# An Accelerator for Posit Arithmetic Targeting Posit Level 1 BLAS Routines and Pair-HMM

Laurens van Dam, Johan Peltenburg*, Zaid Al-Ars, H. Peter Hofstee

CoNGA'19: Conference for Next Generation Arithmetic 2019
Singapore, March 14, 2019

TU Delft

# Credits

- Paper based on MSc thesis [1] work by Laurens van Dam.
- Computer Engineering Lab @ TU Delft, Netherlands.
- Special thanks to Jinho Lee @ IBM.

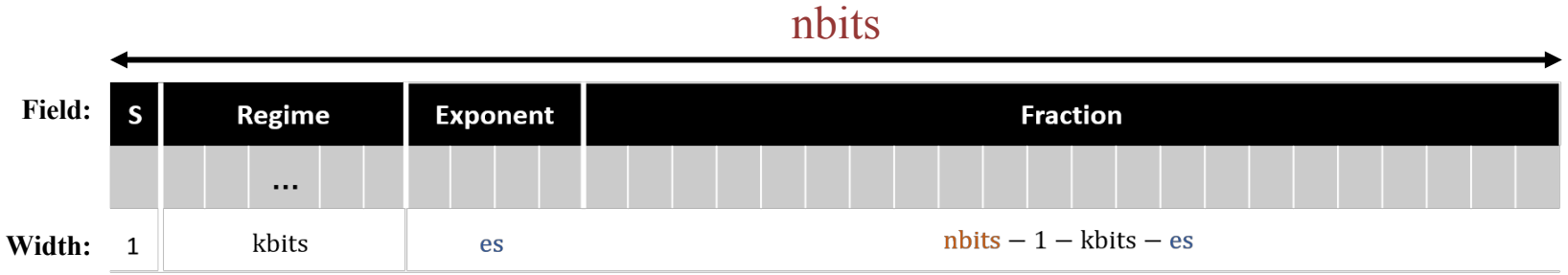
[1] Enabling High Performance Posit Arithmetic Applications Using Hardware Acceleration, L. van Dam, Online: http://resolver.tudelft.nl/uuid:943f302f-7667-4d88-b225-3cd0cd7cf37c

# Outline

- **Posit Arithmetic in Hardware**
- Posit Vector Arithmetic Accelerator
- Pair Hidden Markov Model Accelerator
- Conclusion

# Posit Number Format [1]

- Configurable total and exponent bits

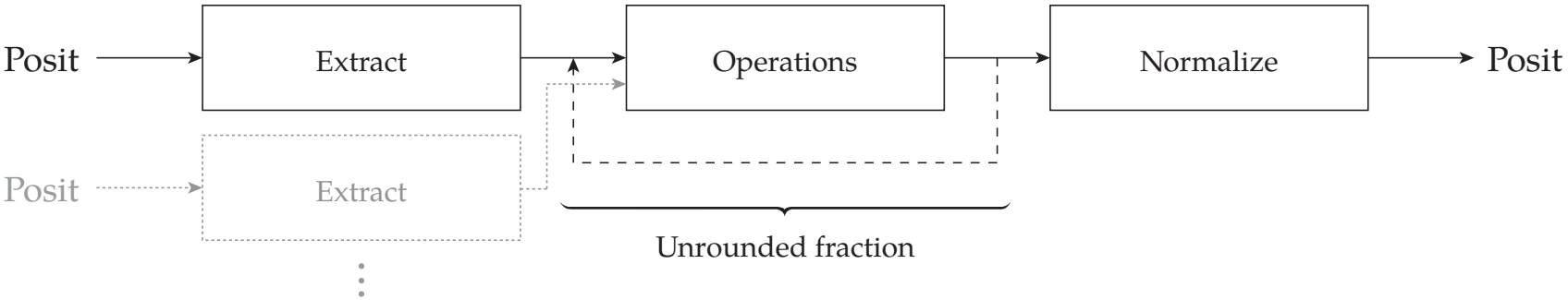



$$\text{value} = (-1)^{s} \cdot (2^{2^{es}})^{\text{regime}} \cdot 2^{\text{exponent}} \cdot (1.\text{fraction})$$

[2] John L. Gustafson. 2017. *Posit Arithmetic*. https://posithub.org/docs/Posits4.pdf

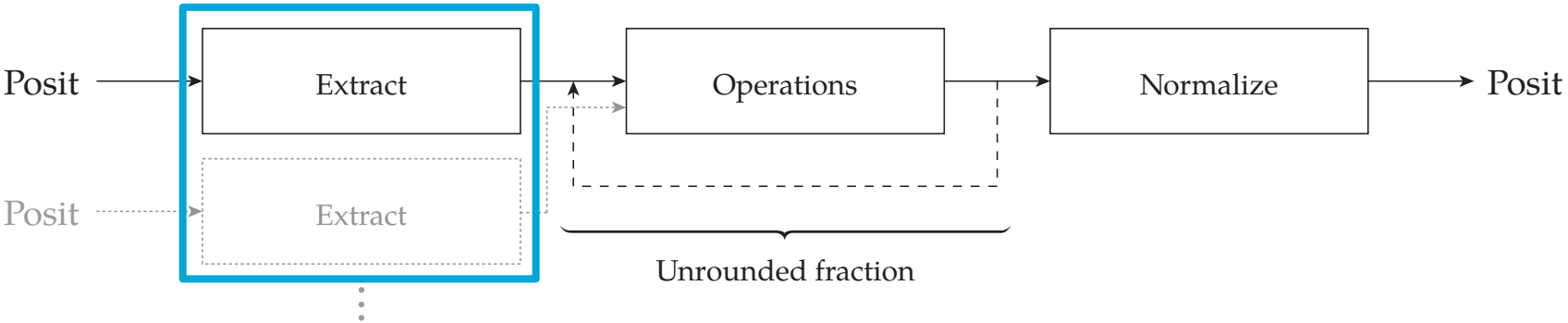
# Posit Arithmetic in Hardware

- Three stages



# Posit Arithmetic in Hardware

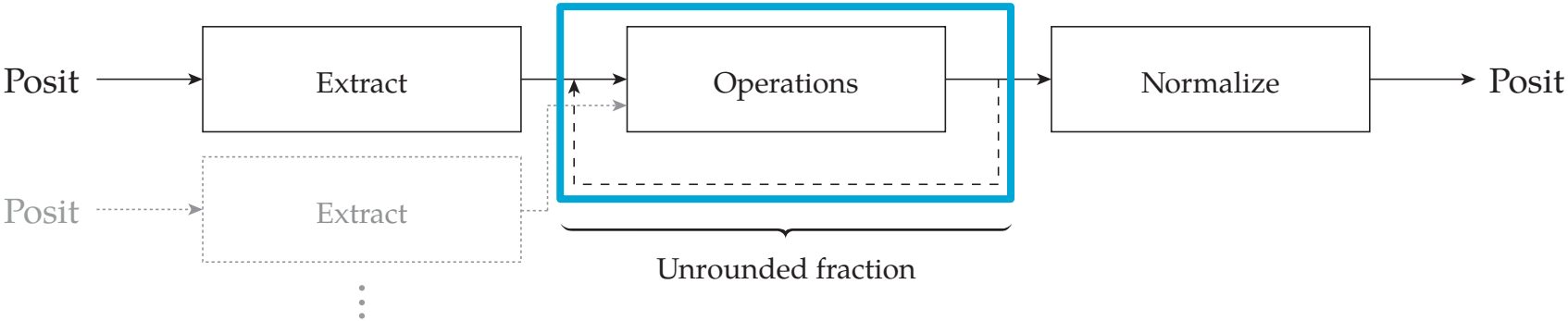
- Three stages




1. **Extraction**: extract fields from posit words

# Posit Arithmetic in Hardware

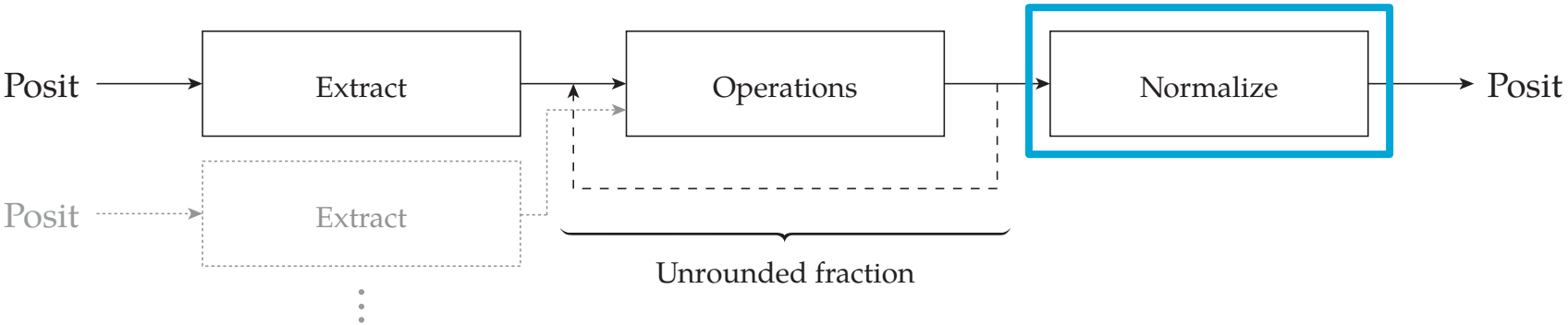
- Three stages




1. **Extraction**: extract fields from posit words
2. **Operation**: perform calculation(s) on extracted posit(s)
    - *Without intermediate rounding*: minimize error

# Posit Arithmetic in Hardware

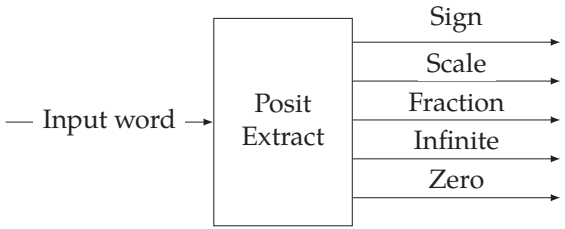
- Three stages



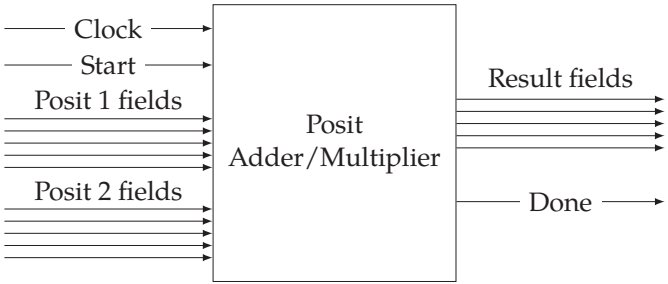

1. **Extraction**: extract fields from posit words
2. **Operation**: perform calculation(s) on extracted posit(s)
    - Without intermediate rounding: minimize error
3. **Normalization**: pack result components into posit word

# Extractor

- Obtain all fields
  - Detect size of regime (*kbits*)
  - Shift fields accordingly
- Signal two special cases:
  - Infinite
  - Zero

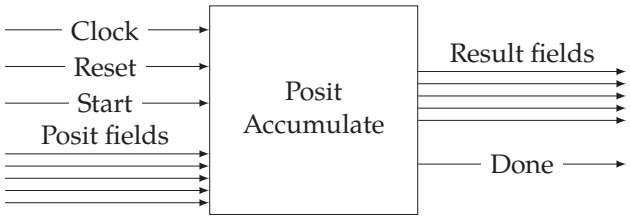
# Adder / multiplier

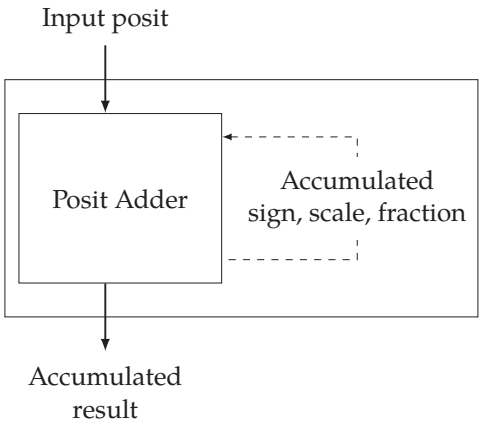



- 4- or 8-stage pipelined adder.
- 4-stage pipelined multiplier.

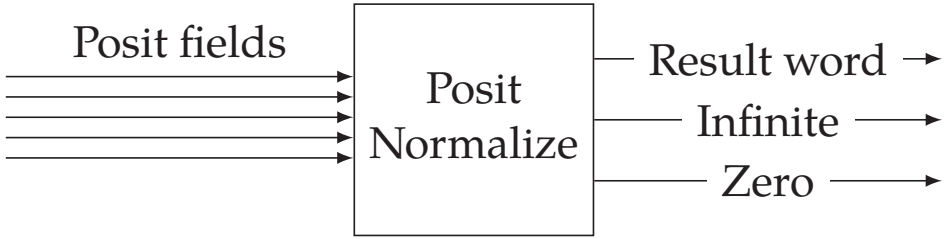
# Accumulator

- Accumulate input without rounding



# Normalizer




- Rounding scheme: round-to-nearest, tie-to-even.

# Outline

- Posit Arithmetic in Hardware
- **Posit Vector Arithmetic Accelerator**
- Pair Hidden Markov Model Accelerator
- Conclusion

# Posit Vector Arithmetic Accelerator

- Accelerate Level 1 Basic Linear Algebra Subprograms (BLAS)
  - Vector addition, subtraction, multiplication
  - Dot product
  - Vector sum
- Optimized for
  - Performance
  - Decimal accuracy
- Uses Apache Arrow data format for cross-language support
  - Arrow interfaces exist for many languages:
    C, C++, C#, Go, Java, JavaScript, MATLAB, Python, R, Ruby, Rust

# Schematic overview

One or two input vectors
(represented as Arrow Arrays)

| Input 1 | Input 2 |
|---|---|
| posit | posit |
| posit | posit |
| ⋮ | ⋮ |
| posit | posit |
| posit | posit |

Either one output vector
(elementwise addition / multiplication)

Or one posit
(dot product, element sum)

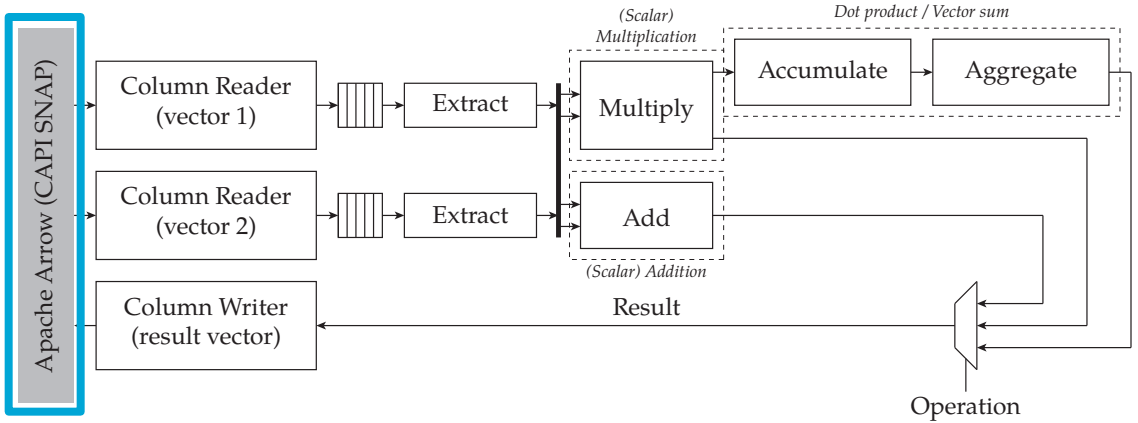
| Result |
|---|
| posit |
| posit |
| ⋮ |
| posit |
| posit |

*Addition, Multiplication*

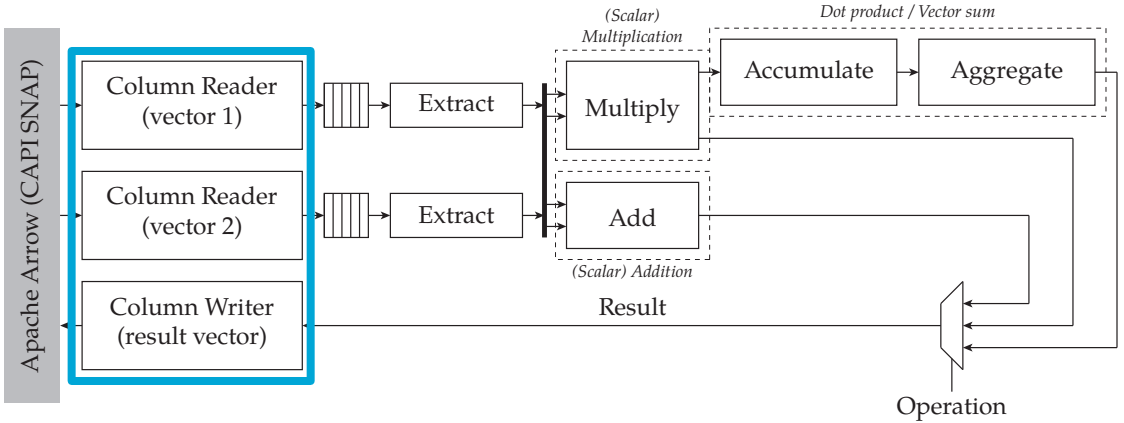
posit

*Dot product, Sum*

# Hardware Architecture for Vector Arithmetic

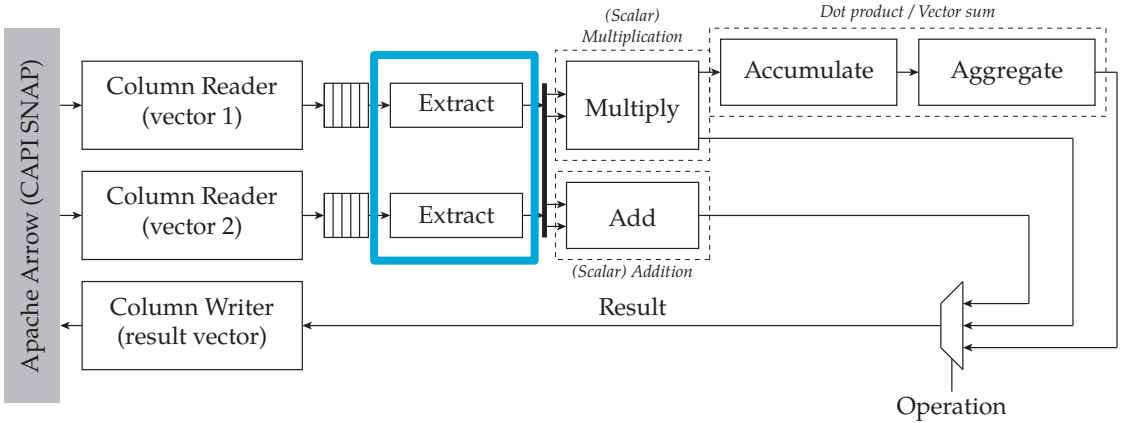


# Hardware Architecture for Vector Arithmetic

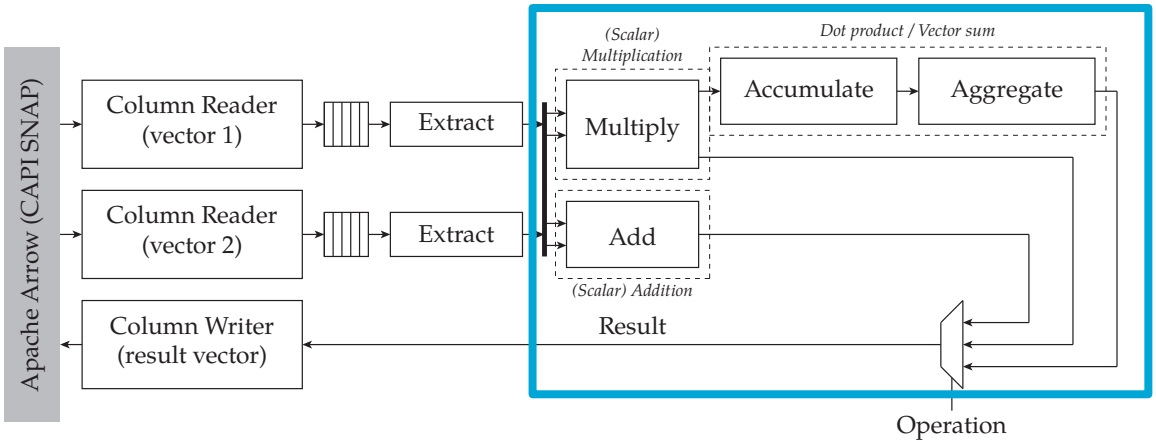



[3] *Fletcher: A framework to integrate FPGA accelerators with Apache Arrow. Online :* https://github.com/johanpel/fletcher

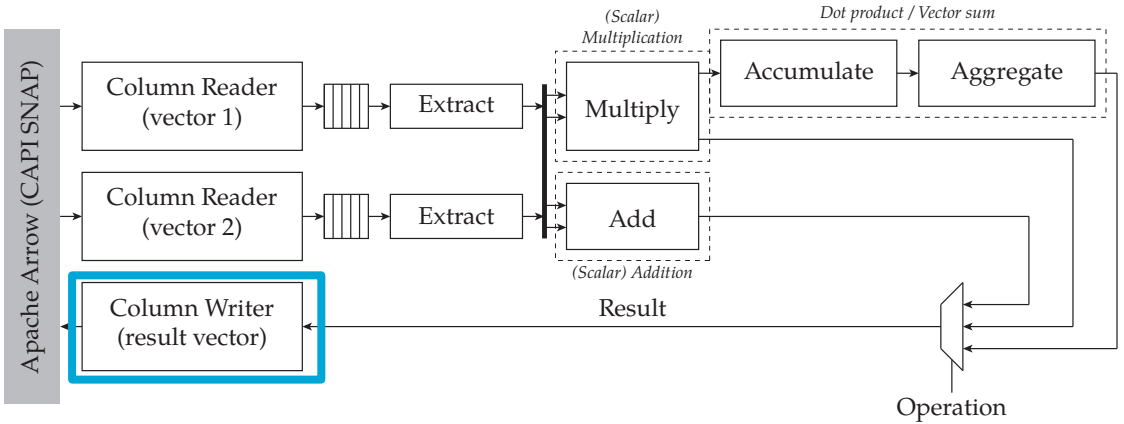
# Hardware Architecture for Vector Arithmetic



# Hardware Architecture for Vector Arithmetic



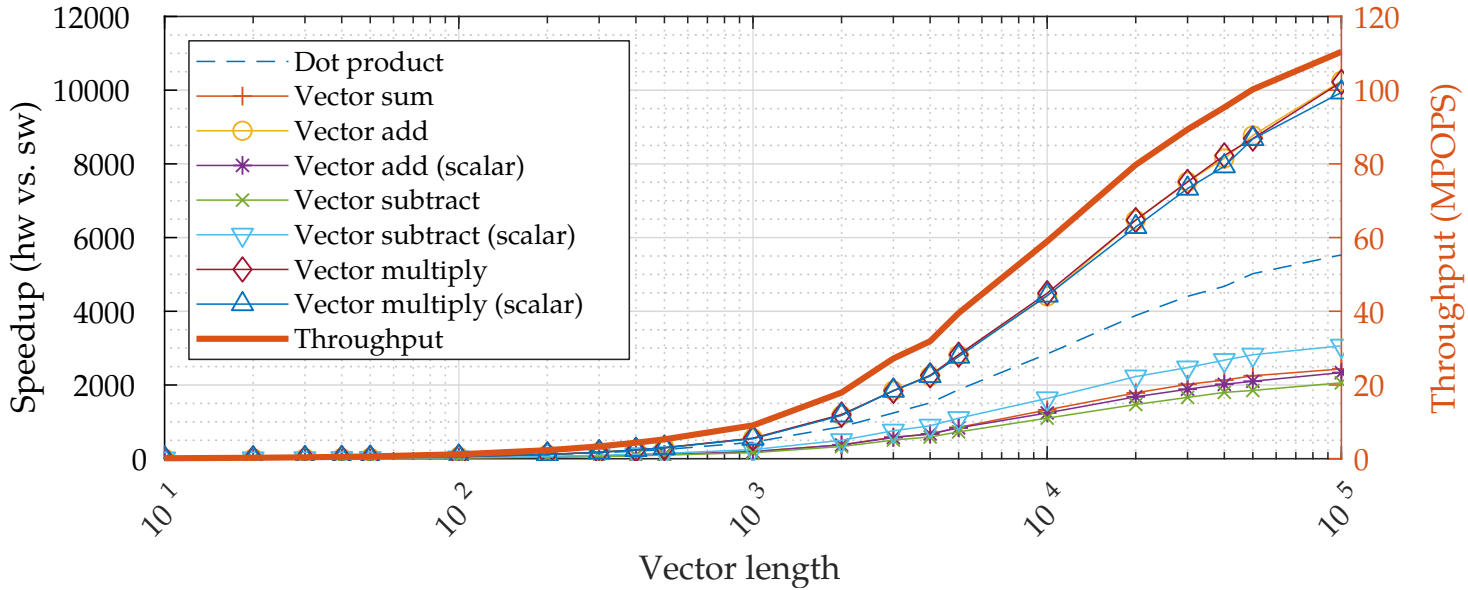
# Hardware Architecture for Vector Arithmetic




[3] *Fletcher: A framework to integrate FPGA accelerators with Apache Arrow.* J. Peltenburg et al., Online: https://github.com/johanpel/fletcher

# Vector Arithmetic Accelerator Performance

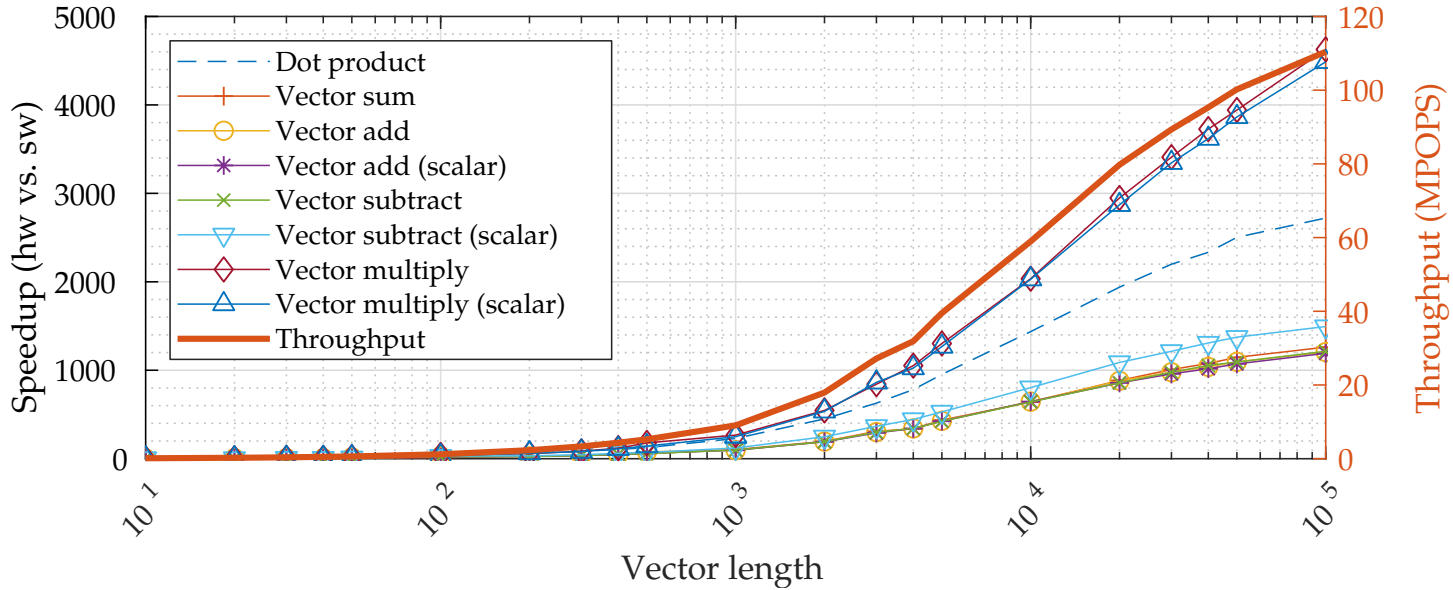
Speedup and throughput in MPOPS*




(*) MPOPS = Mega Posit Operations Per Second

# Vector Arithmetic Accelerator Performance

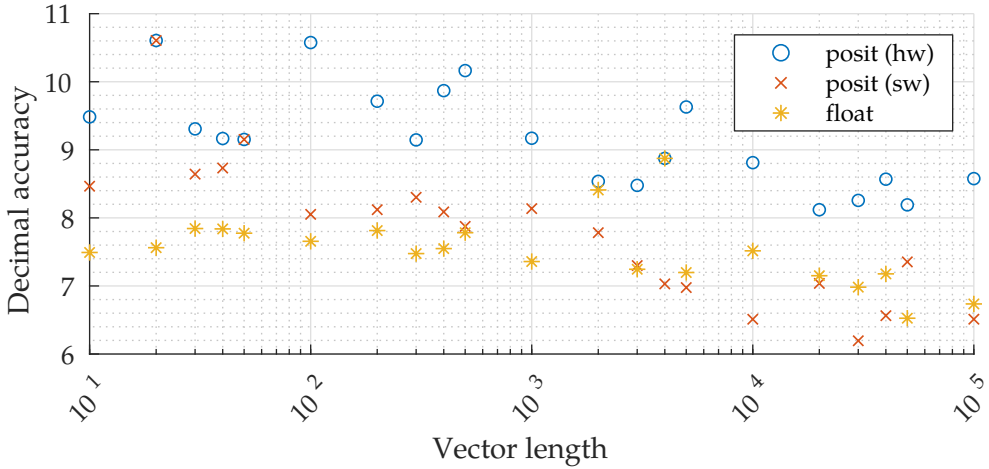
Speedup and throughput in MPOPS* (**BLAS library**)




(*) MPOPS = Mega Posit Operations Per Second

# Vector Arithmetic Accelerator Accuracy

- Increase in accuracy when using *posit* instead of *float*?
- Test case: pseudo randomly generated elements
  - Float and posit representation might not be of equal accuracy!
  - Re-generate until float and posit representation match
- Software *posit* and *float* calculation
  - C++ Matrix Template Library (MTL)




Decimal accuracy of dot product for *posit<nbits=32, es=2>*

*(similar results for <nbits=32, es=3)*

# Vector Arithmetic Accelerator Summary

- Modular design
  - Can be almost drop-in replace float units in existing applications.
- High speedup compared to software
  - Dependent on input vector lengths
  - Dot product: ~3000x for input length of $10^5$ elements
- Improved decimal accuracy
  - Dot product:
    - Over *float* : +2 decimals of accuracy
    - Over *posit* software : +1 decimal of accuracy

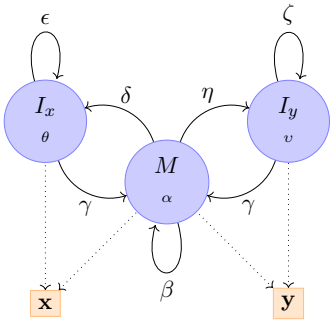
# Outline

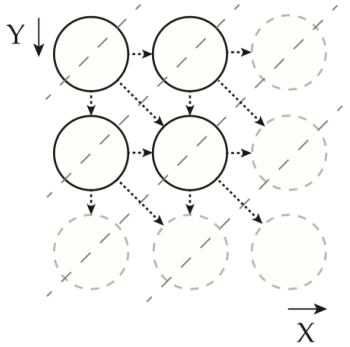
- Introduction
- Posit Arithmetic in Hardware
- Posit Vector Arithmetic Accelerator
- **Pair Hidden Markov Model Accelerator**
- Conclusion

# Pair-HMM

- Used in genomics analysis platforms (e.g. GATK)
- PairHMM Forward Algorithm
- Sparks our interest because:
  - Based on existing PairHMM accelerator [4].
  - Bioinformaticians value accuracy a lot.
  - E.g. in some basic implementations of PairHMM, calculations are done in float32 (to obtain higher throughput than float64)
  - If the output is in a float range that is inaccurate, *forward algorithm is performed again using float64*!

| $x$ | G | T | A | T | G | A | - | - |
|---|---|---|---|---|---|---|---|---|
| $y$ | - | - | A | T | G | A | T | A |
| $z$ | $I_x$ | $I_x$ | $M$ | $M$ | $M$ | $M$ | $I_y$ | $I_y$ |



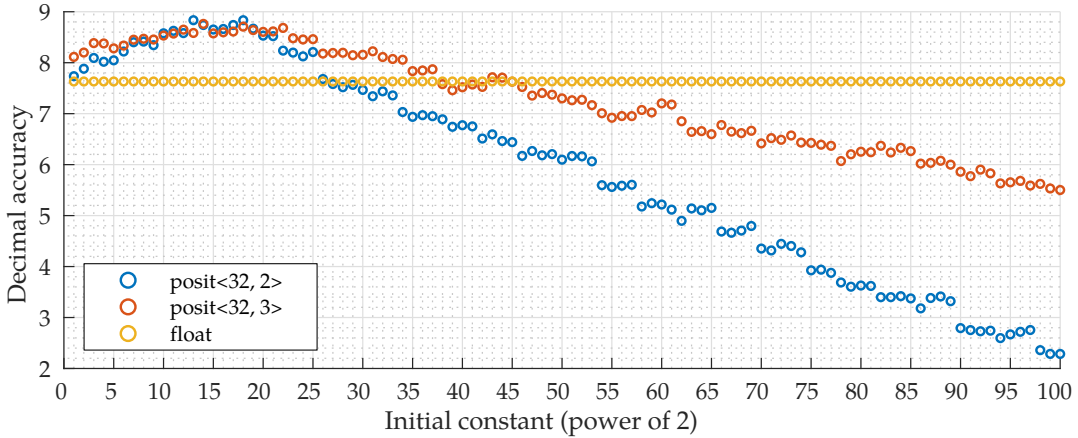




$$M(i,j) = \alpha_{i,j}\left[\beta_i M(i-1,j-1) + \gamma_i(I_x(i-1,j-1) + I_y(i-1,j-1))\right]$$
$$I_x(i,j) = \theta_i\left[\delta_i M(i-1,j) + \epsilon_i I_x(i-1,j)\right]$$
$$I_y(i,j) = \upsilon_j\left[\eta_i M(i,j-1) + \zeta_i I_y(i,j-1)\right]$$

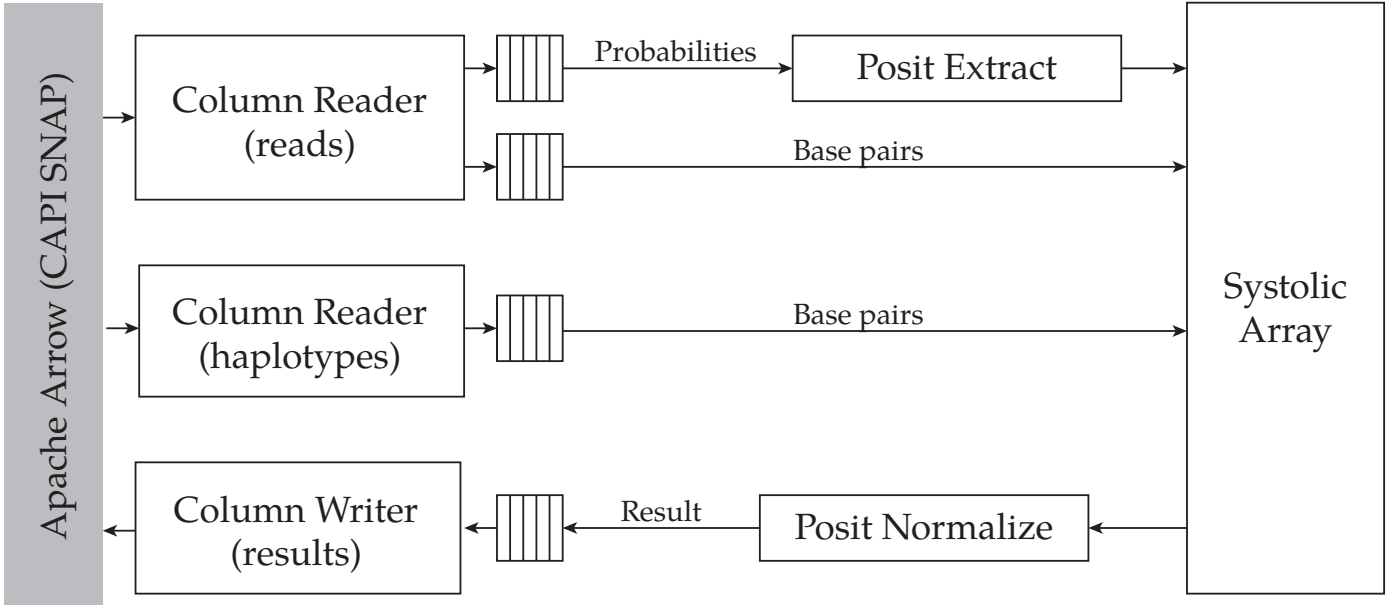

[4] J. Peltenburg, S. Ren, and Z. Al-Ars. 2016. Maximizing systolic array efciency to accelerate the PairHMM Forward Algorithm. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 758–762. https://doi.org/10.1109/BIBM.2016.7822616

# Pair-HMM in *float* and *posit*

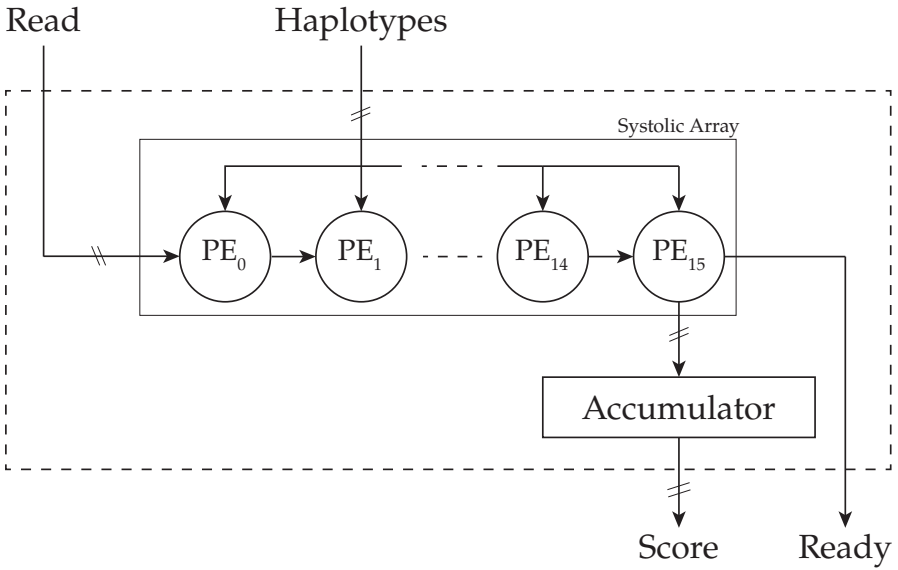
- Scaling of probabilities used to prevent underflow.
- For posits, accuracy depends on chosen initial scaling constant.
- There is a range where posits perform better than floats.



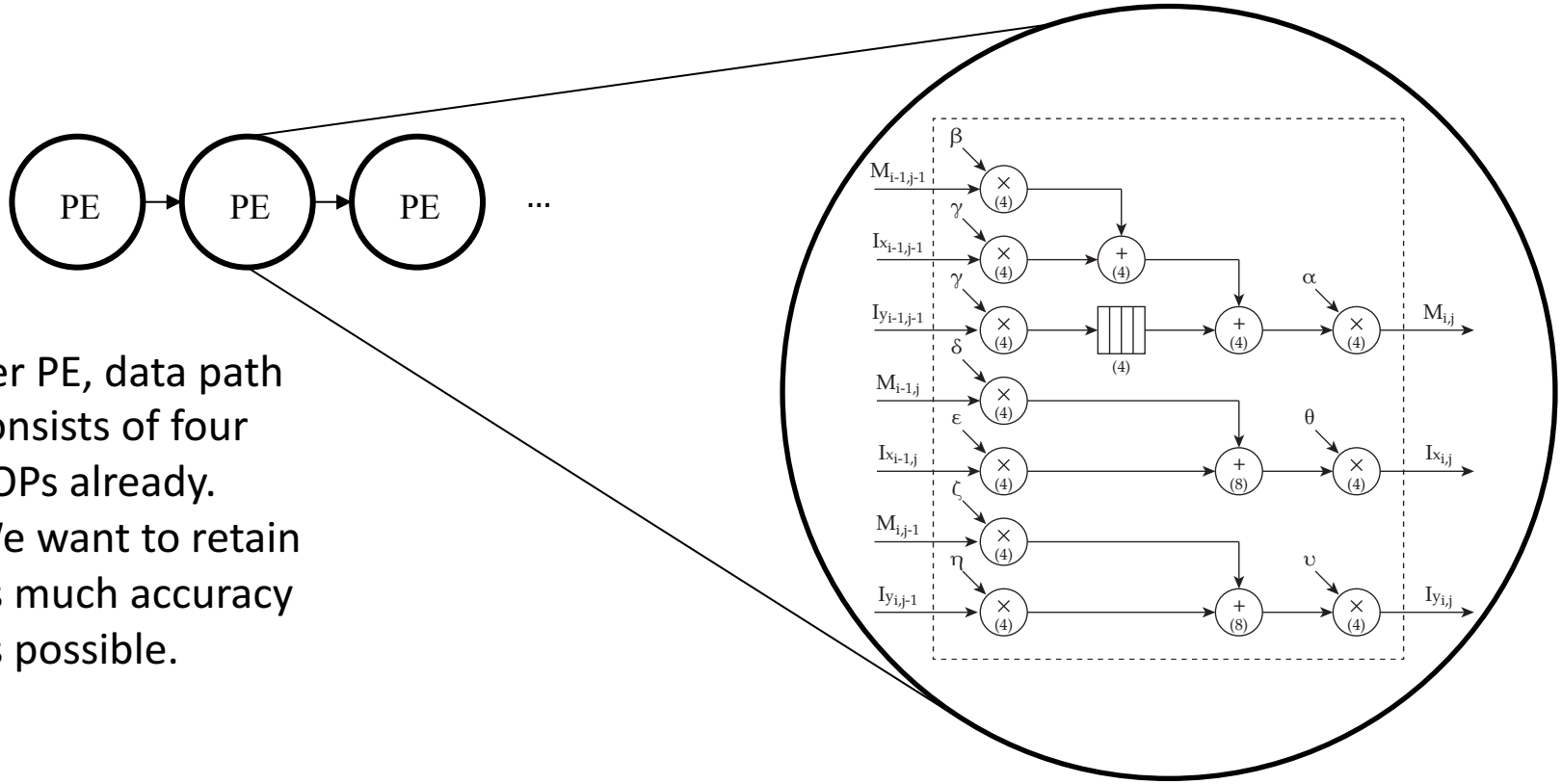
# Pair-HMM Top Level Architecture



# Pair-HMM Systolic Array Architecture

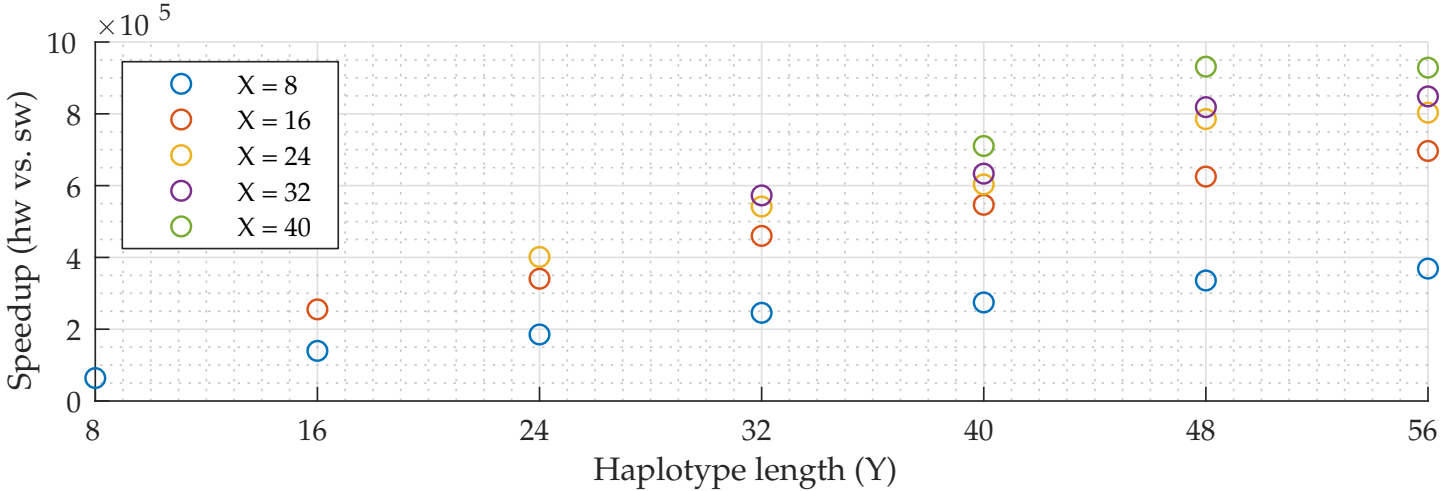


# Pair-HMM Posit Processing Element



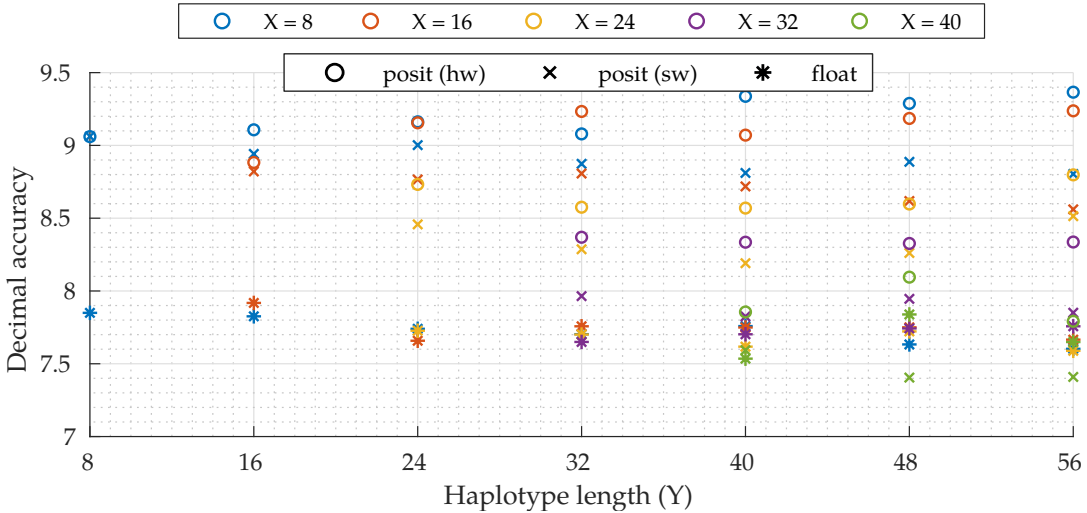

- Per PE, data path consists of four POPs already.
- We want to retain as much accuracy as possible.

# Pair-HMM Results - Performance



# Pair-HMM Results - Accuracy

Decimal accuracy of pair-HMM for *posit<32, 2>*



# Pair-HMM Summary

- Implementation of pair-HMM model accelerator with posits
- Performance
  - Speedup of ~$10^6\times$ for average input sequence lengths (X = 40, Y = 56)
- Decimal accuracy
  - Compared to software posit: +0.5 decimals of accuracy (on average)
  - Compared to software float: +0.75 decimals of accuracy

# Conclusion

- Design and implementation of pipelined **posit adders / multipliers targeting FPGA**.
- Can be used at **drop-in replacement for float units**.
- High-accuracy **accumulator**.
- Applied in two **accelerators**:
  - Vector arithmetic accelerator
  - PairHMM accelerator
  - Both increased accuracy
  - Several orders of magnitude faster than CPU
  - Need high POPS? No choice but to embrace FPGA accelerators!
- Integrates with Apache Arrow – **usable in many programming languages**.

# References


[1] L. van Dam, 2018, Enabling High Performance Posit Arithmetic Applications Using Hardware Acceleration, Online: http://resolver.tudelft.nl/uuid:943f302f-7667-4d88-b225-3cd0cd7cf37c

[2] John L. Gustafson, 2017, Posit Arithmetic. Online: https://posithub.org/docs/Posits4.pdf

[3] J. Peltenburg et al, 2018, Fletcher: A framework to integrate FPGA accelerators with Apache Arrow, Online: https://github.com/johanpel/fletcher

[4] J. Peltenburg, S. Ren, and Z. Al-Ars, 2016, Maximizing systolic array efciency to accelerate the PairHMM Forward Algorithm. In 2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). 758–762. https://doi.org/10.1109/BIBM.2016.7822616

# An Accelerator for Posit Arithmetic Targeting Posit Level 1 BLAS Routines and Pair-HMM

Laurens van Dam, Johan Peltenburg*, Zaid Al-Ars, H. Peter Hofstee

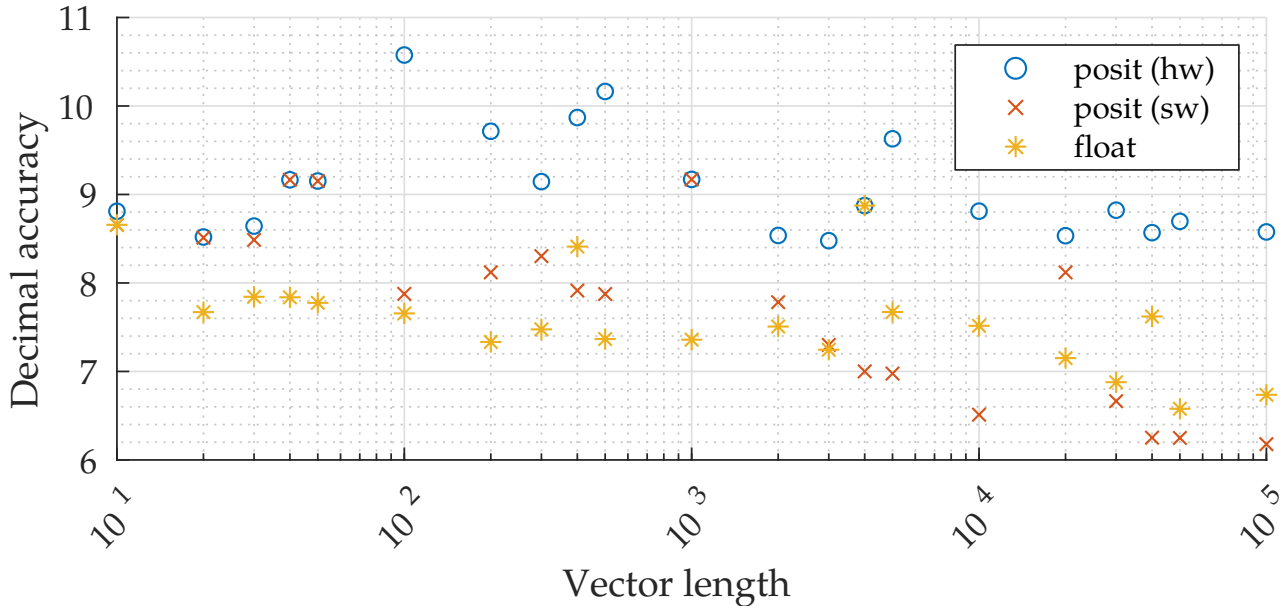
CoNGA'19: Conference for Next Generation Arithmetic 2019
Singapore, March 14, 2019

TU Delft

# Backup slides

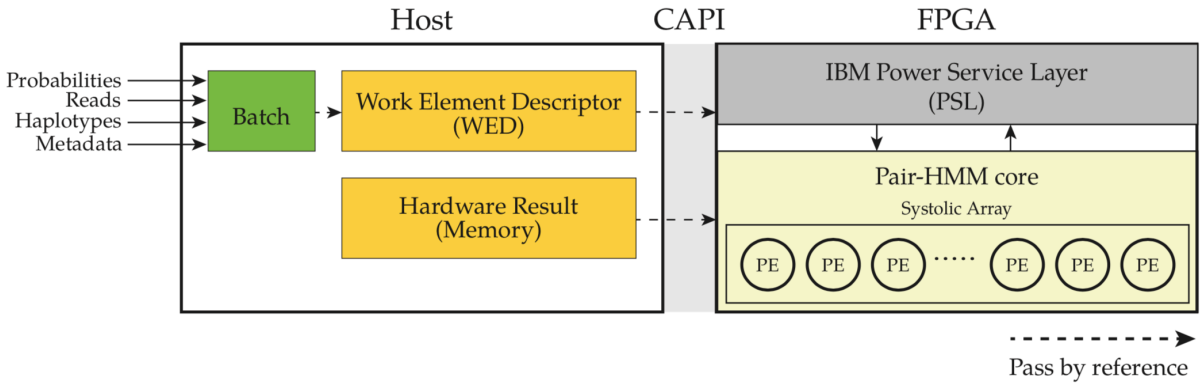
# Posit Vector Arithmetic Accelerator

Decimal accuracy of dot product for *posit<32, 3>*



# Pair-HMM Posit Accelerator

- CAPI: **C**oherent **A**ccelerator **P**rocessor **I**nterface

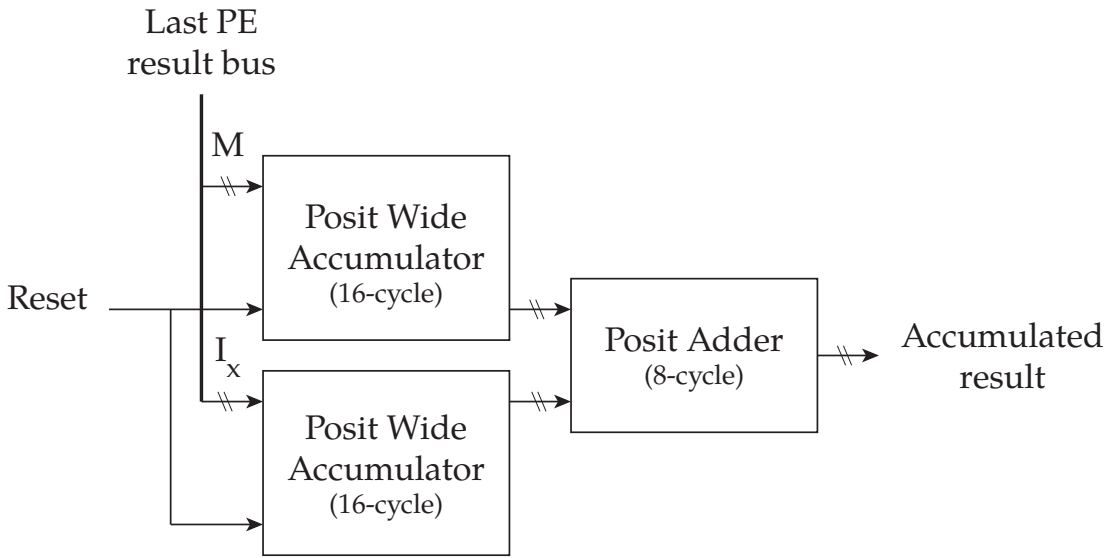


# Measure of Decimal Accuracy

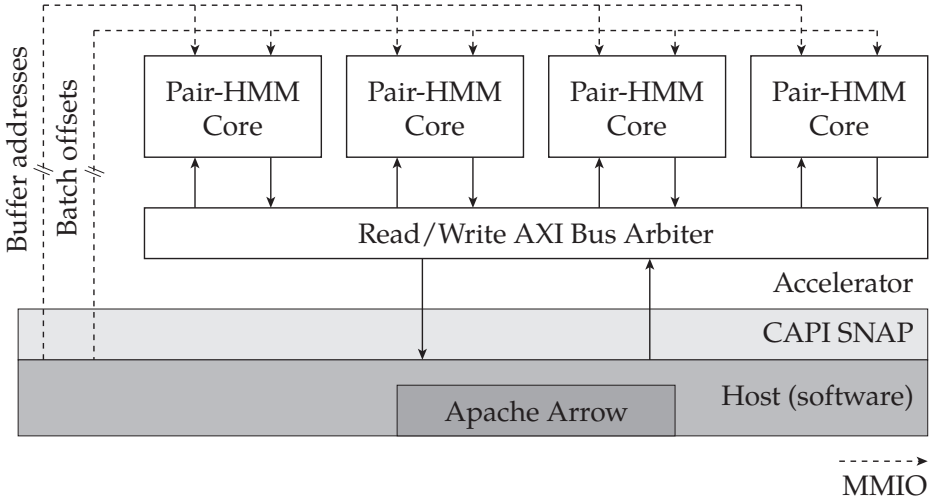
$$\text{decimal accuracy} = -\log_{10}\left|\log_{10}\left(\frac{\tilde{X}}{X}\right)\right|$$

$\tilde{X}$ = measured value
$X$ = true (reference) value

# Pair-HMM Posit Accelerator: final result



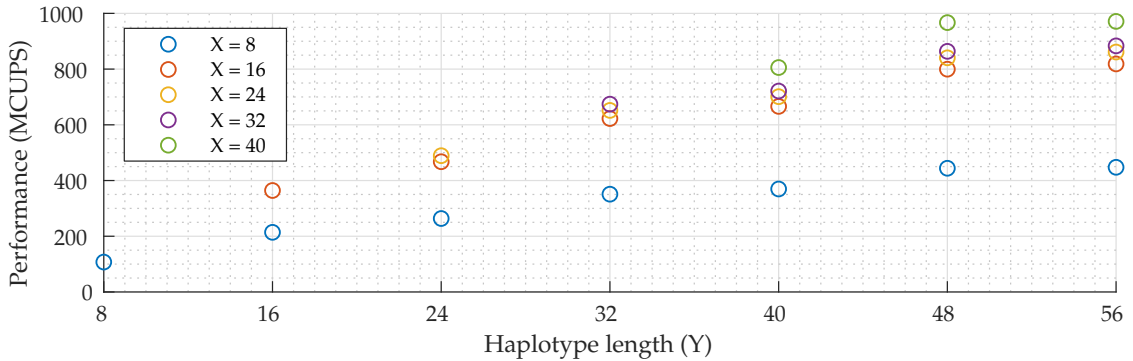
# Pair-HMM parallel architecture



# Pair-HMM Posit Accelerator

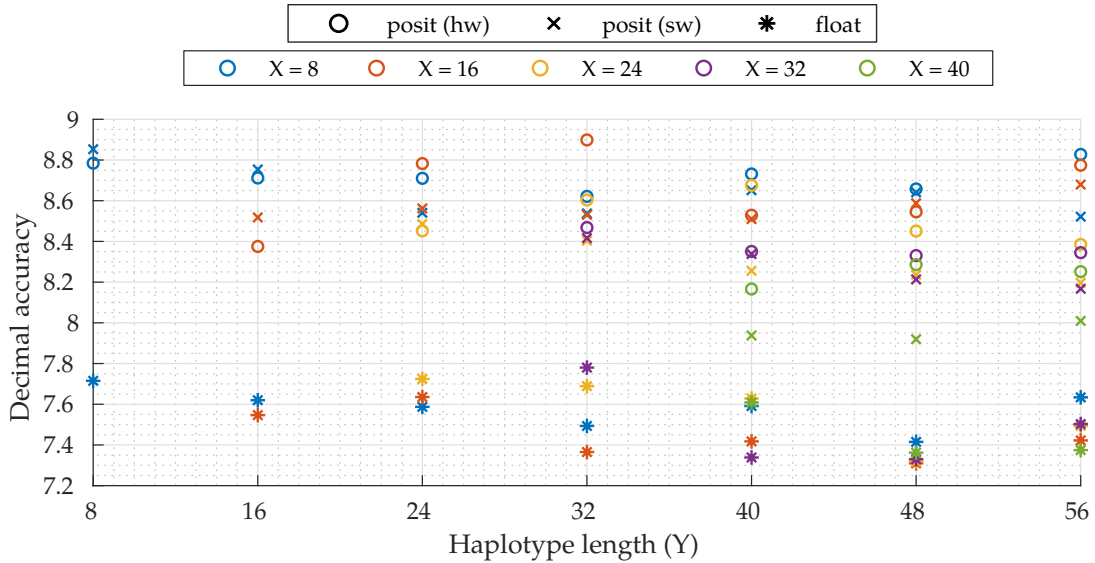
Throughput

- CUPS : **C**ell **U**pdates **P**er **S**econd



# Pair-HMM Posit Accelerator

Decimal accuracy of pair-HMM for *posit<32, 3>*



# Test setup

- Posit Vector Arithmetic Accelerator
  - Intel Core i5-2550K, 8 threads, 16 GB RAM
  - FPGA: Xilinx UltraScale XCKU060

- Pair-HMM Accelerator
  - IBM Power System S822LC, 2x 10-core CPU
  - Xilinx UltraScale XCKU060

# Vector area consumption & power estimation

Xilinx UltraScale XCKU060

| Configuration | | Available | Used (core) | | Used (total) | |
|---|---|---|---|---|---|---|
| posit<32,2> | **LUT** | 331680 | 52265 | (15.76%) | 131189 | (39.55%) |
| | **Register** | 663360 | 64969 | (8.64%) | 156747 | (23.63%) |
| | **BRAM** | 1080 | 91 | (9.79%) | 417 | (37.18%) |
| | **DSP** | 2760 | 4 | (0.14%) | 23 | (0.83%) |
| | **Power** | | 2.495 W | | 9.543 W | |
| posit<32,3> | **LUT** | 331680 | 52262 | (15.76%) | 131179 | (39.55%) |
| | **Register** | 663360 | 65033 | (8.29%) | 156827 | (23.64%) |
| | **BRAM** | 1080 | 91 | (8.43%) | 417 | (38.61%) |
| | **DSP** | 2760 | 4 | (0.14%) | 23 | (0.83%) |
| | **Power** | | 2.294 W | | 9.358 W | |

# PairHMM area consumption & power estimation

Xilinx UltraScale XCKU060

| Configuration | | Available | Used (core) | | Used (total) | |
|---|---|---|---|---|---|---|
| posit<32,2> | LUT | 331680 | 185174 | (55.83%) | 264078 | (79.62%) |
| | Register | 663360 | 179229 | (27.02%) | 271031 | (40.86%) |
| | BRAM | 1080 | 99 | (9.17%) | 425 | (39.35%) |
| | DSP | 2760 | 704 | (25.51%) | 723 | (26.20%) |
| | Power | | 18.299 W | | 25.379 W | |
| posit<32,3> | LUT | 331680 | 191827 | (57.83%) | 270820 | (81.65%) |
| | Register | 663360 | 186591 | (28.13%) | 278385 | (41.97%) |
| | BRAM | 1080 | 99 | (9.17%) | 425 | (39.35%) |
| | DSP | 2760 | 704 | (25.51%) | 723 | (26.20%) |
| | Power | | 17.412 W | | 24.479 W | |