A Matrix-Multiply Unit for Posits in Reconfigurable Logic Using (OPEN)CAPI

Speaker: Jianyu Chen
Author : Jianyu Chen
Zaid Al-Ars
Peter Hofstee

Content

- Motivation
- Matrix-multiply unit
- Multiply-accumulator with quire registers
- Performance on CAPI 1.0 interface

Motivation

- Only SW implementations of posit => very slow
- There is no posit multiply-accumulator with quire registers => can increase accuracy
- HW matrix-multiply unit can speed up the calculation significantly (we achieved 1000x speedup)

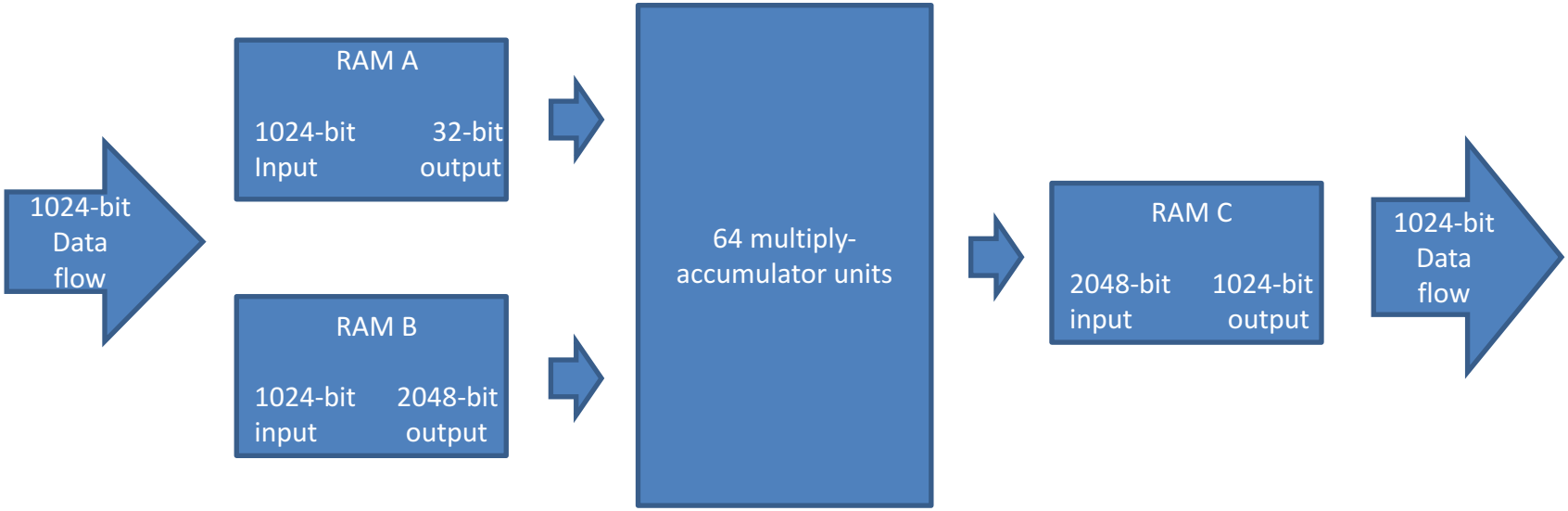
CAPI 1.0 interface

- An interface to connect custom accelerator engine to IBM POWER8 computer
- Accelerator engine can be FPGA platforms
- Only pointers of data will be passed from CPU to FPGA
  - No driver required, API more implementation independent
- Simple to program on API

Introduction of matrix-multiply unit

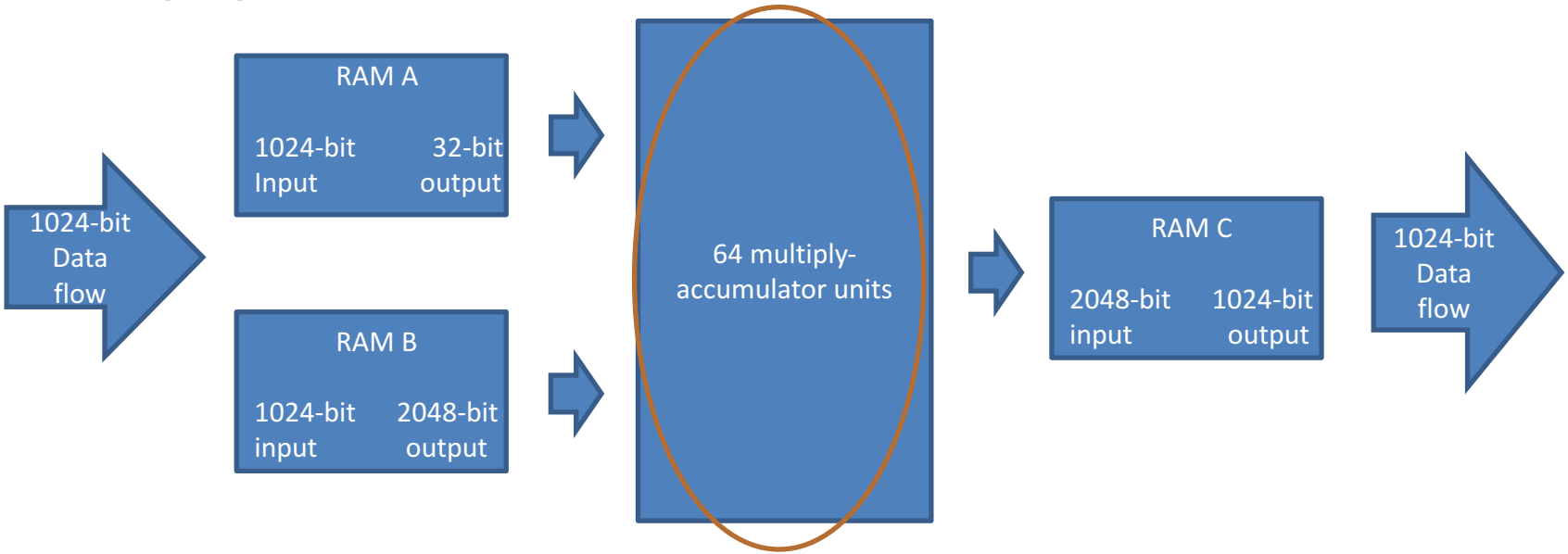
- Work on 32-bit Posit number with es=2
- Use FPGA to multiply two matrices
- Do calculation more accurate than traditional one

Matrix-multiply unit



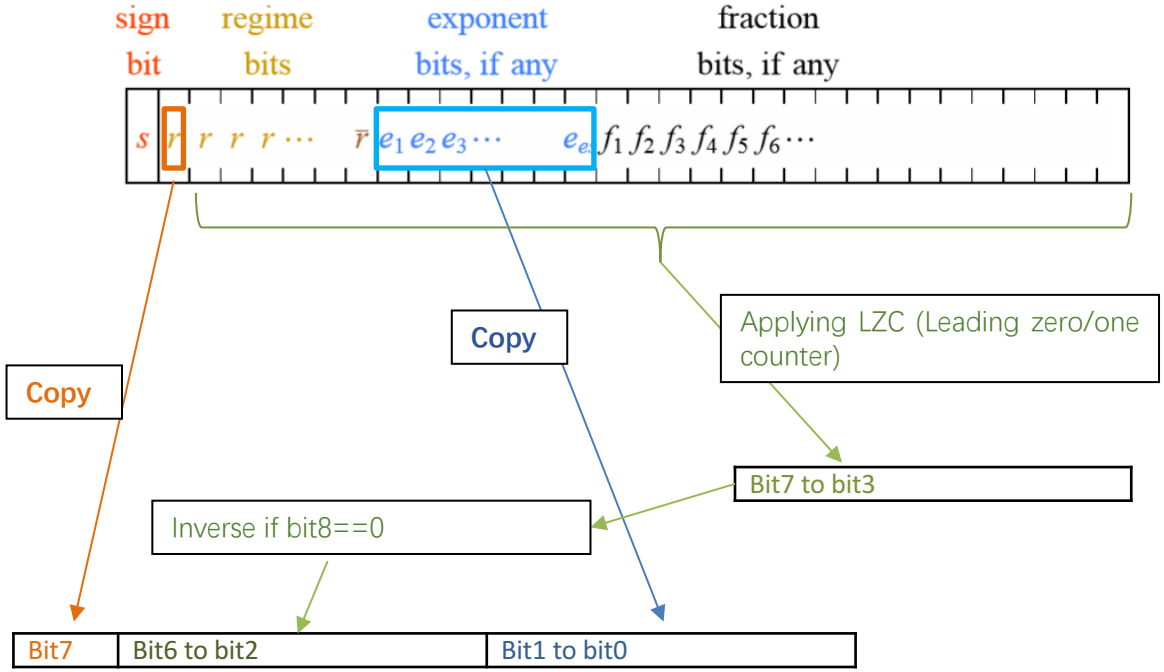
- C = A * B
- Matrix A, B and C stored in RAM A, RAM B and RAM C
- 64 numbers in C will be calculated at the same time

Multiply-accumulator unit



Transform regime bits and exponent bits to exponent value

- Get exponent value in unsigned integer
- Use multiply-accumulator



$$(-1)^{sign\ bit} * 2^{exponent\ value-128} * fraction$$

Transform regime bits and exponent bits to exponent value

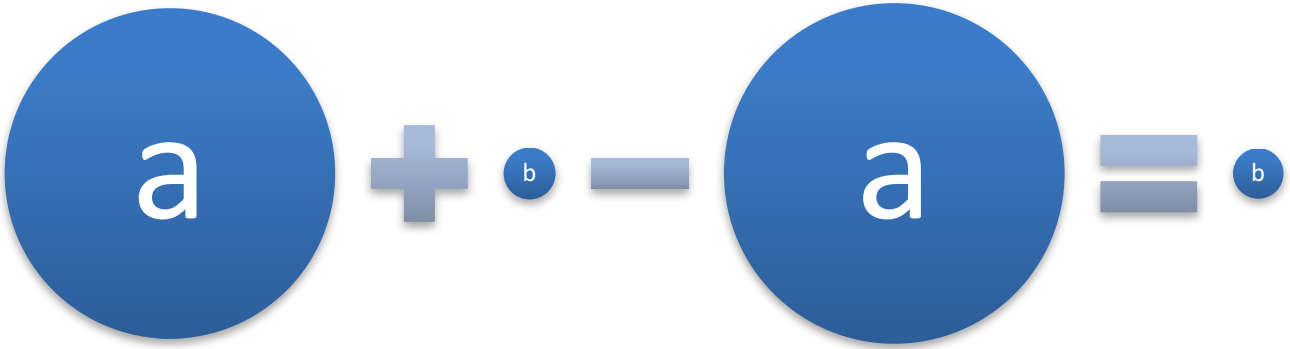
$$(-1)^{sign\ bit} * useed^{k} * 2^{e} * fraction$$

$$useed = 2^{2^{es}} = 16\ (when\ es = 2)$$

$$(-1)^{sign\ bit} * 2^{exponent\ value - 128} * fraction$$

Multiply-accumulator with quire registers

- Maintain a 512-bit accuracy of intermediate results
- Do the rounding after all accumulation
- Example: a + b – a = b (even when a >> b)



Multiply-accumulator with quire register

- This 512 bits represent from 2^-255 to 2^256

Bit 511	Bit 510	Bit 509	Bit 508	…	…	Bit 3	Bit 2	Bit 1	Bit 0
2^256	2^255	2^254	2^253	…	…	2^-252	2^-253	2^-254	2^-255

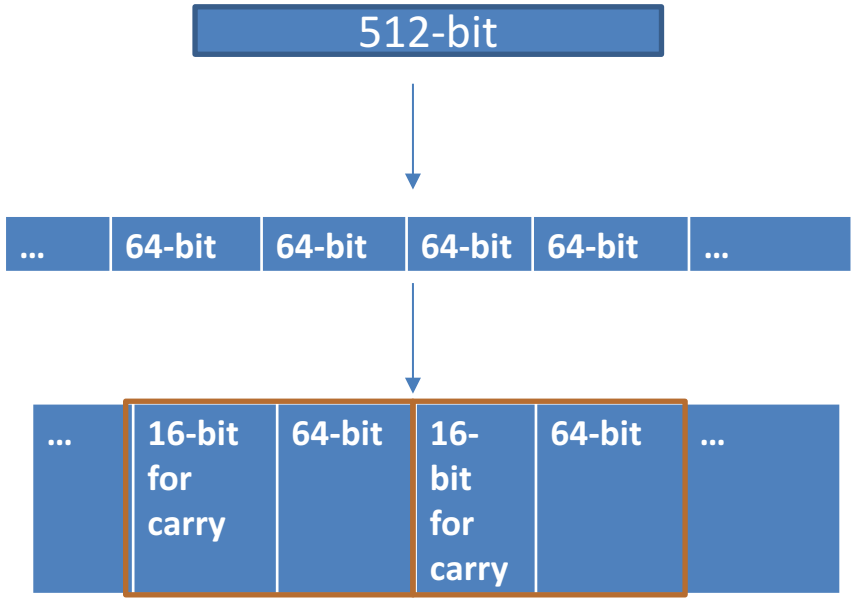
Reasons for low performance

- Adding a number to a 512-bit number directly is very expensive and slow
- Do normalization for a 512-bit number directly is also very expensive and slow

Bit 511	Bit 510	Bit 509	Bit 508	…	…	Bit 3	Bit 2	Bit 1	Bit 0
2^256	2^255	2^254	2^253	…	…	2^-252	2^-253	2^-254	2^-255

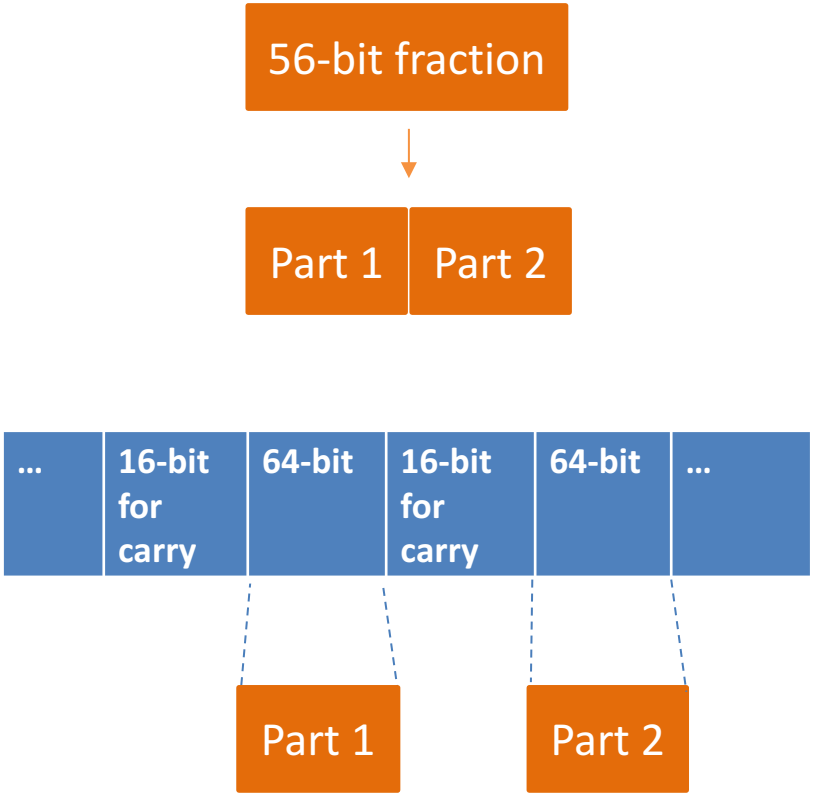
Idea to improve performance

- Divide 512 bits into 8 64-bit blocks
- For each 64-bit block, append a 16-bit carry block on the left



Save the carry for each block

- Adding 56-bit fraction will only influence two or one blocks
- Carry of each 64-bit blocks will be saved in its 16-bit carry
- After all the addition, do the carry for all blocks



Do normalization

0	…	00…000	00…000	00…001XX…XXX	XXX…XXX	XXX…XXX	…
Sign bit	…	64-bit block	64-bit block	64-bit block	64-bit block	64-bit block	…

- Do the carry from right to left
- Only the blocks with Leading ONE and its right neighboring blocks are useful for normalization (if the result is positive)
- Carry and part of normalization will be done at the same time

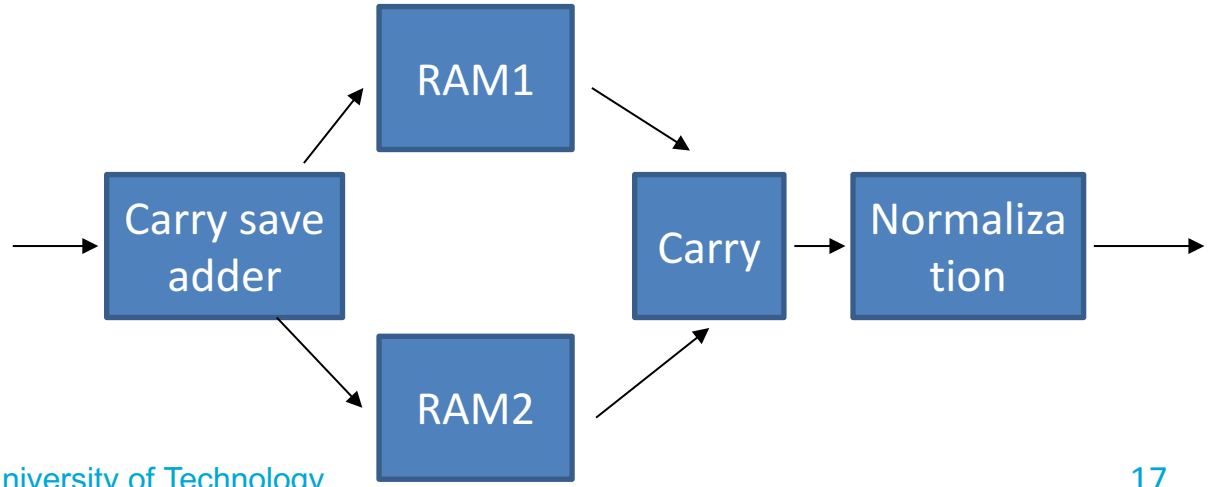
Do normalization

1	…	11…111	11…111	11…110XX…XXX	XXX…XXX	XXX…XXX	…
Sign bit	…	64-bit block	64-bit block	64-bit block	64-bit block	64-bit block	…

- Only the blocks with Leading ZERO and its right neighboring blocks is useful for normalization (if the result is negative)

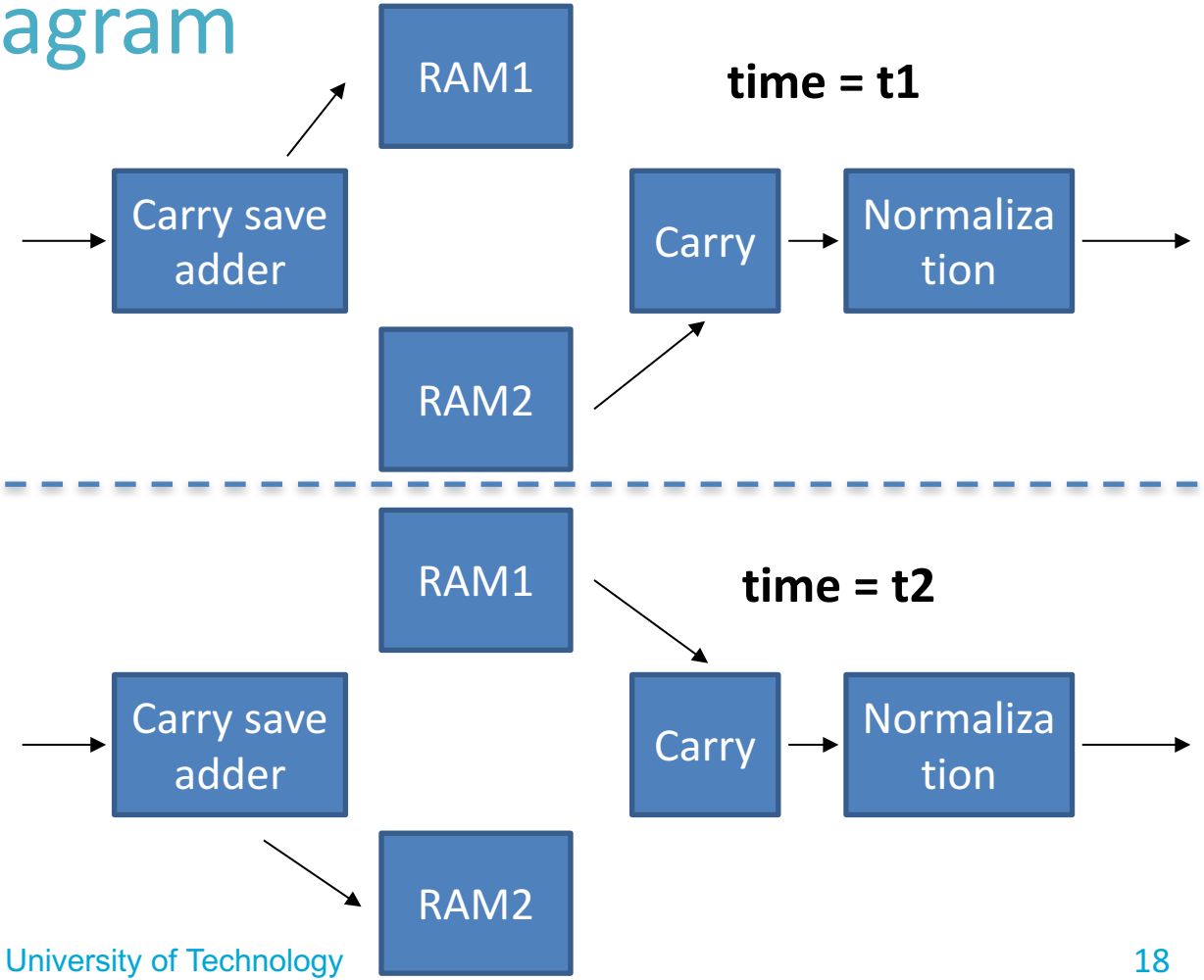
Diagram of the multiply-accumulator

- Consist of two RAMs (two 512-bit quire registers)
- When the “Carry save adder” block uses RAM1, the “Carry” block uses RAM2, and vice versa



Operation diagram

- Two blocks take turns to use 2 RAMs
- Addition and carry can work at the same time



Performance on CAPI 1.0 interface

- 16 Gpop (Giga posit operation per second) when multiply two 128*128 matrices
- More than 1000 times faster than software Julia implementation

1000x faster!!

Using matrix-multiplier on TACC

- There is CAPI 1.0 device available on TACC (Texas Advanced Computing Center) at https://www.tacc.utexas.edu/systems/fabric
- Design of the matrix-multiply unit is available on Github: https://github.com/ChenJianyunp/Unum_matrix_multiplier

Conclusions

- We developed a 16 Gpop HW posit design
  => 1000x speedup
- Open source version available on Github
- Can be used for free on TACC

- Future plans
  - Working towards a 64Gpop version with OpenCAPI