4th International Conference on Pervasive and Embedded Computing and Communication Systems, PECCS-2014
Lisbon, January 7-9th, 2014

Code Size and Accuracy-Aware Synthesis of Fixed-Point Programs for Matrix Multiplication

Matthieu Martel **Amine Najahi** 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 matrix multiplication
  - → 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

- Novel tradeoff algorithm for the synthesis of matrix multiplication
  - → up to 50% code size reduction for some benchmarks
  - → while satisfying the accuracy criterion

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)

Outline of the talk

1. Background and straightforward approaches

2. A novel tradeoff algorithm for the synthesis of matrix multiplication codes

3. Experimental results

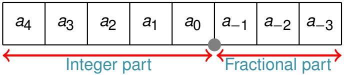
4. Concluding remarks and future work

Outline of the talk

1. Background and straightforward approaches
2. A novel tradeoff algorithm for the synthesis of matrix multiplication codes
3. Experimental results
4. Concluding remarks and future work

Background on fixed-point arithmetic

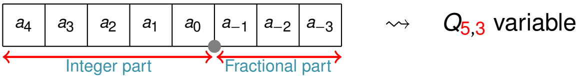
- Principle: interpret bit packets as integers coupled with an implicit scale factor



$\rightsquigarrow$ $Q_{5,3}$ variable

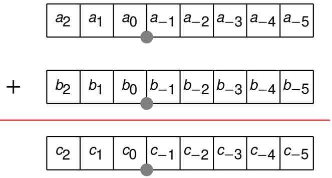
Background on fixed-point arithmetic

- Principle: interpret bit packets as integers coupled with an implicit scale factor



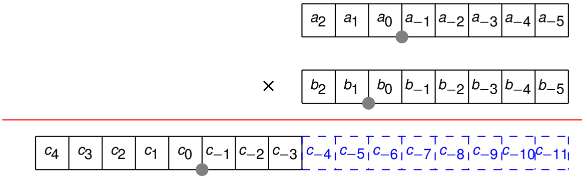
Addition

- The operands have to be in the same fixed-point format



Multiplication

- The product of a $Q_{v,w}$ variable by a $Q_{x,y}$ variable yields a $Q_{v+x,w+y}$ variable



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
    - ⇝ 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:

```
1: for 1 < i ≤ m do
2:     for 1 < j ≤ p do
3:         DPSynthesis(A_{i,:}, B_{:,j})
4:     end for
5: end for
6: Check 𝒞1 and 𝒞2
```

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 ≤ m do
    for 1 < j ≤ p do
        DPSynthesis(A_{i,:}, B_{:,j})
    end for
end for
Check C_1 and 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:

$\mathscr{U} = A_{1,:} \cup A_{2,:} \cup \cdots \cup A_{m,:}$, with $\mathscr{U} \in \mathbb{F}ix^{1 \times n}$

$\mathscr{V} = B_{:,1} \cup B_{:,2} \cup \cdots \cup B_{:,p}$, with $\mathscr{V} \in \mathbb{F}ix^{n \times 1}$

DPSynthesis$(\mathscr{U}, \mathscr{V})$

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	$Q_{11,21}$	$Q_{12,20}$	$Q_{2,30}$	$Q_{2,30}$	$Q_{11,21}$	$Q_{3,29}$	$Q_{2,30}$	$Q_{5,27}$

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	$Q_{11,21}$	$Q_{12,20}$	$Q_{2,30}$	$Q_{2,30}$	$Q_{11,21}$	$Q_{3,29}$	$Q_{2,30}$	$Q_{5,27}$

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

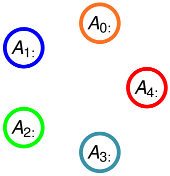
Dot-product	$A_{1,:} \cdot B_{:,1}$	$A_{1,:} \cdot B_{:,2}$	$A_{2,:} \cdot B_{:,1}$	$A_{2,:} \cdot B_{:,2}$
Evaluated using	$\text{DPCode}_{\mathcal{U},\mathcal{V}}$			
Output format	$Q_{26,6}$			
Certified error	$\approx 2^{-5}$			
Maximum error	$\approx 2^{-5}$			
Average error	$\approx 2^{-5}$			

Outline of the talk

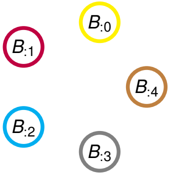
1. Background and straightforward approaches
2. A novel tradeoff algorithm for the synthesis of matrix multiplication codes
3. Experimental results
4. Concluding remarks and future work

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)

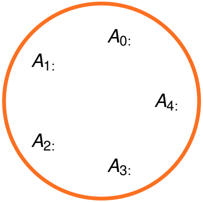


$$C = A \cdot B = \begin{pmatrix} \text{DPCode}_{0,0}(A_{0,:}, B_{:,0}) & \text{DPCode}_{0,1}(A_{0,:}, B_{:,1}) & \text{DPCode}_{0,2}(A_{0,:}, B_{:,2}) & \text{DPCode}_{0,3}(A_{0,:}, B_{:,3}) & \text{DPCode}_{0,4}(A_{0,:}, B_{:,4}) \\ \text{DPCode}_{1,0}(A_{1,:}, B_{:,0}) & \text{DPCode}_{1,1}(A_{1,:},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

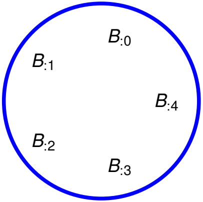
Tradeoff algorithms

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

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



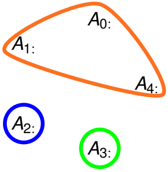
Compact algorithm:
(1 dot-product code)



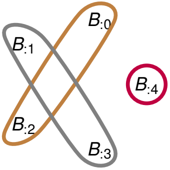
$$C = A \cdot B = \begin{pmatrix} \text{DPCode}_{0,0}(A_{0,:}, B_{:,0}) & \text{DPCode}_{0,0}(A_{0,:}, B_{:,1}) & \text{DPCode}_{0,0}(A_{0,:}, B_{:,2}) & \text{DPCode}_{0,0}(A_{0,:}, B_{:,3}) & \text{DPCode}_{0,0}(A_{0,:}, B_{:,4}) \\ \text{DPCode}_{0,0}(A_{1,:}, B_{:,0}) & \text{DPCode}_{0,0}(A_{1,:},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

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)



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

Tradeoff algorithms

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

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

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

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

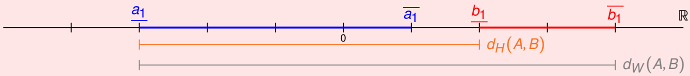
Width criterion

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

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

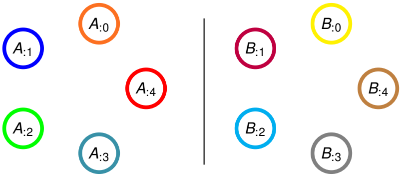
Output:

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:

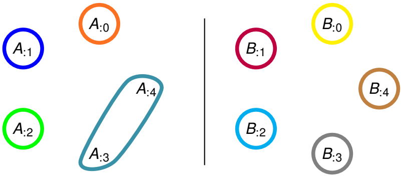
Two matrices $A \in \mathbb{F}\text{ix}^{m \times p}$ and $B \in \mathbb{F}\text{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

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

Algorithm:

```
S_A ← {A_{0,:}, ..., A_{m-1,:}}
S_B ← {B_{:,0}, ..., B_{:,n-1}}
while C_1 is satisfied do
    (u_A, v_A), d_A ← findClosestPair(S_A, d)
    (u_B, v_B), d_B ← findClosestPair(S_B, d)
    if d_A ≤ d_B then
        remove(u_A, v_A, S_A)
        insert(u_A ∪ v_A, S_A)
    else
        remove(u_B, v_B, S_B)
        insert(u_B ∪ v_B, S_B)
    end if
    for (A_i, B_j) ∈ S_A × S_B do
        DPSynthesis(A_i, B_j)
    end for
end while
/* Revert the last merging step, and check the bound C_2. */
```



20 DPcodes

$\mathscr{C}_1$ is satisfied

Closest pair strategy

Input:

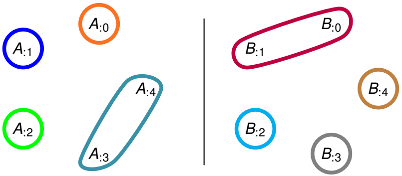
Two matrices $A \in \mathbb{F}\text{ix}^{m\times p}$ and $B \in \mathbb{F}\text{ix}^{p\times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

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**
  $(u_A, v_A), d_A \leftarrow$ *findClosestPair*$(\mathscr{S}_A, d)$
  $(u_B, v_B), d_B \leftarrow$ *findClosestPair*$(\mathscr{S}_B, d)$
  if $d_A \le d_B$ **then**
    remove$(u_A, v_A, \mathscr{S}_A)$
    insert$(u_A \cup v_A, \mathscr{S}_A)$
  else
    remove$(u_B, v_B, \mathscr{S}_B)$
    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**
    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:

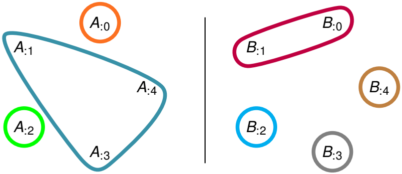
Two matrices $A \in \mathbb{F}\text{ix}^{m \times p}$ and $B \in \mathbb{F}\text{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

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

Closest pair strategy

Input:

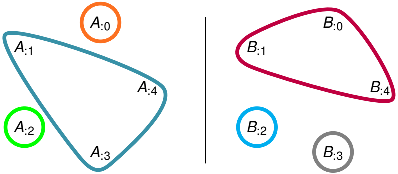
Two matrices $A \in \mathbb{F}\text{ix}^{m\times p}$ and $B \in \mathbb{F}\text{ix}^{p\times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

Output:

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

Algorithm:

```
𝒮_A ← {A_{0,:}, ..., A_{m−1,:}}
𝒮_B ← {B_{:,0}, ..., B_{:,n−1}}
while 𝒞_1 is satisfied do
    (u_A, v_A), d_A ← findClosestPair(𝒮_A, d)
    (u_B, v_B), d_B ← findClosestPair(𝒮_B, d)
    if d_A ≤ d_B then
        remove(u_A, v_A, 𝒮_A)
        insert(u_A ∪ v_A, 𝒮_A)
    else
        remove(u_B, v_B, 𝒮_B)
        insert(u_B ∪ v_B, 𝒮_B)
    end if
    for (A_i, B_j) ∈ 𝒮_A × 𝒮_B do
        DPSynthesis(A_i, B_j)
    end for
end while
/* Revert the last merging step, and check the bound 𝒞_2. */
```



9 DPcodes

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

Closest pair strategy

Input:

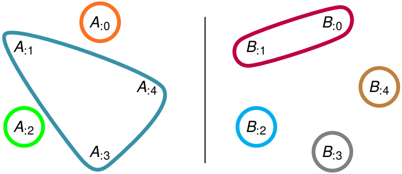
Two matrices $A \in \mathbb{F}\text{ix}^{m \times p}$ and $B \in \mathbb{F}\text{ix}^{p \times n}$
An accuracy bound $\mathscr{C}_1$ (ex. the average error bound is $< \epsilon$)
A code size bound $\mathscr{C}_2$
A metric d

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 $\mathscr{C}_2$. */



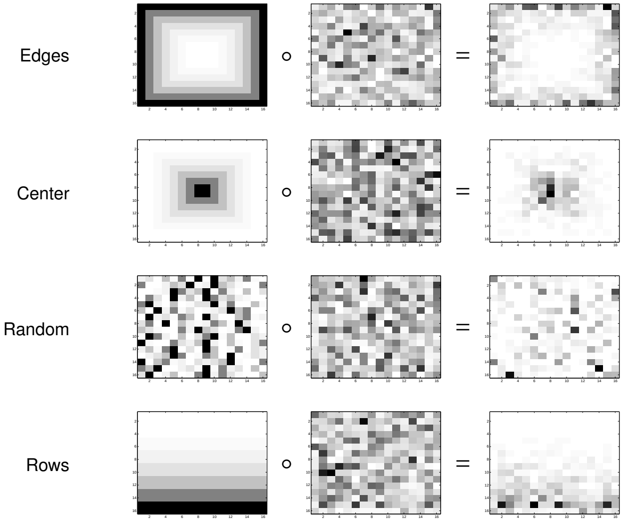
12 DPcodes

$\mathscr{C}_1$ is satisfied

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

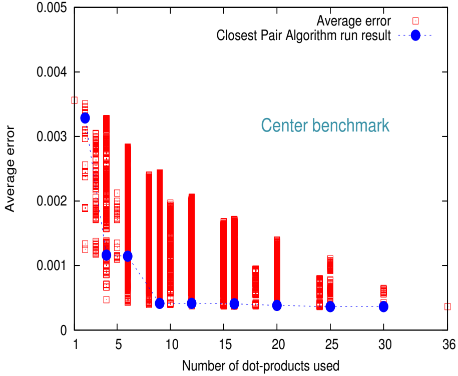
Outline of the talk

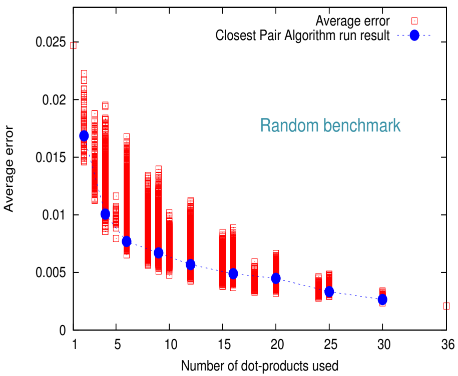
Benchmarks generation methodology



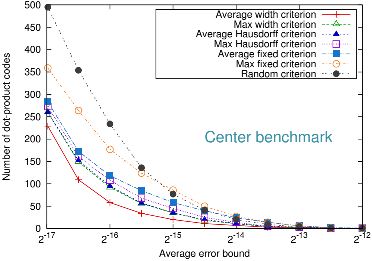
Efficiency of the distance-based heuristic

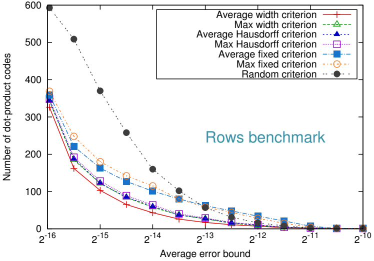
- Example of 6×6 matrix multiplication

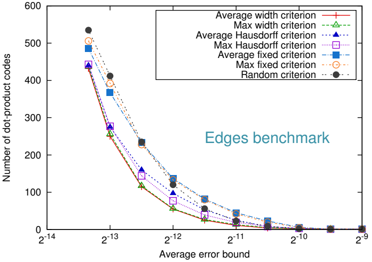


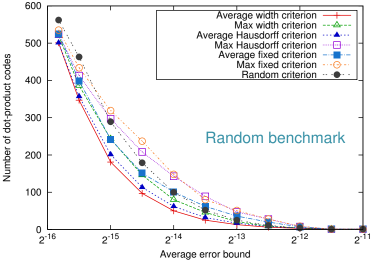


Impact of the metric on the tradeoff strategy









Outline of the talk

Conclusion remarks and future work

Work done so far

- We suggested a new algorithm to synthesize fixed-point codes that finds accuracy/code size tradeoffs
- The algorithm is implemented in the FPLA (*Fixed-Point Linear Algebra*) tool

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

- We are able to synthesize code for matrices of size up to 80 in few minutes

Future work

- Measure the gain in resource usage when the target is an FPGA
- Use similar techniques for other linear algebra basic blocks