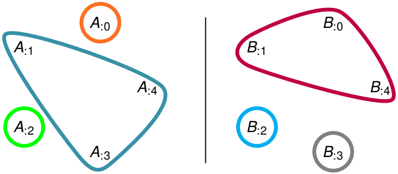
Two matrices $A \in \mathbb{F}\mathrm{ix}^{m \times p}$ and $B \in \mathbb{F}\mathrm{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

Code to compute $A \cdot B$ s.t. $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfied,
or no code otherwise

Algorithm:

$\mathscr{S}_A \leftarrow \{A_{0,:}, \ldots, A_{m-1,:}\}$
$\mathscr{S}_B \leftarrow \{B_{:,0}, \ldots, B_{:,n-1}\}$
while $\mathscr{C}_1$ is satisfied **do**
    $(u_A, v_A), d_A \leftarrow \mathit{findClosestPair}(\mathscr{S}_A, d)$
    $(u_B, v_B), d_B \leftarrow \mathit{findClosestPair}(\mathscr{S}_B, d)$
    if $d_A \leq d_B$ **then**
        $\mathit{remove}(u_A, v_A, \mathscr{S}_A)$
        $\mathit{insert}(u_A \cup v_A, \mathscr{S}_A)$
    else
        $\mathit{remove}(u_B, v_B, \mathscr{S}_B)$
        $\mathit{insert}(u_B \cup v_B, \mathscr{S}_B)$
    end if
    for $(A_i, B_j) \in \mathscr{S}_A \times \mathscr{S}_B$ **do**
        $\mathit{DPSynthesis}(A_i, B_j)$
    end for
end while
/* Revert the last merging step, and check the bound $\mathscr{C}_2$. */



9 DPcodes

$\mathscr{C}_1$ is no longer satisfied

Closest pair strategy

Input:

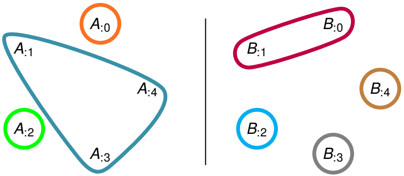
Two matrices $A \in \mathbb{F}\mathrm{ix}^{m \times p}$ and $B \in \mathbb{F}\mathrm{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

Code to compute $A \cdot B$ s.t. $\mathscr{C}_1$ and $\mathscr{C}_2$ are satisfied,
or no code otherwise

Algorithm:

$\mathscr{S}_A \leftarrow \{A_{0,:}, \ldots, A_{m-1,:}\}$
$\mathscr{S}_B \leftarrow \{B_{:,0}, \ldots, B_{:,n-1}\}$
while $\mathscr{C}_1$ is satisfied **do**
    $(u_A, v_A), d_A \leftarrow \mathit{findClosestPair}(\mathscr{S}_A, d)$
    $(u_B, v_B), d_B \leftarrow \mathit{findClosestPair}(\mathscr{S}_B, d)$
    if $d_A \leq d_B$ **then**
        $\mathit{remove}(u_A, v_A, \mathscr{S}_A)$
        $\mathit{insert}(u_A \cup v_A, \mathscr{S}_A)$
    else
        $\mathit{remove}(u_B, v_B, \mathscr{S}_B)$
        $\mathit{insert}(u_B \cup v_B, \mathscr{S}_B)$
    end if
    for $(A_i, B_j) \in \mathscr{S}_A \times \mathscr{S}_B$ **do**
        $\mathit{DPSynthesis}(A_i, B_j)$
    end for
end while
/* Revert the last merging step, and check the bound $\mathscr{C}_2$. */



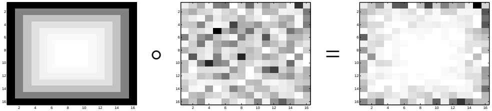
12 DPcodes

$\mathscr{C}_1$ is satisfied

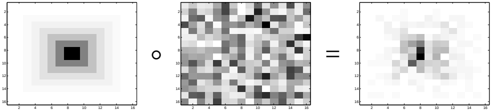
$\rightsquigarrow$ Revert the last merging step and check if $\mathscr{C}_2$ is satisfied

Benchmarks generation methodology

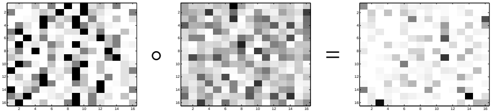
Edges



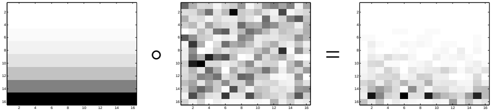
Center



Random

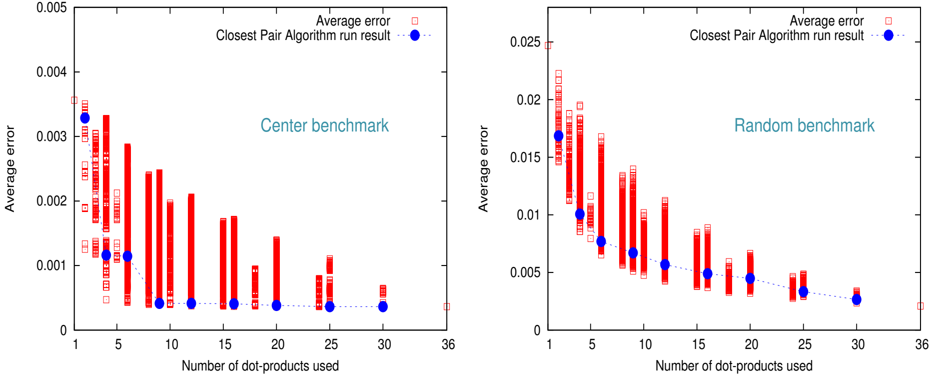


Rows

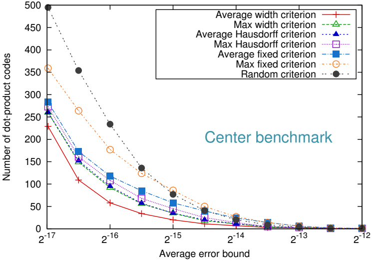


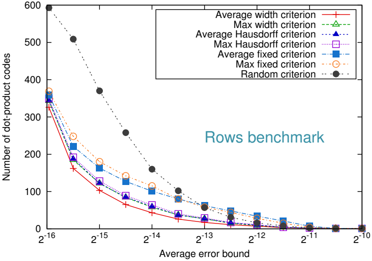
Efficiency of the distance-based heuristic

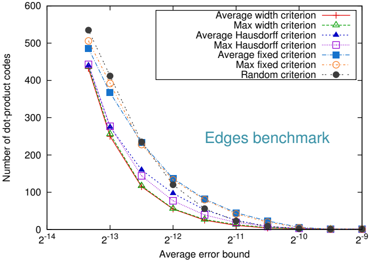
- Example of 6×6 matrix multiplication

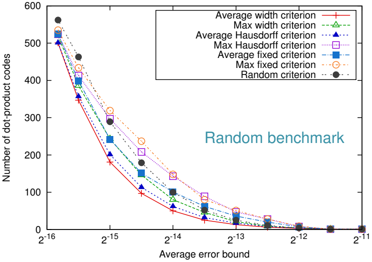


Impact of the metric on the tradeoff strategy









Outline of the talk

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: triangular matrix inversion and Cholesky decomposition

Inputs

- A lower triangular matrix B of interval fixed-point variables

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

Output

- Fixed-point code (C, VHDL, ...) that evaluates the inverse

$$N' = (B')^{-1}, \quad \text{where} \quad B' \in B$$

- Accuracy certificate (verifiable by a formal proof checker)

Inputs

- A matrix M of interval fixed-point variables

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

Output

- Fixed-point code (C, VHDL, ...) that computes the decomposition

$$B' = chol(M'), \quad \text{where} \quad M' \in M \quad \text{and}$$

- 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

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

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

Figure: Dependencies of the coefficient $b_{4,2}$ in the inversion and decomposition of a 6×6 matrix.

The dilemma of the division output format

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

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7

y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7

Multiplication

z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}

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

Division

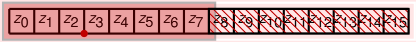
z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}	z_{11}	z_{12}	z_{13}	z_{14}	z_{15}

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

The dilemma of the division output format

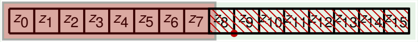
- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:

Multiplication



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

Division

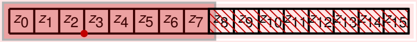


- Keeping the upper half of the result.
- $\mathbf{Err}_{/} \in [-2, 2]$

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.
- $\mathbf{Err}_\times \in [-2^{-5}, 2^{-5}]$

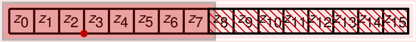
Division

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

The dilemma of the division output format

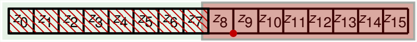
- Consider two fixed-point variables in the formats $\mathbf{Q}_{2.6}$ and $\mathbf{Q}_{1.7}$:

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 $\mathbf{Q}_{1.7}$:

$x_0\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7$ $\qquad$ $y_0\ y_1\ y_2\ y_3\ y_4\ y_5\ y_6\ y_7$

Multiplication

$z_0\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9\ z_{10}\ z_{11}\ z_{12}\ z_{13}\ z_{14}\ z_{15}$

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

Division

$z_0\ z_1\ z_2\ z_3\ z_4\ z_5\ z_6\ z_7\ z_8\ z_9\ z_{10}\ z_{11}\ z_{12}\ z_{13}\ z_{14}\ z_{15}$

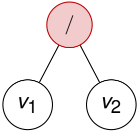
- Taking more risk of overflow!!
- $\mathbf{Err}_/ \in [-2^{-6}, 2^{-6}]$

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

A propagation rule and an implementation of division



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

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

Given $v_1 = V_1 \cdot 2^{-f_1}$ and $v_2 = V_2 \cdot 2^{-f_2}$, how to determine $\mathbf{Err}_/$?

Naive approach

- Compute
$$\frac{v_1}{v_2} = \frac{V_1 \cdot 2^{-f_1}}{V_2 \cdot 2^{-f_2}} = \frac{V_1}{V_2} \cdot 2^{-(f_1-f_2)}$$
- $\mathbf{Err}_/ = [-2^{-(f_1-f_2)}, 2^{-(f_1-f_2)}]$

Accurate approach

- Compute $\frac{v_1}{v_2} = \frac{V_1 \cdot 2^{\eta}}{V_2} \cdot 2^{-(f_1-f_2+\eta)}$
- $\mathbf{Err}_/ = [-2^{-(f_1-f_2+\eta)}, 2^{-(f_1-f_2+\eta)}]$

Comparison of the two implementations of division

- Consider x a 4-bit fixed-point variable in the format $\mathbf{Q}_{1.3}$ with $X = (0101)_2 = (5)_{10}$.
  - The value of x is 0.625
- Consider y a 4-bit fixed-point variable in the format $\mathbf{Q}_{2.2}$ with $X = (0110)_2 = (6)_{10}$.
  - The value of y is 1.5
- The mathematical value for $\frac{x}{y}$ is given by $\frac{x}{y} = 0.41666\ldots$

Naive approach

$$
\begin{array}{llll}
 & \boxed{0}\boxed{1}\boxed{0}\boxed{1} & X = 5 & x = 0.625 \\
X/Y & \boxed{0}\boxed{1}\boxed{1}\boxed{0} & Y = 6 & y = 1.5 \\
\hline
 & \boxed{0}\boxed{0}\boxed{0}\boxed{0} & Z = 0 & z = 0
\end{array}
$$

- $\mathbf{Err}_{/} \in [-2^{-1}, 2^{-1}] = [-0.5, 0.5]$

Comparison of the two implementations of division

- Consider x a 4-bit fixed-point variable in the format $\mathbf{Q}_{1.3}$ with $X = (0101)_2 = (5)_{10}$.
  - The value of x is 0.625
- Consider y a 4-bit fixed-point variable in the format $\mathbf{Q}_{2.2}$ with $X = (0110)_2 = (6)_{10}$.
  - The value of y is 1.5
- The mathematical value for $\frac{x}{y}$ is given by $\frac{x}{y} = 0.41666\ldots$

Naive approach

	0 1 0 1	$X = 5$	$x = 0.625$
	0 1 1 0	$Y = 6$	$y = 1.5$
X/Y	0 0 0 0	$Z = 0$	$z = 0$

- $\mathbf{Err}_/ \in [-2^{-1}, 2^{-1}] = [-0.5, 0.5]$

Accurate approach

	0 1 0 1	$X = 5$	$x = 0.625$
	0 0 1 0 1 0 0 0 0	$X' = 40$	$x' = 0.625$
	0 1 1 0	$Y = 6$	$y = 1.5$
X/Y	0 1 1 0	$Z = 6$	$z = 0.375$

- $\mathbf{Err}_/ \in [-2^{-4}, 2^{-4}] = [-0.0625, 0.0625]$

Overview of synthesis process

- Two main difficulties of the synthesis process

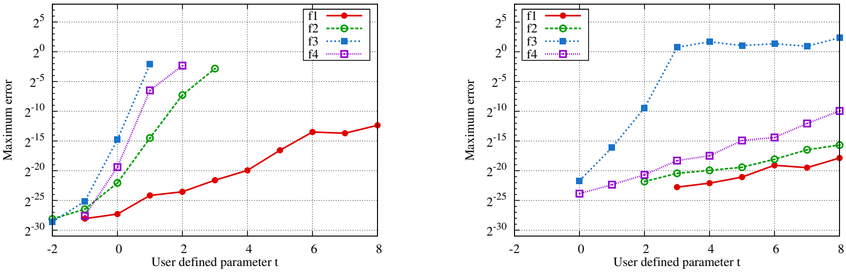
  1. compared to matrix multiplication: the format of a given matrix coefficient depends directly upon the ones of previous computed coefficients
  2. the parameter η must be chosen at synthesis-time

Overview of synthesis process

- 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. the parameter η must be chosen at synthesis-time

- Instead of choosing the parameter η:
  - we fix the expected output of the operator,
  - and we decide the parameter η accordingly.

Impact of the output format of division



(a) Cholesky 5×5.

(b) Triangular 10×10.

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

We tested with multiple means to set the format of output of division

$$f_1(i_1, i_2) = t, \; f_2(i_1, i_2) = \min(i_1, i_2) + t,$$

$$f_3(i_1, i_2) = \max(i_1, i_2) + t, \text{ and } f_4(i_1, i_2) = \lfloor (i_1 + i_2)/2 \rfloor + t,$$

How fast is generating triangular matrix inversion codes?

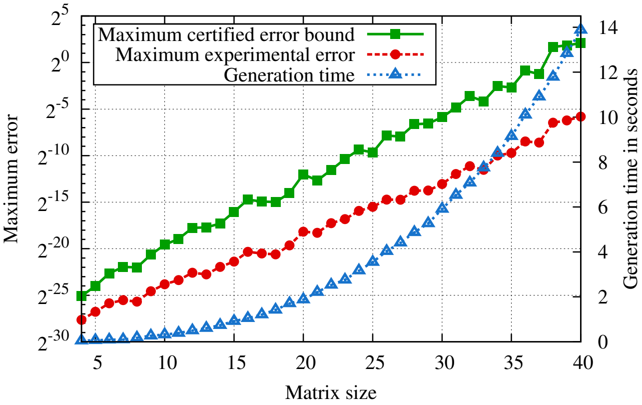
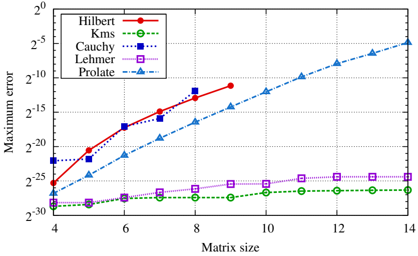


Figure: Comparison of the error bounds and experimental errors together with generation time, for the inversion of triangular matrices of size 4 to 40.

Decomposing some well known matrices



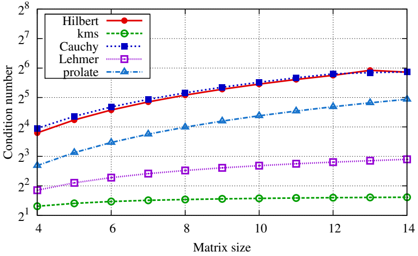


Figure: Maximum errors measured when computing the Cholesky decomposition of various kinds of matrices for sizes varying from 4 to 14.

Decomposing some well known matrices

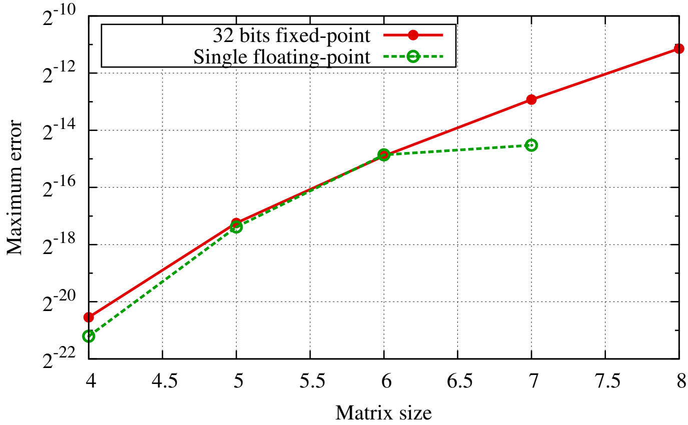


Figure: Maximum errors of the Cholesky decomposition of Hilbert matrix for sizes varying from 4 to 8.

Outline of the talk

The FPLA tool

- FPLA: Fixed-Point Linear Algebra
- Automated code synthesis for linear algebra basic block
  - → matrix multiplication,
  - → triangular matrix inversion,
  - → and Cholesky decomposition
- More information on FPLA are available on its webpage

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

The FPLA tool

- FPLA: Fixed-Point Linear Algebra
- Automated code synthesis for linear algebra basic block
  - → matrix multiplication,
  - → triangular matrix inversion,
  - → and Cholesky decomposition
- More information on FPLA are available on its webpage

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

Let us now have a try on the FPLA tool

Conclusion remarks and future work

We are close to our initial goal of fixed-point code synthesis for matrix inversion.

Work done so far

- New algorithm to synthesize codes that satisfy accuracy/code size tradeoffs for matrix multiplication
  - matrices of size up to 80 in few minutes
- Approach for the synthesis of triangular matrix inversion and Cholesky decomposition
  - matrices of size up to 40 in few minutes
- These algorithms are implemented in the FPLA tool

Conclusion remarks and future work

We are close to our initial goal of fixed-point code synthesis for matrix inversion.

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
  - find trade-offs between code size and accuracy

Journée *Arithmétique pour le traitement de signal et de l'image* du GDR ISIS
Paris, July 3rd, 2014

Code synthesis for linear algebra basic blocks in fixed-point arithmetic

The cases of matrix multiplication and inversion

Amine Najahi
Advisors: Matthieu Martel and Guillaume Revy

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