Decoding-free Two-Input Arithmetic for Low-Precision Real Numbers

John L. Gustafson, Marco Cococcioni, Federico Rossi, Emanuele Ruffaldi and Sergio Saponara

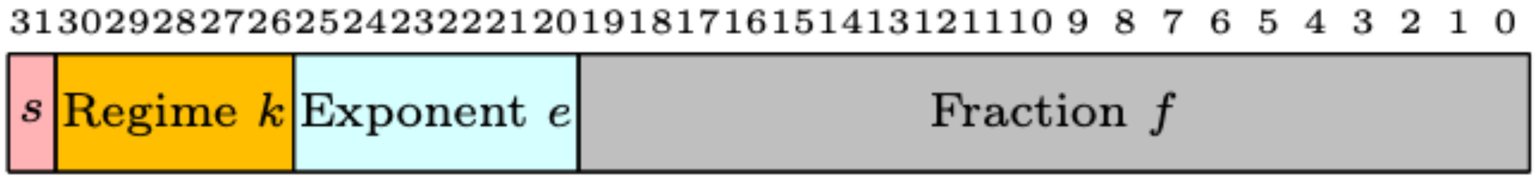
Introduction

- Real numbers have been represented with a scientific notation for nearly a century
  - An integer for the significand
  - An integer for the exponent
- IEEE754 standard has been the guidance for this notation
- This notation heavily impacts the hardware that executes two-input arithmetic operations
- In this work we tried to overcome this difficulties

Posit numbers

- A number in the posit format is *n* bits length, with $n \geq 2$
- It only holds two exceptions: 0 and Not a Real (NaR)
- It can be configured in the number of bits *n* and maximum exponent bits ***es***

$$r = (1 - 3s + f) \times 2^{(1-2s)\times(2^{es}k+e+s)}.$$



Standard two-input arithmetic

Posit	Value	Posit	Value
1000	NaR	0000	0
1001	-4	0001	$1/4$
1010	-2	0010	$1/2$
1011	$-3/2$	0011	$3/4$
1100	-1	0100	1
1101	$-3/4$	0101	$3/2$
1110	$-1/2$	0110	2
1111	$-1/4$	0111	4

- Simple example: Posit<4,0> format
- We have 16 different configurations
- The mapping between the bit configuration and the value is *bijective*
- The mapping is also monotone if we consider bit configurations as **2's complement signed integers**

Standard two-input arithmetic

×	1/4	1/2	3/4	1	3/2	2	4
1/4	1/16	1/8	3/16	1/4	3/8	1/2	1
1/2	1/8	1/4	3/8	1/2	3/4	1	2
3/4	3/16	3/8	9/16	3/4	9/8	3/2	3
1	1/4	1/2	3/4	1	3/2	2	4
3/2	3/8	3/4	9/8	3/2	9/4	3	6
2	1/2	1	3/2	2	3	4	8
4	1	2	3	4	6	8	16

- Multiplication table for Posit<4,0>
- We ignore negative values for symmetry
- Since multiplication is commutative the table is symmetric

Standard two-input algorithm

1. Test for exceptional cases
2. Decode each input into signficand and exponent, both stored as signed integers ⚠️
3. Use logic circuits to implement the binary operation (e.g. addition, subtraction etc…)
4. Encode the result into the appropriate format, rounding and normalizing the ouput of step 3.

Motivation

- The input decoding and output normalization phase are costly
- Depending on the format, several special cases must be tested during both decoding and normalization
- Several logic levels between input and output can increase latency of the overall arithmetic circuit

- Our idea: transform input operands so that two-input arithmetic does not need decoding but only integer arithmetic (= sum of integer numbers).

Core idea for decoding free arithmetic

- Map each integer value of the input operands to another space of integer values
- Chose the mapping so that sum in the new space can be reversely mapped to the correspondent binary operation in the original space
- Example: instead of multiplying two values a, b map them to a', b' so that $a' + b'$ can be reversely mapped to $a * b$, without decoding a and b.

Mathematical background

- Start from X, Y two finite sets of real numbers.
- X^* and Y^* are the sets of bits strings that digitally encode X and Y. The mapping between X, X^* and Y, Y^* is bijective, as seen before.
- ∇ is any binary operation between an element of X and an element of Y
- Z is the set of real values $z_{ij} = x_i \nabla y_j$
- $\hat{Z}$ is the set of real values obtained from the rounding of z_{ij} to obtain representable values in X and Y.

Mathematical background

- L^x and L^y are ordered sets of natural numbers
- Suppose we have a bijective f_x that maps X into L^x and f_y Y to L^y (through their encoded X^* and Y^* sets)
- Each x is uniquely mapped to a value in L^x (the same for y,Ly)
- L^z is the set of all distinct sums between L^x and L^y and fz is the mapping between L^z and Z

Mathematical background

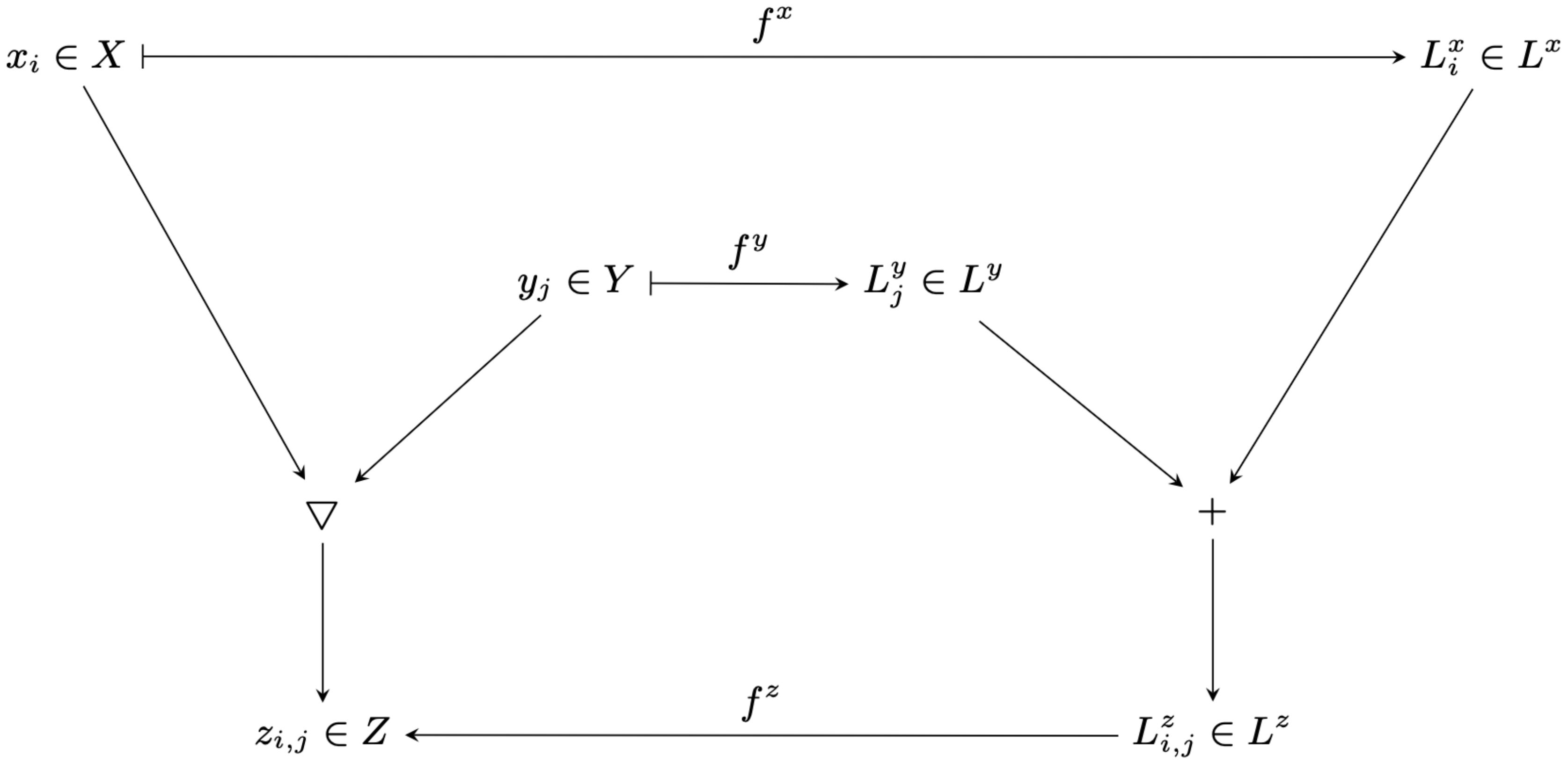
- We must ensure that for any pair xi,yj and xp,yq whose binary operation results differ we have

$$L_i^x + L_j^y \neq L_p^x + L_q^y$$

- If this holds we obtain the relation representing our method:

$$z_{i,j} = x_i \nabla y_j = f^z(f^x(x_i) + f^y(y_j))$$

Overview


$x_i \in X$
f^x
$L_i^x \in L^x$
$y_j \in Y$
f^y
$L_j^y \in L^y$
$\bigtriangledown$
$+$
$z_{i,j} \in Z$
f^z
$L_{i,j}^z \in L^z$

Obtaining the mapping

- When choosing the mapping we must enforce the requirement that different results are mapped into different sums in Lz (but not necessarily the opposite).
- The idea is to set-up an integer programming problem to solve this assignment.
- If we can provide an initial feasible solution to the problem, under the right assumptions, we can state that we always have an optimal solution for it.

General Problem

$$
\begin{aligned}
\min \quad & \sum_i L_i^x + \sum_j L_j^y \\
\text{s.t.} \quad & L_1^x \geq 0 \\
& L_1^y \geq 0 \\
& L_{i_1}^x \neq L_{i_2}^x && \forall i_1 \neq i_2 \\
& L_{j_1}^y \neq L_{i_2}^y && \forall j_1 \neq j_2 \\
& L_i^x + L_j^y \neq L_p^x + L_q^y && \forall i, j, p, q \ s.t. \ x_i \nabla y_j \neq x_p \nabla y_q \\
& L_i^x, L_j^y \in \mathbb{Z} && \forall i, \forall j
\end{aligned}
$$

Monotonic and commutative operations

$$
\begin{aligned}
\min \quad & \sum_i L_i^x + \sum_j L_j^y \\
\text{s.t.} \quad & L_1^x \geq 0 \\
& L_1^y \geq 0 \\
& L_i^x \geq L_j^x + 1 \qquad i > j \\
& L_i^y \geq L_j^y + 1 \qquad i > j \\
& L_i^x + L_j^y = L_j^x + L_i^y \quad \forall i, \forall j \\
& L_i^x + L_j^y + 1 \leq L_p^x + L_q^y \quad \forall i, j, p, q \ s.t. \ x_i \nabla y_j < x_p \nabla y_q \\
& L_i^x, L_j^y \in \mathbb{Z} \qquad \forall i, \forall j
\end{aligned}
$$

Application to $Posit\langle 4,0 \rangle$

- We apply the method presented until now to a 4-bit posit, for simplicity
- We consider the four arithmetic operations $+, -, \times, /$
- We consider the strategies for the solution (i.e. ordering of the resulting Lx, Ly sets)
- We evaluate the result, comparing it to a traditional 2D look-up table

Strategies for solution

	L^x	L^y
SUM	Increasing	Increasing
MUL	Increasing	Increasing
SUB	Decreasing	Increasing
DIV	Increasing	Decreasing

Optimal problem solution

operation	L^x	L^y
$+$	$\{0, 1, 2, 3, 5, 6, 11\}$	$\{0, 1, 2, 3, 5, 6, 11\}$
$\times$	$\{0, 2, 3, 4, 5, 6, 8\}$	$\{0, 2, 3, 4, 5, 6, 8\}$
$-$	$\{0, 1, 2, 3, 5, 6, 7\}$	$\{15, 14, 13, 12, 10, 8, 0\}$
$/$	$\{0, 2, 3, 4, 5, 6, 8\}$	$\{8, 6, 5, 4, 3, 2, 0\}$

Multiplication Example

- Let us take the multiplication results
- We have 3 ordered sets of real numbers
  - $X = Y = \{\frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, \frac{3}{2}, 2, 4\}$
  - $\hat{Z} = \{\frac{1}{4}, ½, ¾, 1, 3/2, 2, 4\}$
- 3 ordered sets of natural numbers
  - Lx = {0, 2, 3, 4, 5, 6, 8}
  - Ly = {0, 2, 3, 4, 5, 6, 8}
  - Lz = {0, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16}

Multiplication Example

$z_{i,j}$	$\hat{z}_{i,j}$	$L^z_{i,j}$ $(= L^x_i + L^y_j)$	L^z_k	w_k
1/16	1/4	0	**0**	**1/4**
1/8	1/4	2	**2**	**1/4**
1/8	1/4	2		
3/16	1/4	3	**3**	**1/4**
3/16	1/4	3		
1/4	1/4	4	**4**	**1/4**
1/4	1/4	4		
1/4	1/4	4		
3/8	1/4	5	**5**	**1/4**
3/8	1/4	5		
3/8	1/4	5		
3/8	1/4	5		
1/2	1/2	6	**6**	**1/2**
1/2	1/2	6		
1/2	1/2	6		
1/2	1/2	6		
9/16	1/2	6		

- We have also the correspondence table from L^z to z built using the previous sets
- A group in the table corresponds to a single mapping entry (in bold)

Multiplication Example – at work!

x_i	L_i^x	y_j	L_j^y	$L_{i,j}^z$ $(= L_i^x + L_j^y)$	$z_{i,j}$ $(= x_i \times y_j))$	$\hat{z}_{i,j}$ $(= \texttt{cast}(x_i \times y_j)$
$\frac{1}{2}$	2	$\frac{1}{4}$	0	2	$\frac{1}{8}$	$\frac{1}{4}$
$\frac{1}{2}$	2	$\frac{1}{2}$	2	4	$\frac{1}{4}$	$\frac{1}{4}$
$\frac{1}{2}$	2	$\frac{3}{4}$	3	5	$\frac{3}{8}$	$\frac{1}{4}$
$\frac{1}{2}$	2	$\frac{3}{2}$	5	7	$\frac{3}{4}$	$\frac{3}{4}$
$\frac{1}{2}$	2	2	6	8	1	1
$\frac{1}{2}$	2	4	8	10	2	2

$z_{i,j}$	$\hat{z}_{i,j}$	$L_{i,j}^z$ $(= L_i^x + L_j^y)$	L_k^z	w_k
1/16	1/4	0	**0**	**1/4**
1/8	1/4	2	**2**	**1/4**
1/8	1/4	2		
3/16	1/4	3	**3**	**1/4**
3/16	1/4	3		
1/4	1/4	4	**4**	**1/4**
1/4	1/4	4		
1/4	1/4	4		
3/8	1/4	5	**5**	**1/4**
3/8	1/4	5		
3/8	1/4	5		
3/8	1/4	5		
1/2	1/2	6	**6**	**1/2**
1/2	1/2	6		
1/2	1/2	6		
1/2	1/2	6		
9/16	1/2	6		

Evaluation of results

- We compare our solution to a typical 2D look-up table
- This table is indexed by the 4 bits of the Posit4,0 encoding integer, therefore it has $2^{2*4} = 256$ entries
- Each entry contains the result, therefore it holds 4 bits.
- In total the 2D LUT occupies 1024 bits at most

Quality metrics

Total gate count AND-OR for each operation for Posit$\langle 4, 0 \rangle$.

	Total gates for L^x	Total gates for L^y	Total gates for L^z	**Grand total gates**	Grand total gates of the naïve solution	Gate reduction
$+$	10	10	11	**31**	138	4.4×
$\times$	7	7	9	**23**	138	6×
$-$	8	5	5	**18**	138	7.6×
$/$	7	7	9	**23**	138	6×

Conclusions

- We presented a method to perform two-input arithmetic without decoding the operands
- We proposed a general integer programming model that solves the problem of producing mapping for operands and result
- We applied the method to a Posit4,0 format
- We compared a logic synthesis of the obtained mapping against a 2D Look-Up table, being able to reduce logic gates up to 7 times

THANKS

Contacts: federico.rossi@ing.unipi.it