INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING
ICPP/2021/CHICAGO/USA
acm In-Cooperation
sighpc
AUGUST 9-12, 2021

# WAVE-PIM: ACCELERATING WAVE SIMULATION USING PROCESSING-IN-MEMORY

**Bagus Hanindhito***
**Ruihao Li***
**Dimitrios Gourounas**

The University of Texas at Austin
Austin, TX, USA

**Arash Fathi**
**Karan Govil**
**Dimitar Trenev**

ExxonMobil Research and Engineering
Annandale, NJ, USA

**Andreas Gerstlauer**
**Lizy K. John**

The University of Texas at Austin
Austin, TX, USA

*Both authors contributed equally to this research.

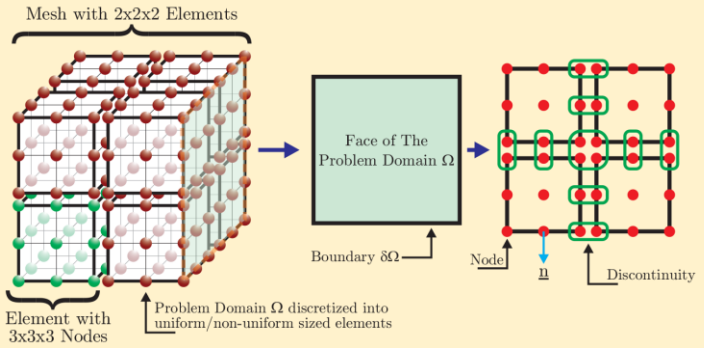
# Brief Introduction to Wave Simulation

## PRELUDE TO WAVE SIMULATION

- Acoustic wave equation
  - Approximate the propagation of compressional waves in the Earth, water, and body tissue.
  - Four variables: Pressure (P), Particle Velocities (Vx, Vy, Vz).
- Elastic wave equation
  - Describes propagation of compress-ional and shear waves in elastic solids.
  - Nine variables.
- Wide applications
  - Oil and gas exploration, earthquake hazard mitigation, oceanography, medical imaging, defense systems.

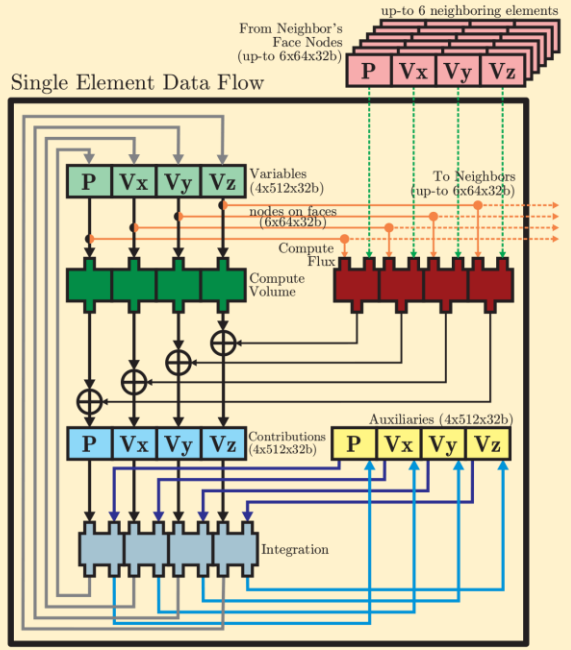
## DG DISCRETIZATION

- Problem domain is discretized into multiple elements using discontinuous Galerkin (dG) method.




  - The solution for each node inside an element is local (Volume).
  - To solve discontinuity, Flux is computed and depends on neighboring elements.

## SIMULATION DATAFLOW




- Dataflow for each element in one int. step.
- One time step consists of 5 int. steps.

# Design Motivation

## BOTTLENECKS IN CPU/GPU IMPLEMENTATION

- Original code uses p4est for mesh generation and workload distribution into multiple CPUs in multi-node environment.
  - Even for small problem, CPU code takes significant time to run.
- For acoustic wave simulation at refinement-level 5, the GPU implementation can achieve 131.10x, 223.95x, and 369.05x on GTX 1080Ti, Tesla P100, and Tesla V100, respectively, compared to dual Intel Xeon Platinum 8160 (48 cores).
- Upon profiling, GPU has not reached their peak performance.
  - Wave simulation performance is bounded by memory bandwidth, even for Tesla V100 GPU with 900GB/s memory bandwidth.
- The excessive data movements not only limit performance, but also costs energy that can be higher than the computation energy itself.

## BOTTLENECKS IN CURRENT PIM ARCHITECTURES

- Although PIM can reduce data movements between off-chip and on-chip memory, the internal data movement between memory subarray to another can affects performance.
- FloatPIM [26] solution does not explore non-neighbor communication, and thus not a general solution.
- Other works [33,43] considered an identical latency for all inter-block transmission using global buffer.
  - Inefficient for data transfer between adjacent blocks.

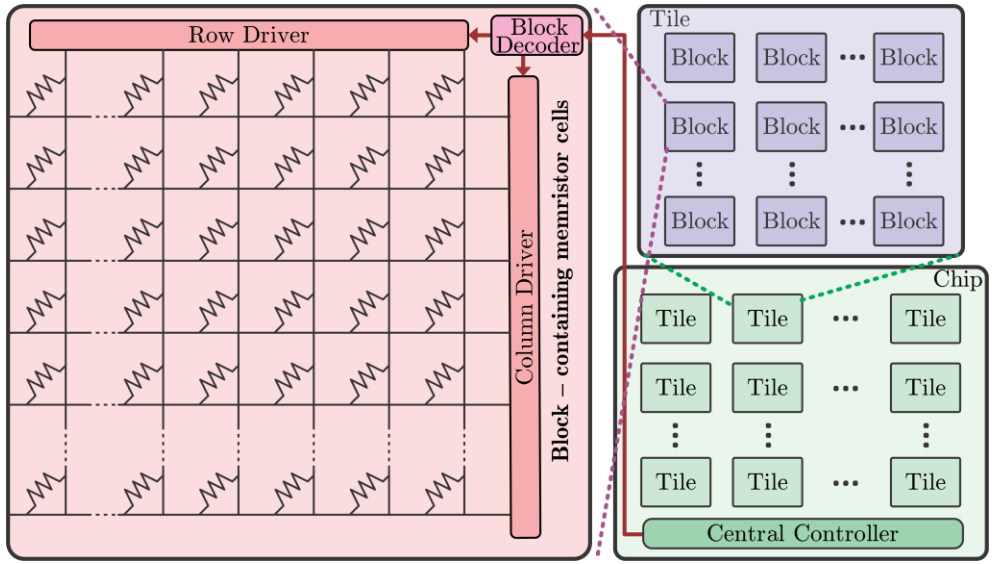
[26] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. 2019. Floatpim: In-memory acceleration of deep neural network training with high precision. In Proceedings of the 46th International Symposium on Computer Architecture. ACM, 802–815.

[33] Ruihao Li, Shuang Song, Qinzhe Wu, and Lizy. K John. 2020. Accelerating Forcedirected Graph Layout with Processing-in-Memory Architecture. In 2020 27th IEEE International Conference on High Performance Computing, Data, & Analytics (HiPC). IEEE.

[43] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. Pipelayer: A pipelined reram-based accelerator for deep learning. In 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 541–552.

# Wave-PIM Architecture

- Wave-PIM architecture is a digital PIM.
  - Each PIM chip consists of multiple memory tiles.
  - Each tile has multiple memory blocks, which is the most basic unit.
  - Each memory block contains memristor memory cells, sense amplifiers, decoders, row/column drivers, and row/column buffers.
- Each memory block is identical and can perform computation independently.
  - Computations are performed inside the blocks in a bit-serial way by utilizing NOR Operations.
  - **There is no need for separate ALU hardware.**
- Host CPU controls the Wave-PIM by sending instructions.
  - Instructions are decoded by the PIM Chip.
  - Micro-sequences are generated by the decoder and sent to each memory block.
  - Computations are performed in row-parallel way. The operands must be ready in each row before computation is started and the results are stored in each row.

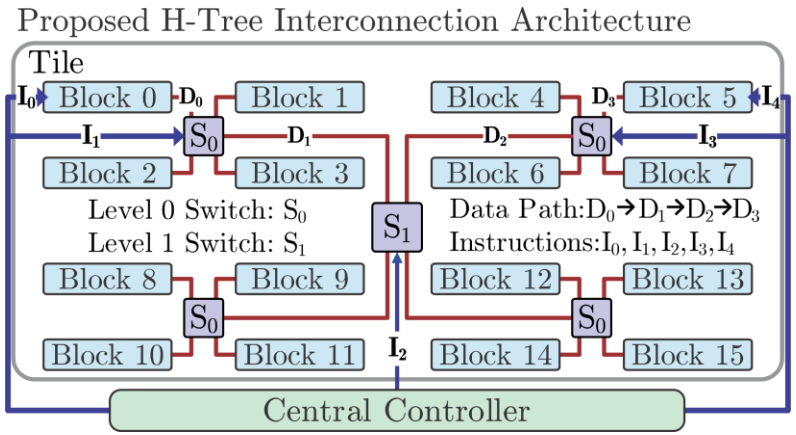



| Components | Param | Value | Power |
|---|---|---|---|
| Crossbar Array | size | 1Mb | 6.14mW |
| Sense Amp | number | 1K | 2.38mW |
| Decoder | number | 1 | 0.31mW |
| Memory Block | number | 1 | 8.83mW |
| Tile Memory | num_block | 256 | 1.57W |
| H-tree Switch | number | 85 | 107.13mW |
| Bus Switch | number | 1 | 17.2mW |
| Tile | Size | 32MB | 1.68W (H-tree)<br>1.59W (Bus) |
| Central Controller | number | 1 | 6.41W |
| CPU Host | number | 1 | 3.06W |
| Total | Size | 2GB | 115.02W (H-tree)<br>109.25W (Bus) |

# Inter-block Interconnection

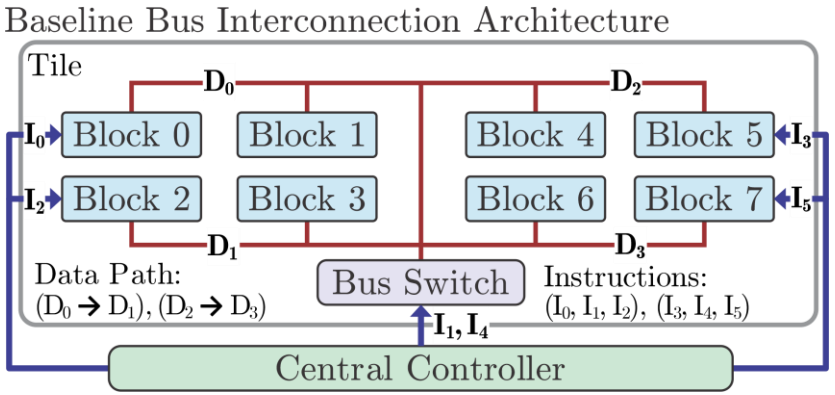
**H-TREE**

- Suitable for use if there are lot inter-block data movements.
- Multiple inter-block data transmission can be processed in-parallel, if they are not using same switch, improving efficiency and reducing overhead.
  - Data transmission between block 0 and block 2 can be processed in-parallel with data transmission between block 5 and block 7.
- Example: Data transmission from block 0 to Block 5
  - One read instruction (I0) is sent to block 0 to load data to row/column buffer.
  - Memcpy instructions (I1, I2, I3) are sent to each H-tree nodes along the data path.
  - One write instruction (I4) is sent to block 5 to store the data.




**BUS**

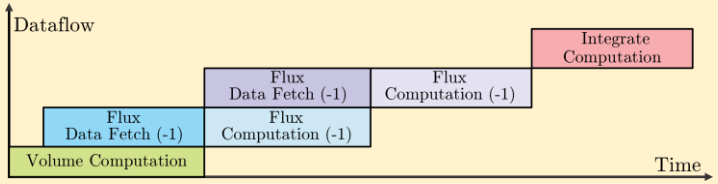
- Suitable for use if there is little inter-block data movements.
- Bus only need one central-bus switch, reducing the leakage power.
  - In contrast, H-Tree needs 1+4+16+64 = 85 H-Tree Node Switches for 256-block memory tile.
- Unfortunately, the data transmission must be processed sequentially.
  - Data transmission between block 0 and block 2 cannot be processed in parallel with data transmission between block 5 and block 7.



# Optimizing PIM Performance

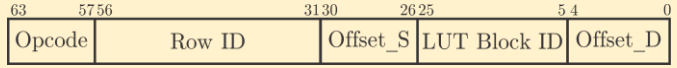
## PIPELINING

- Flux
  - Additional PIM blocks are reserved for buffering neighboring element's data.
  - 6-step pipelining (3 axis, 2 normal vectors) with no data dependencies between them.
  - Neighbor data can be fetched while computing the Volume.
- Volume and Integration
  - Unable to pipeline due to hardware hazard and intra-block communication.




## LOOK-UP TABLE

- Look-up table is constructed at the beginning to make it easier for PIM to identify the neighboring elements.
  - Supports for both regular and irregular mesh structures, if the structures remain the same throughout the runtime.




**Algorithm 1** Steps for executing one look-up table instruction.

1: Generating read instruction $R\_1$ (location = Row Address 1024 + Offset_S × 32, size = 32).
2: Issuing $R\_1$ to fetch the 32-bit *index*.
3: Generating read instruction $R\_2$ (location = LUT Block ID 1024 × 1024 + *index* × 32, size = 32).
4: Issuing $R\_2$ to fetch the 32-bit content *data*.
5: Generating write instruction $W\_1$ (location = Row Address 1024 + Offset_D × 32, size = 32, content = *data*).
6: Issuing $W\_1$.

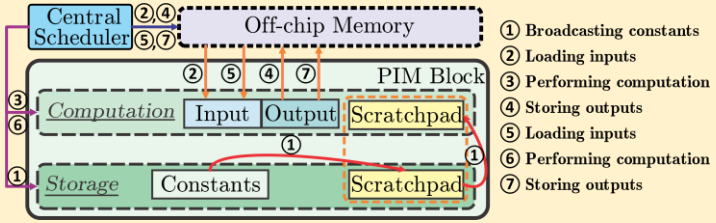
## OFFLOAD COMPUTATION

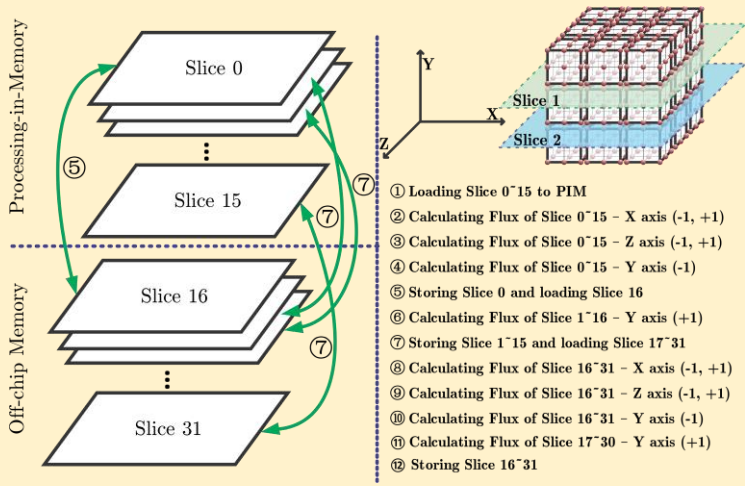
- Complicated arithmetic operations cannot be efficiently implemented with NOR operations.
  - Square Root
  - Inverse
- The host CPU is used to offload these operations and the result is buffered in look-up table inside PIM.
  - In Wave Simulation, the number of square root and inverse operations are significantly smaller compared to the problem domain size.

# Fitting The Problem Domain to PIM

## BATCHING

- For problem size that is larger than the capacity of Wave-PIM.
- The data is loaded in smaller batch from off-chip memory to Wave-PIM for computation.
  - Overlap the overhead of data movements with computation.
- Volume and Integration
  - Batching is easy to implement.




- Flux
  - More difficult to implement due to inter-element data dependency.
  - Divide the problem domain alongside z-axis to form a slice. Pre-load a slice while waiting other to finish.

## EXPANSION

- For problem size that is smaller than the capacity of Wave-PIM.
- Improve performance by utilizing more memory blocks that are normally left idle.
  - Need to duplicate some of data.
  - More inter-block data movements, especially for Volume.
- Compute each variables in parallel by distributing them into different memory blocks.

# Evaluation

## EXPERIMENT PLATFORMS

- Three GPU-based Platforms.
  - GTX 1080Ti, Tesla P100, Tesla V100
- Four different size Wave-PIM Platforms.
  - 512MB, 2GB, 8GB, 16GB.

| Platform | GPU | GPU | GPU | PIM |
|---|---|---|---|---|
| Name | GTX 1080Ti | Tesla P100 | Tesla V100 | Wave-PIM |
| Host CPU Model | Xeon E5-2620 v4 | Xeon Platinum 8160 | Xeon Platinum 8160 | ARM Cortex-A72 |
| Process Node | 16nm | 16nm | 12nm | 28nm |
| Clock Frequency | 1,530MHz | 1,480MHz | 1,582MHz | 900MHz |
| Register Size | 7,168KB | 14,336KB | 20,480KB | N/A |
| L2 Cache Size | 2,816KB | 4,096KB | 6,144KB | N/A |
| Memory Size and Type | 11GB GDDR5X | 16GB HBM2 | 16GB HBM2 | 512MB, 2GB, 8GB, 16GB |
| Memory BW | 484GBps | 720GBps | 900GBps | 900GBps |
| FP32 CUDA Cores | 3,584 | 3,584 | 5,120 | N/A |
| FP32 Peak Throughput | 11.3TFLOPS | 10.6TFLOPS | 15.7TFLOPS | 1.7, 6.6, 26.4, 52.8 TFLOPS |

## BENCHMARKS

- Acoustic Wave Simulations
  - Refinement-Level 4 and 5.
- Elastic Wave Simulations
  - Refinement-Level 4 and 5.
  - Central and Riemann Flux Solver.
- Single-precision FP OPS in the range of 400M-12B per kernel launch.

| Benchmark | Refinement Level | Number of Elements | Number of Instructions[1] | Number of FP Ops.[2] |
|---|---|---|---|---|
| Acoustic_4 | 4 | 4,096 | 2,140,930,048 | 391,380,992 |
| Elastic-Central_4 | 4 | 4,096 | 3,465,543,680 | 990,117,888 |
| Elastic-Riemann_4 | 4 | 4,096 | 9,870,131,200 | 1,472,200,704 |
| Acoustic_5 | 5 | 32,768 | 17,127,440,384 | 3,131,047,936 |
| Elastic-Central_5 | 5 | 32,768 | 27,724,349,440 | 7,920,943,104 |
| Elastic-Riemann_5 | 5 | 32,768 | 78,960,159,424 | 11,777,661,440 |

[1] From `inst_executed` metric multiplied by 32 to obtain thread-level value.
[2] From `flop_count_sp` and `flop_count_sp_special` metrics.
[1,2] Obtained using `nvprof` running on Tesla V100 with fused implementation. Values are the total from each kernel launched once. In each time-step, each kernel is launched five times.

## IMPLEMENTATION

- GPU Kernel Implementation
  - Unfused
  - Fused with optimized data movements and neighbor look-up.
- Wave-PIM Implementation
  - Naïve implementation (N)
  - Expansion for Acoustic ($E_a$)
  - Expansion for Elastic ($E_e$) / limited row.
  - Batching (B)

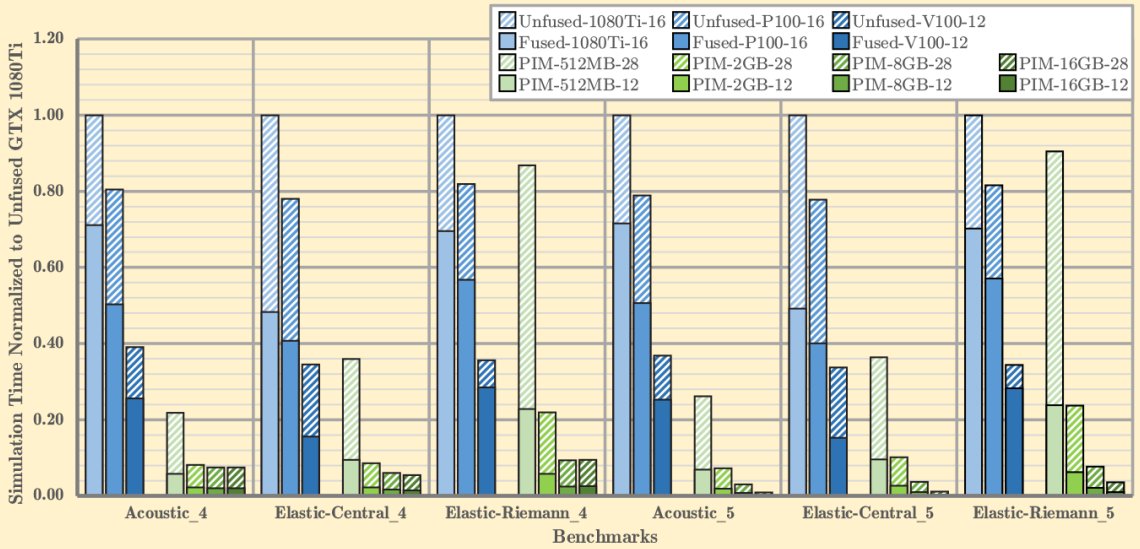
| Configuration | 512MB | 2GB | 8GB | 16GB |
|---|---|---|---|---|
| Acoustic_4 | $N$ | $E_a$ | $E_a$ | $E_a$ |
| Elastic_4 | $E_e$&$B$ | $E_e$ | $E_a$&$E_e$ | $E_a$&$E_e$ |
| Acoustic_5 | $B$ | $B$ | $N$ | $E_a$ |
| Elastic_5 | $E_e$&$B$ | $E_e$&$B$ | $E_e$&$B$ | $E_e$ |

$N$ represents the naive implementation, $E_a$ represents implementation using expansion to increase the parallelism (can be used for both acoustic and elastic wave simulation), $E_e$ represents implementation using expansion due to limited row size (only exists in the elastic wave simulation) , and $B$ represents implementation using the batching technique.

# Performance and Energy Comparison

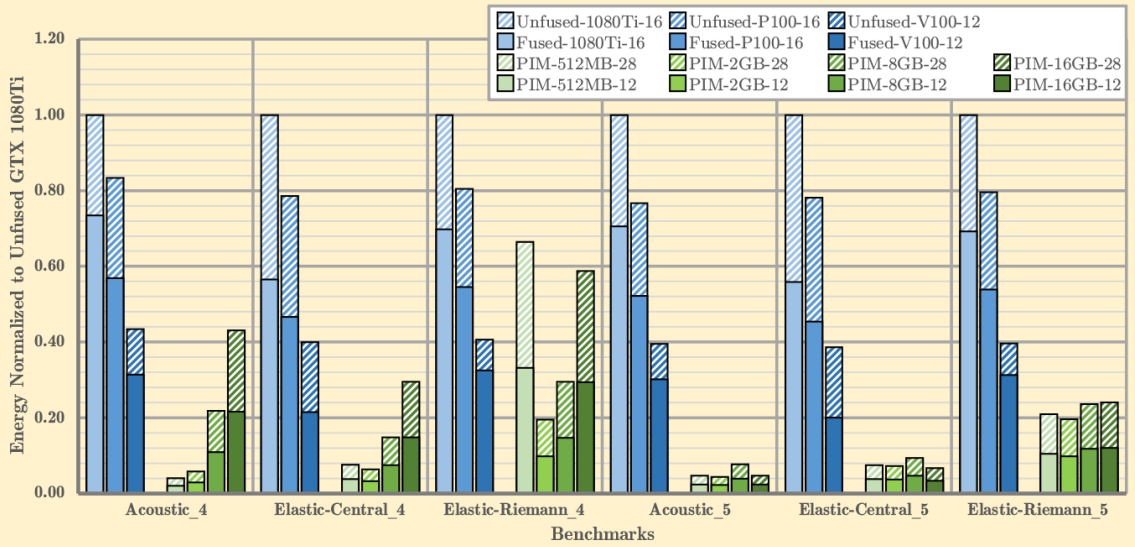
We report both unscaled (28nm) and scaled (12nm) for Wave-PIM performance and energy consumption.

## PERFORMANCE COMPARISON




- Lower is better.
- Due to the reduction in data-movements, Wave-PIM can achieve up-to 414.37x speed-up compared to GPU implementation.

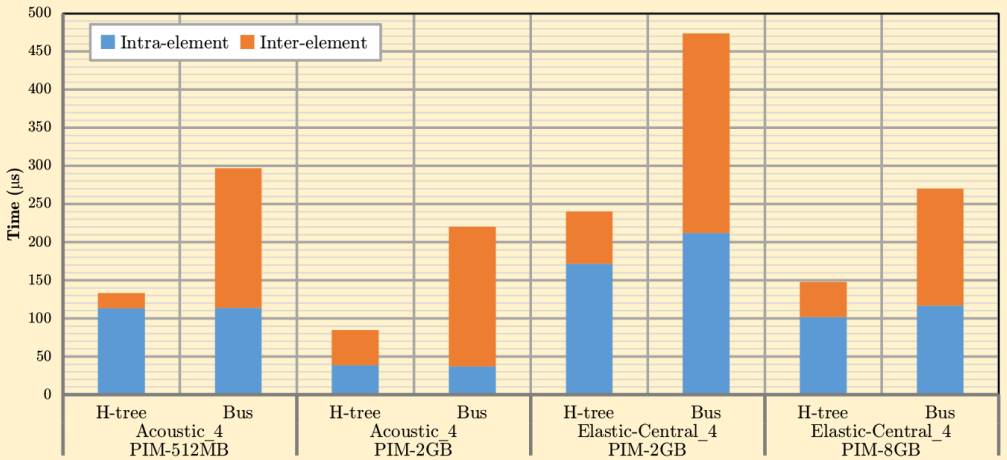
## ENERGY COMPARISON




- Lower is better.
- Up-to 50.56x energy savings compared to GPU.
- Small problem may not benefit from larger PIM capacity.

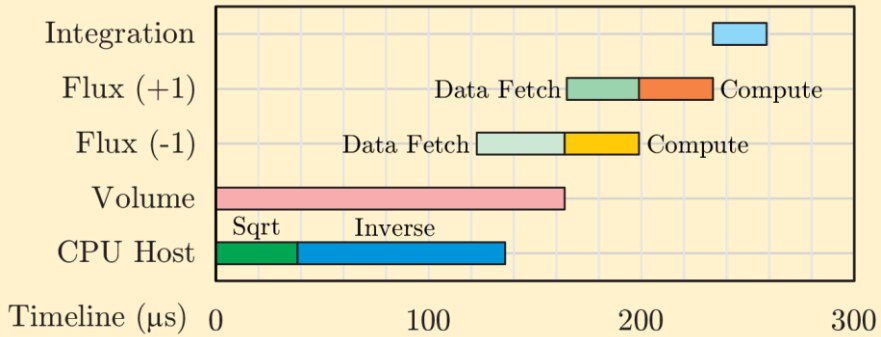
# Interblock Communication and Pipelining Analysis

## H-TREE VS. BUS: ANALYSIS




- Both H-Tree and Bus have same performance if there is little inter-block data transmission (e.g., volume and integration).
  - Bus has lower leakage power.
- When there are a lot of inter-block data transmission (e.g., flux), H-tree is significantly better.

## PIPELINE ANALYSIS




- The square-root and inverse operations are offloaded into host CPU and processed simultaneously with volume and first neighboring element data fetch for flux.
- Flux is divided into two parts based on normal vector direction along z-axis.
- Without pipelining, the Wave-PIM can only reach 0.77x throughput.

# Conclusion

- Processing-in-Memory reduces data movement and allows harnessing the data-level parallelism that is abundant in many scientific computing applications.
- We profiled our GPU implementation of the acoustic wave simulation. Although the GPU solution can provide up to 369x speed-up over a high-end CPU, we find the main bottleneck is the off-chip data movement, which makes the simulation memory-bound, even with 900GB/s of off-chip HBM2 DRAM bandwidth.
- Data-layout is a key factor affecting the overall performance of the PIM, and thus we laid out the data in a hardware-friendly manner for the PIM architecture to minimize the overhead of inter-element data transfer.
- We investigated H-tree and Bus interconnects to balance the tradeoffs between the latency and energy consumption caused by the inter-element data transfer and observe that the H-tree results in approximately 2.16× time savings in comparison to a bus architecture.
- We offer solutions to fold the workloads in batches or expand the workloads, to improve the scalability of the design, making it capable to support larger or smaller problem sizes at the highest possible performance.
- We evaluated our PIM design with extensive experiments on different PIM configurations. Compared to three state-of-the-art GPU platforms: GTX 1080Ti, Tesla P100, and Tesla V100, our PIM system yields an average performance increase of 45.31×, 34.52×, and 15.89×, and energy savings of 13.75×, 10.67×, and 5.66×, respectively.