TransPimLib:

Efficient Transcendental Functions for Processing-in-Memory Systems

Maurus Item, Juan Gómez Luna, Yuxin Guo,
Geraldo F. Oliveira, Mohammad Sadrosadati, Onur Mutlu

https://arxiv.org/pdf/2304.01951.pdf
https://github.com/CMU-SAFARI/transpimlib

ETH Zürich

SAFARI

Friday, December 22, 2023

Executive Summary

- Processing-in-Memory (PIM) promises to alleviate the *data movement bottleneck*
- However, current real-world PIM systems have very constrained hardware, which results in limited instruction sets
    - Difficulty/impossibility of computing complex operations, such as transcendental functions (e.g., trigonometric, exp, log) and other hard-to-calculate functions (e.g., square root)
    - These functions are important for modern workloads, e.g., activation functions in machine learning applications
- TransPimLib is the first library for transcendental and other hard-to-calculate functions on general-purpose PIM systems
    - CORDIC-based and LUT-based methods for trigonometric functions, hyperbolic functions, exponentiation, logarithm, square root, etc.
    - Source code: https://github.com/CMU-SAFARI/transpimlib
- We implement TransPimLib for the UPMEM PIM architecture and evaluate its methods in terms of performance, accuracy, memory requirements, and setup time
    - Three real workloads (Blackscholes, Sigmoid, Softmax)

Outline

Processing-in-memory
and transcendental functions

TransPimLib:
A library for transcendental
and other hard-to-calculate functions

Evaluation

Outline

**Processing-in-memory
and transcendental functions**

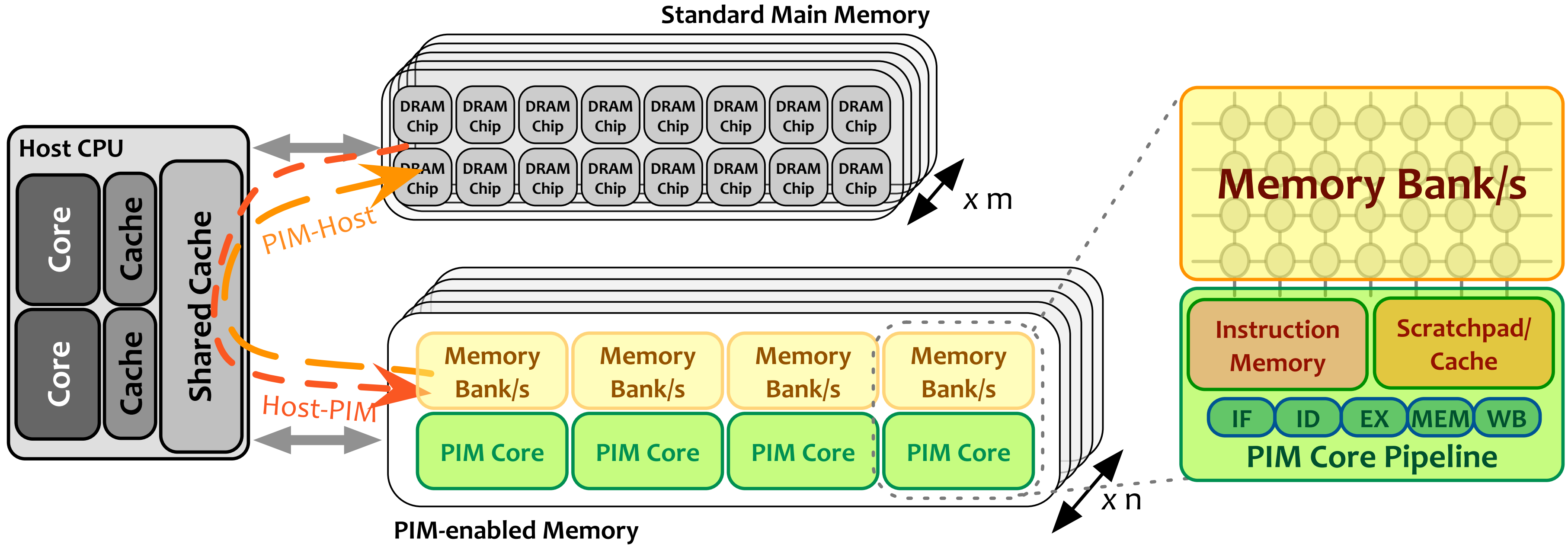
TransPimLib:
A library for transcendental
and other hard-to-calculate functions

Evaluation

Processing-in-Memory (PIM)

- PIM is a computing paradigm that advocates for memory-centric computing systems, where processing elements are placed near or inside the memory arrays
- Real-world PIM architectures are becoming a reality
  - UPMEM PIM, Samsung HBM-PIM, Samsung AxDIMM, SK Hynix AiM, Alibaba HB-PNM
- These PIM systems have some common characteristics:
  1. There is a host processor (CPU or GPU) with access to (1) standard main memory, and (2) PIM-enabled memory
  2. PIM-enabled memory contains multiple PIM processing elements (PEs) with high bandwidth and low latency memory access
  3. PIM PEs run only at a few hundred MHz and have a small number of registers and small (or no) cache/scratchpad
  4. PIM PEs may need to communicate via the host processor

A State-of-the-Art PIM System

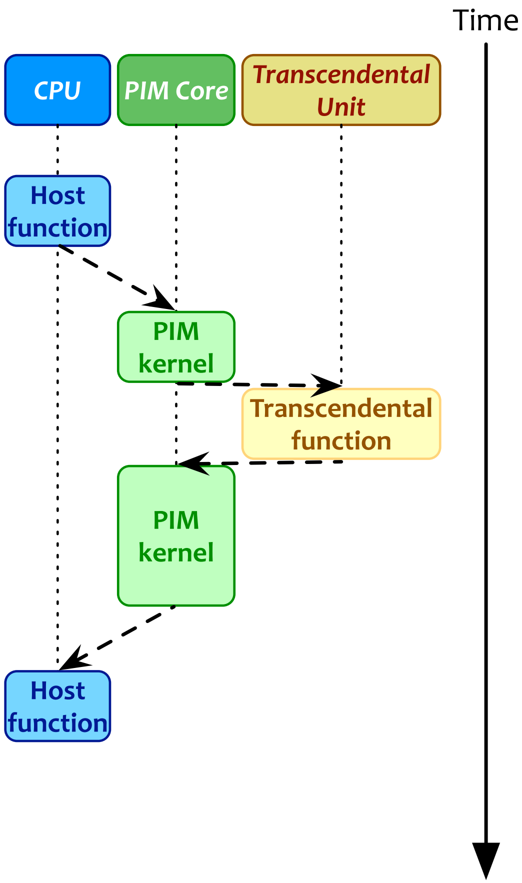


- In our work, we use the UPMEM PIM architecture
  - General-purpose processing cores called *DRAM Processing Units (DPUs)*
    - Up to 24 PIM threads, called *tasklets*
    - 32-bit integer arithmetic, but multiplication/division are emulated*, as well as floating-point operations
  - 64-MB DRAM bank (*MRAM*), 64-KB scratchpad (*WRAM*)

* 8-bit integer multiplication is natively supported

How to Calculate Transcendental Functions in a PIM System?

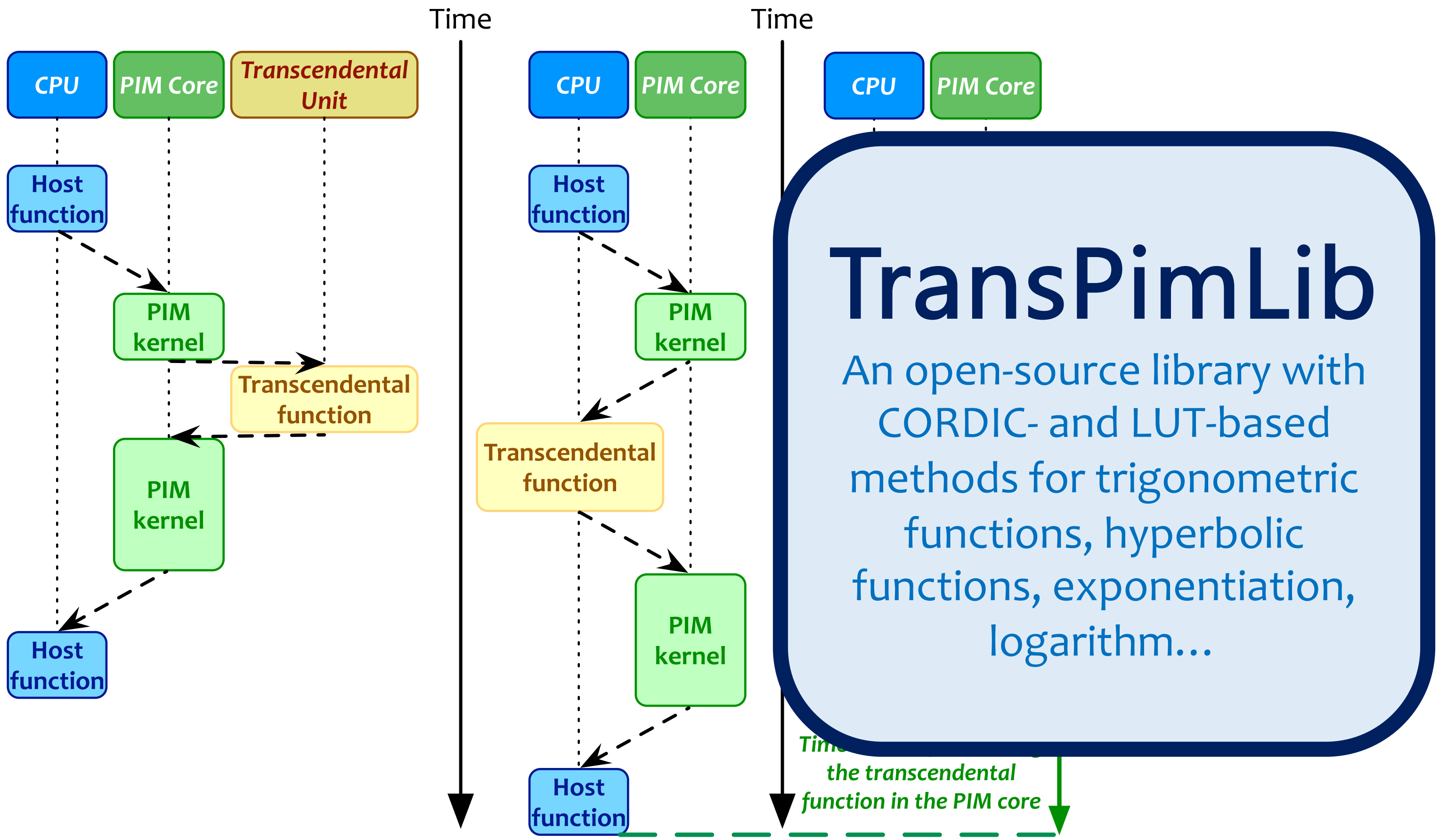
- Three possible alternatives



(a) Transcendental function executed in special PIM unit

How to Calculate Transcendental Functions in a PIM System?

- Three possible alternatives



(a) Transcendental function executed in special PIM unit

(b) Transcendental function executed in host CPU

(c) Transcendental function executed in PIM core

Outline

Processing-in-memory
and transcendental functions

TransPimLib:
A library for transcendental
and other hard-to-calculate functions

Evaluation

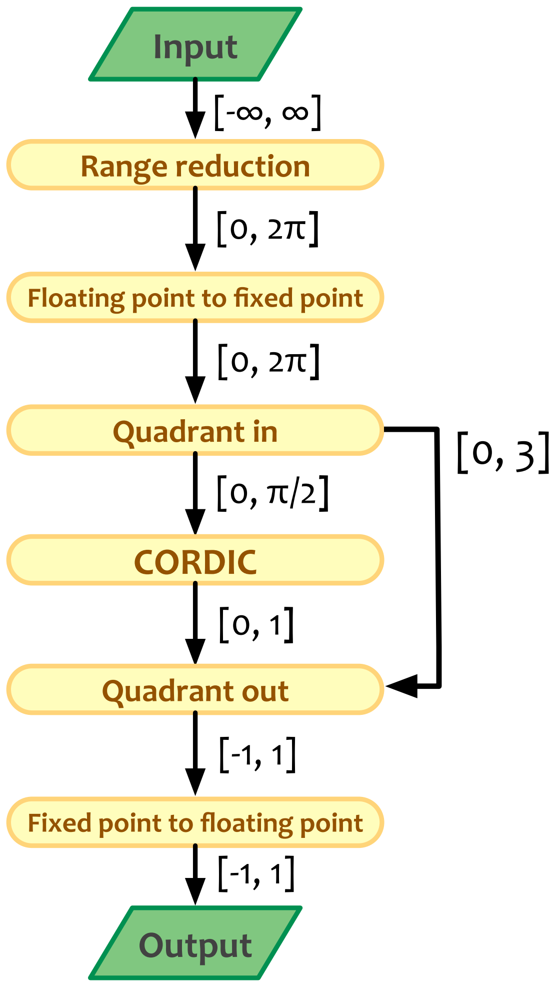
TransPimLib: Implementation

- Various methods to calculate transcendental functions:
  - Taylor approximation, minimax polynomials, CORDIC, LUTs
- CORDIC is an iterative method that uses bit-shifts, additions, and table lookups
  - In rotation mode, CORDIC computes the function value for an input θ by rotating a vector [1, 0] iteratively
  - The rotation is done by multiplying the vector and a matrix
  - The matrix represents the rotation angle, which decreases in each iteration
- Fuzzy Lookup Tables (LUTs) return an (approximate) output $f(x)$ for each input x
  - A function $a(x)$ returns an address to access the LUT
  - The table returns $LUT(a(x)) \simeq f(x)$
  - To generate the LUT, we need a helper function $a^{-1}()$, such that $x = a(a^{-1}(x))$
  - LUTs' accuracy improves with interpolation:

    $$f(x) \simeq LUT(a(x)) + LUT(a(x)+1) - LUT(a(x))\cdot\Delta$$

TransPimLib: CORDIC-based Methods

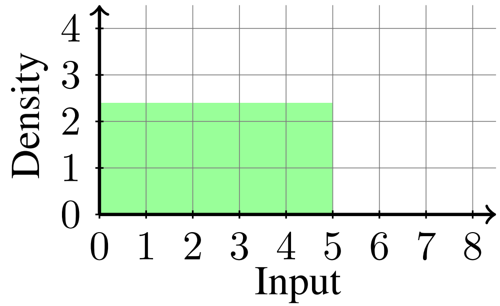
- TransPimLib contains CORDIC implementations of trigonometric (sin, cos, tan) and hyperbolic (sinh, cosh, tanh) functions, exponentiation, logarithm, and square root
- Example: Sine function

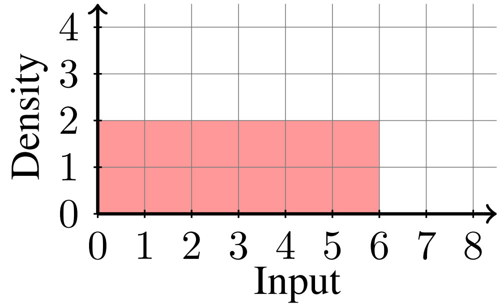


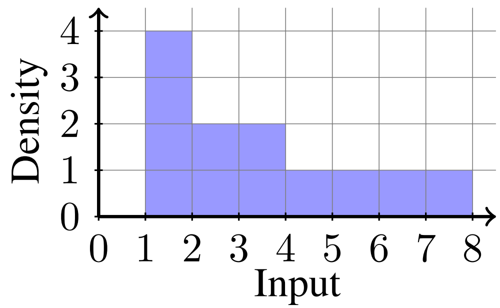
TransPimLib: LUT-based Methods

- Multiplication-based LUT (M-LUT)
  - Regular spacing between table entries
  - $a(x) = round((x - p) \cdot k)$, where k represents the LUT density
- LDEXP-based LUT (L-LUT)
  - Multiplication is cheaper if we multiply by 2^n
  - $ldexp(arg, exp)$ to perform $arg \cdot 2^{exp}$
  - $a(x) = round((x - p) \cdot 2^n)$
    - k is a power-of-two, which results in less precision but avoids multiplication
- Direct Float Conversion-based LUT (D-LUT)
  - $a(x)$ uses the last n bits of the exponent and p bits of the mantissa
  - Piece-wise linear density: 2^n steps of 2^p addresses

Map interval [0, 5] to a 12-entry LUT

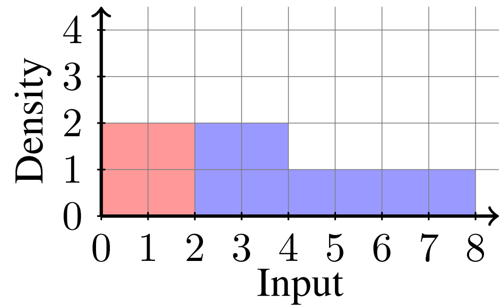


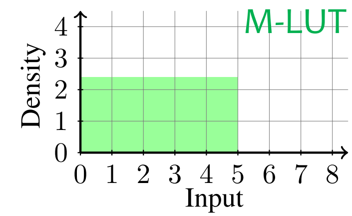


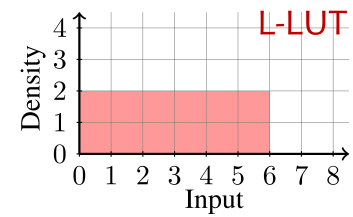


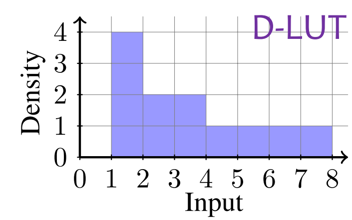
TransPimLib: Combined Methods

- Direct Float Conversion + LDEXP-based LUT (DL-LUT)
  - Uses an L-LUT between 0 and the smallest exponent and a D-LUT for larger inputs









- CORDIC+L-LUT (CORDIC+LUT)
  - Replaces the first few iterations of CORDIC with a LUT
  - Flexible tradeoff between computing cost, table size, and precision

TransPimLib: Supported Functions

	Supported Functions									
Implementation Method	**sin**	**cos**	**tan**	**sinh**	**cosh**	**tanh**	**exp**	**log**	**sqrt**	**GELU**
CORDIC	✓	✓	✓	✓	✓	✓	✓	✓	✓	
M-LUT	✓	✓	✓				✓	✓	✓	
M-LUT+Interp.	✓	✓	✓				✓	✓	✓	
L-LUT	✓	✓	✓				✓	✓	✓	
L-LUT+Interp.	✓	✓	✓				✓	✓	✓	
D-LUT+Interp.	✓					✓				✓
DL-LUT+Interp.	✓					✓				✓
CORDIC+LUT	✓	✓	✓	✓	✓	✓	✓			

Based on our preliminary analysis, we provide the most suitable methods for each of the supported functions (other than sine).

Outline

Processing-in-memory
and transcendental functions

TransPimLib:
A library for transcendental
and other hard-to-calculate functions

Evaluation

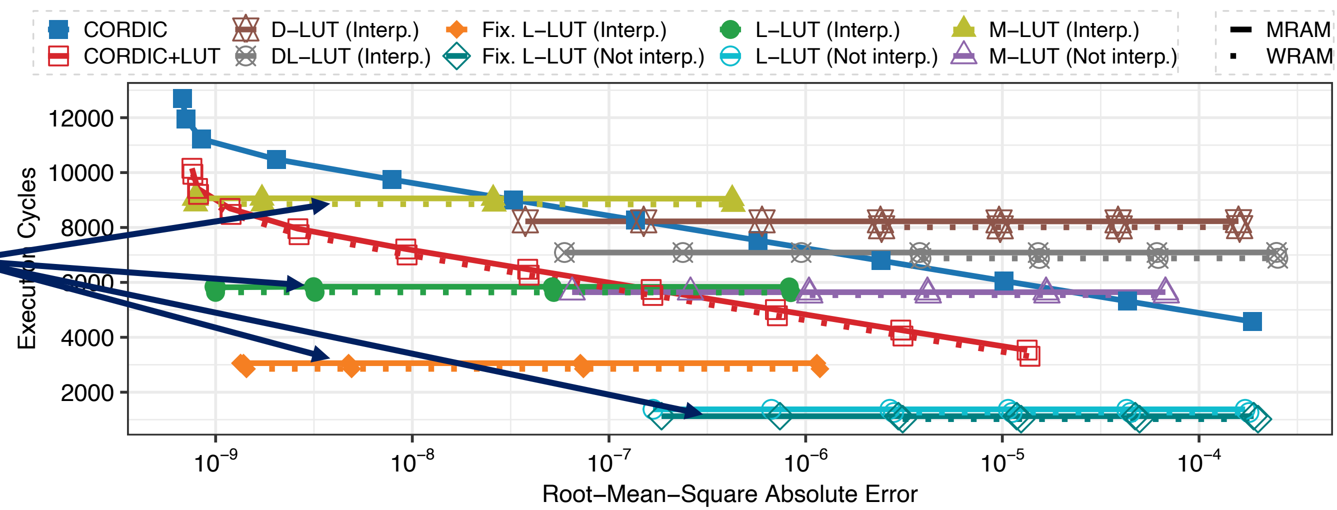
Evaluation Methodology

- Evaluated systems
  - UPMEM PIM system with 2,545 PIM cores @ 350 MHz and 159 GB of DRAM
  - 2-socket Intel Xeon CPU (32 cores)
- Microbenchmarks
  - Performance evaluation
    - We measure execution cycles
  - Accuracy evaluation
    - Root-mean-square absolute error (RMSE) with respect to the CPU with the standard math library
  - Setup time
    - Generation on the host CPU and transfers to the PIM side
  - Memory consumption
    - All tables and variables allocated in the DRAM bank of a PIM core
  - We use sine, as a representative function
- Real-world Benchmarks
  - Blackscholes: exp, log, sqrt, cumulative normal distribution (CNDF)
  - Sigmoid
  - Softmax

Microbenchmark Results: Performance (I)

- We measure the execution cycles for an accuracy range between 10^{-4} and 10^{-9}
- LUT-based versions place the LUT in either the PIM core's DRAM bank (MRAM) or the scratchpad (WRAM)

Performance of LUT-based methods is independent of the accuracy



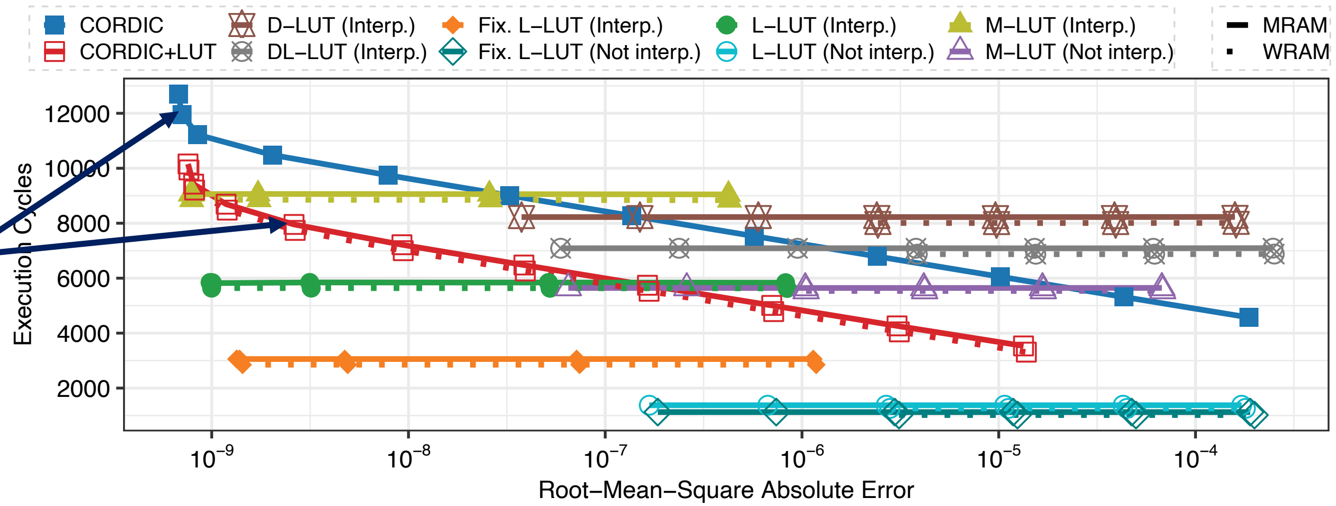
Execution cycles depend on the number of multiplications:
- Interp. M-LUT: 2 FP multiplications
- Non-interp. M-LUT and inter. L-LUT: 1 FP multiplication
- Non-interp. L-LUT: No FP multiplication

Fixed-point version of the L-LUT
- Interp. Fix. L-LUT doubles the performance of inter. L-LUT due to faster fixed-point multiplication

Microbenchmark Results: Performance (II)

- We measure the execution cycles for an accuracy range between 10^{-4} and 10^{-9}
- CORDIC-based methods take more execution cycles to provide higher accuracy

CORDIC accuracy increases with each iteration of the CORDIC algorithm



CORDIC+LUT runs faster than **CORDIC**, as it replaces the initial iterations with an L-LUT query

At some point (~10^{-9}), further increasing the LUT size or CORDIC iterations does not improve accuracy

Little benefit from placing LUTs in the scratchpad (WRAM) instead of the DRAM bank (MRAM)

Microbenchmark Results: Performance (III)

- We measure the execution cycles for an accuracy range between 10^{-4} and 10^{-9}
- CORDIC-based methods take more execution cycles to provide

Key Takeaway 1

Interpolated L-LUT methods (lookup table with LDEXP operation) offer the **best tradeoff** in terms of performance and accuracy

CORDIC+LUT runs faster than CORDIC, as it replaces the initial iterations with an L-LUT query

At some point (~10^{-9}), further increasing the LUT size or CORDIC iterations does not improve accuracy

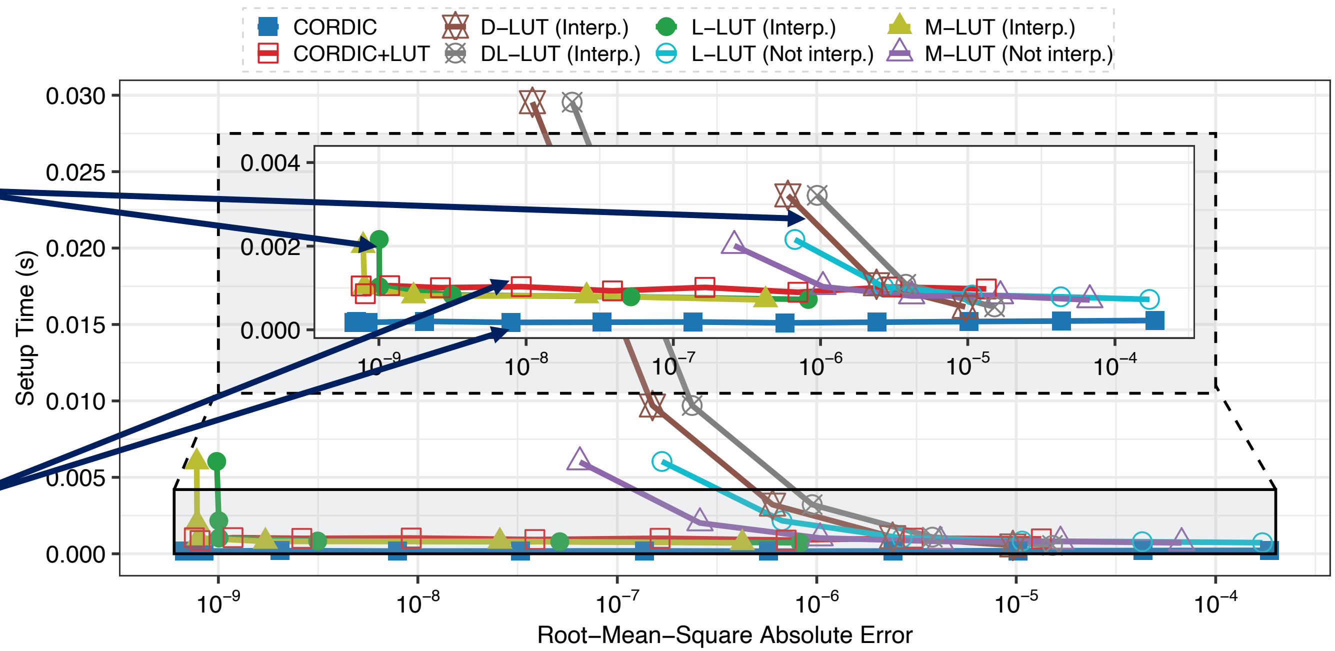
Little benefit from placing LUTs in the scratchpad (WRAM) instead of the DRAM bank (MRAM)

Microbenchmark Results: Setup Time (I)

- The setup time can also impact the decision of what method to use

For LUT-based methods, setup times increase with LUT size

CORDIC methods have flat setup times



CORDIC methods can provide higher overall performance (i.e., setup time + PIM kernel time) than LUT-based methods when the total number of transcendental functions in a workload is low.
For example, we estimate ~40 sine operations (see paper)

Microbenchmark Results: Setup Time (II)

- The setup time can also impact the decision of what method to use

Key Takeaway 2

CORDIC-based methods are preferable when a PIM kernel needs to execute **just a few transcendental functions** due to their low setup time in the host CPU

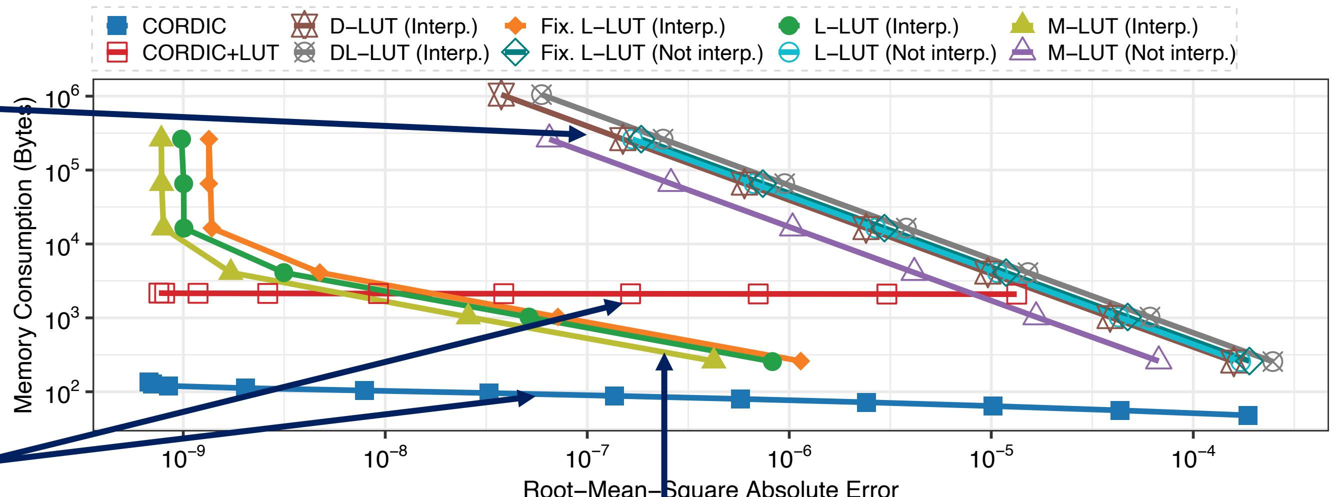
CORDIC methods can provide higher overall performance (i.e., setup time + PIM kernel time) than LUT-based methods when the total number of transcendental functions in a workload is low. For example, we estimate ~40 sine operations (see paper)

Microbenchmark Results: Memory (I)

- We also obtain the memory consumption (in bytes) in the DRAM bank of a PIM core

Accuracy of non-interp. LUT methods is limited by the available memory

Memory consumption of CORDIC methods does *not* increase exponentially



Interpolation is an effective way of increasing accuracy without increasing LUT size

Microbenchmark Results: Memory (II)

- We also obtain the memory consumption (in bytes) in

Key Takeaway 3

Interpolated L-LUT methods offer a good tradeoff in terms of accuracy, execution cycles, and memory consumption.

However, **CORDIC and CORDIC+LUT methods** are recommended for applications that require high accuracy, where the available memory is limited (e.g., needed for large datasets)

More in the Paper

- Analysis of other supported functions
- Evaluation of range reduction/extension
- Discussion and takeaway about D-LUT and DL-LUT methods

TransPimLib: Efficient Transcendental Functions for Processing-in-Memory Systems

Maurus Item Juan Gómez-Luna Yuxin Guo
Geraldo F. Oliveira Mohammad Sadrosadati Onur Mutlu
ETH Zürich

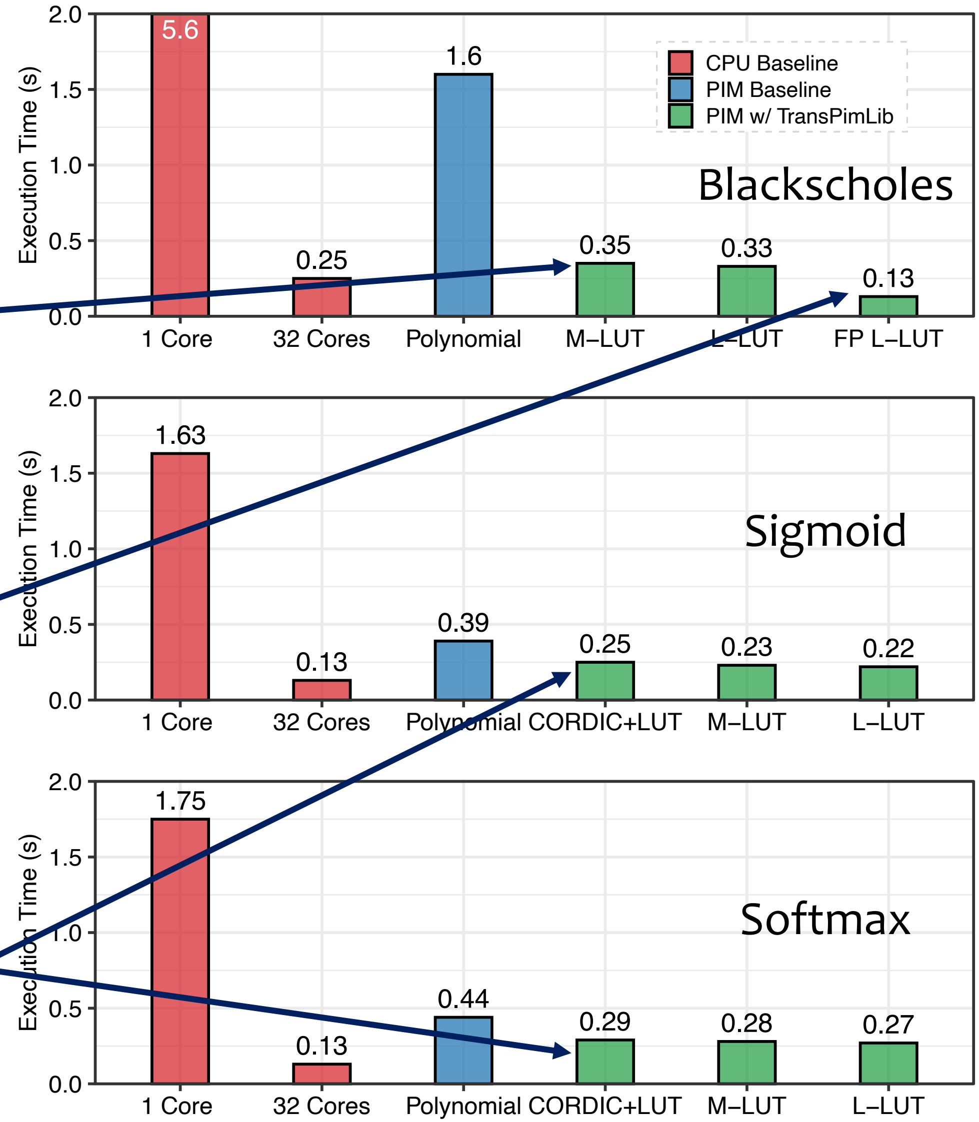
Real-world Benchmark Results (I)

- 1 & 32 CPU cores
- PIM baseline: Polynomial

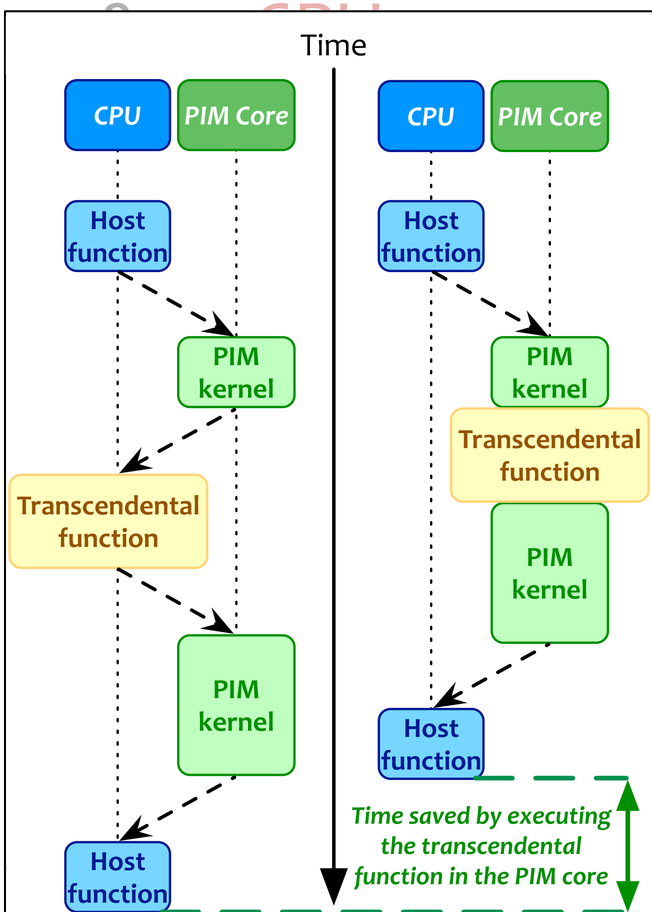
For Blackscholes, TransPimLib is 5-12x faster than the PIM baseline

Fixed-point L-LUT is 92% faster than the 32-thread CPU baseline

For Sigmoid and Softmax, TransPimLib outperforms the PIM baseline and shows that it can save data movement from executing activation functions in the host CPU



Real-world Benchmark Results (II)



(b) Transcendental function executed in host CPU

(c) Transcendental function executed in PIM core

Key Takeaway 4

TransPimLib can **reduce data movement from PIM cores to the CPU** (Fig. (b)) for applications running on the PIM cores.

As a result, the execution of transcendental functions in the PIM cores (Fig. (c)) could be 6–8× faster than the execution in the host CPU.

TransPimLib: arXiv Version

TransPimLib: A Library for Efficient Transcendental Functions on Processing-in-Memory Systems

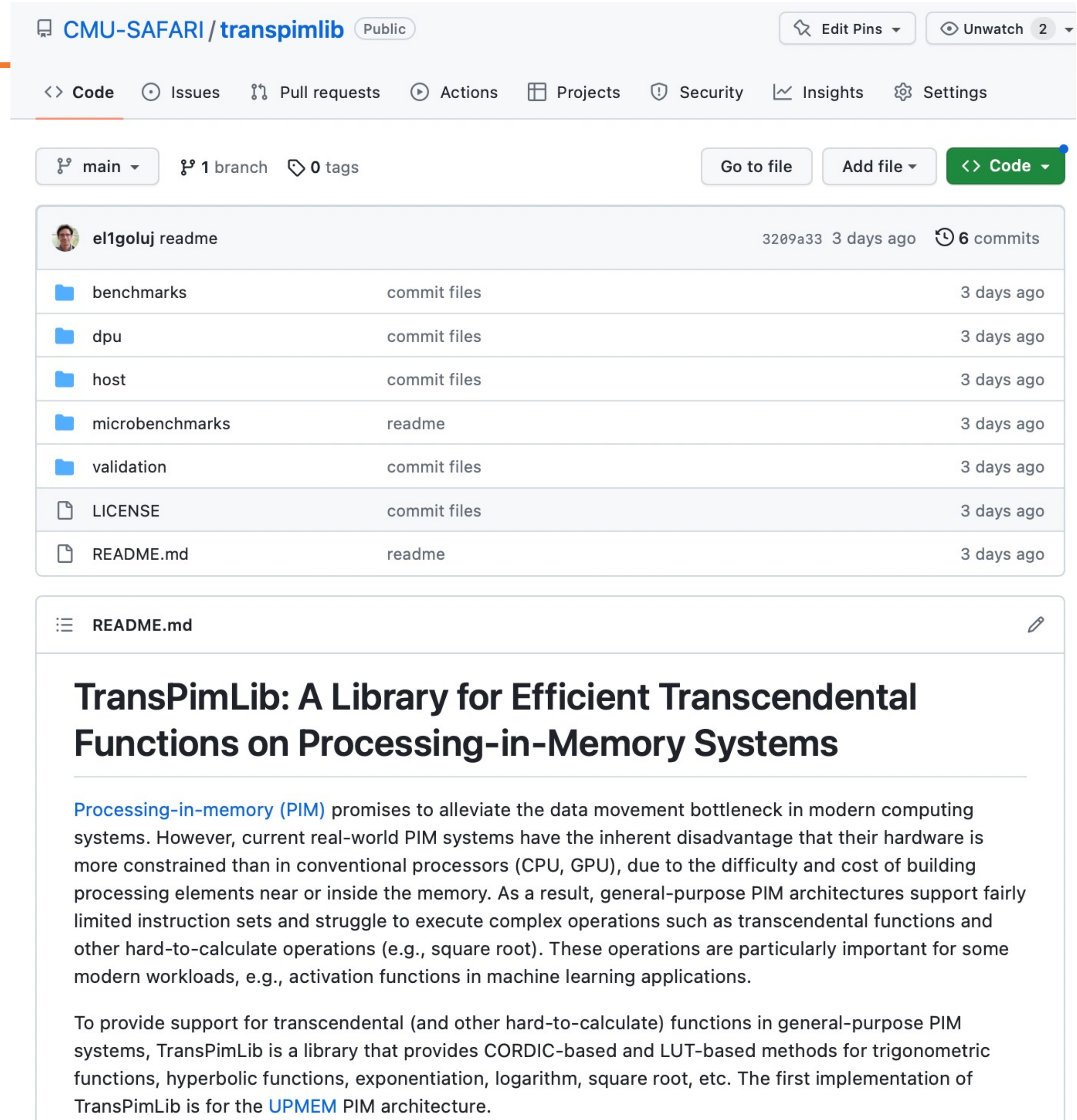
Maurus Item   Juan Gómez-Luna   Yuxin Guo
Geraldo F. Oliveira   Mohammad Sadrosadati   Onur Mutlu

ETH Zürich

https://arxiv.org/pdf/2304.01951.pdf

Source Code

- https://github.com/CMU-SAFARI/transpimlib



Conclusion

- Processing-in-Memory (PIM) promises to alleviate the *data movement bottleneck*
- However, current real-world PIM systems have very constrained hardware, which results in limited instruction sets
  - Difficulty/impossibility of computing complex operations, such as transcendental functions (e.g., trigonometric, exp, log) and other hard-to-calculate functions (e.g., square root)
  - These functions are important for modern workloads, e.g., activation functions in machine learning applications
- TransPimLib is the first library for transcendental and other hard-to-calculate functions on general-purpose PIM systems
  - CORDIC-based and LUT-based methods for trigonometric functions, hyperbolic functions, exponentiation, logarithm, square root, etc.
  - Source code: https://github.com/CMU-SAFARI/transpimlib
- We implement TransPimLib for the UPMEM PIM architecture and evaluate its methods in terms of performance, accuracy, memory requirements, and setup time
  - Three real workloads (Blackscholes, Sigmoid, Softmax)

TransPimLib:
Efficient Transcendental Functions for Processing-in-Memory Systems

Maurus Item, Juan Gómez Luna, Yuxin Guo,
Geraldo F. Oliveira, Mohammad Sadrosadati, Onur Mutlu

https://arxiv.org/pdf/2304.01951.pdf
https://github.com/CMU-SAFARI/transpimlib

ETH Zürich

SAFARI

Friday, December 22, 2023