PRIME: Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory

Presented by: Ravi Raju

March 23, 2018

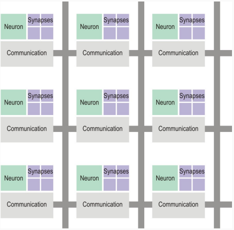
QII Presentation Spring 2018

Overview

Problem Statement and Solution

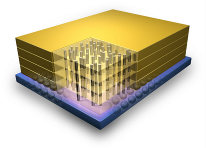
Problem Statement/Motivation

1. Neural Networks
    - Popular for image/speech recognition application
    - High Memory Bandwidth Requirement
2. Current Solutions
    - DaDianNao - large on-chip eDRAM for high bandwith and data locality
    - TrueNorth - SRAM crossbar memory for synapses
3. Both solutions suffer from latency of data movement

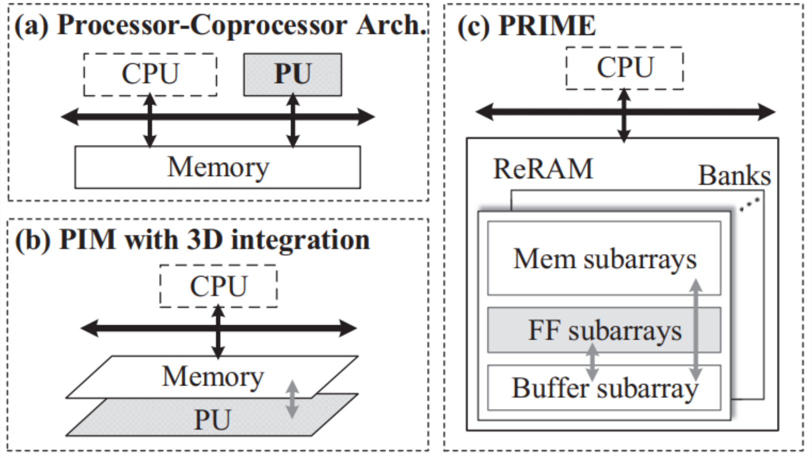


Proposed Solution: PRIME

1. Processing in Memory is a natural solution
    - Inspired by HMC
    - Place compute units in memory to do NN computation
    - Latency of In-memory data communication vs. DRAM memory access
2. PRIME
    - ReRAM crossbar array solution
    - Dynamically reconfigure between NN accerelation and memory
        - 2.1 Architectural/circuit level support
        - 2.2 Software interface
3. Targets large-scale MLP and CNN applications



Key Idea: PRIME



Background

What is ReRAM

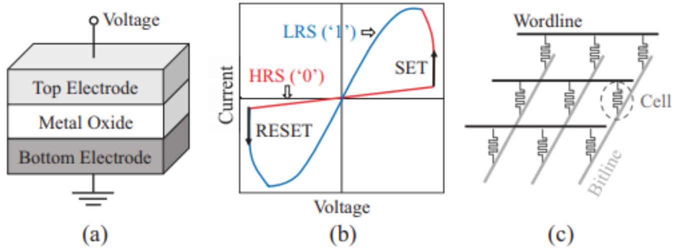


Figure 1. (a) Conceptual view of a ReRAM cell; (b) I-V curve of bipolar switching; (c) schematic view of a crossbar architecture.

ReRAM in relation to Neural Nets

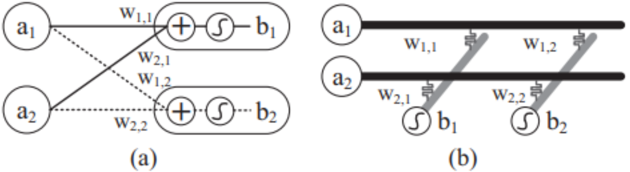
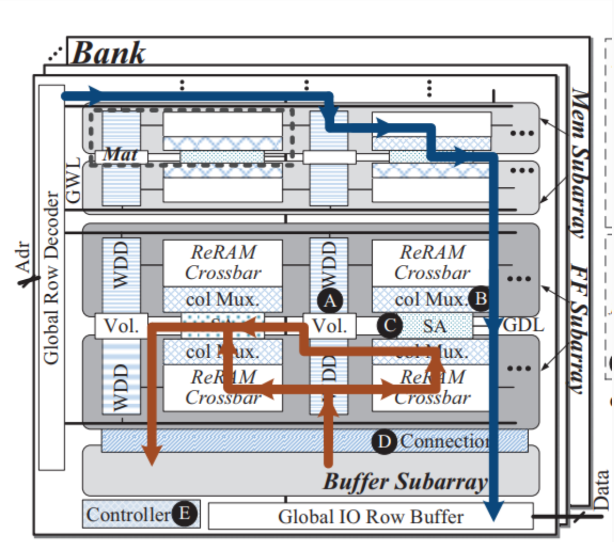


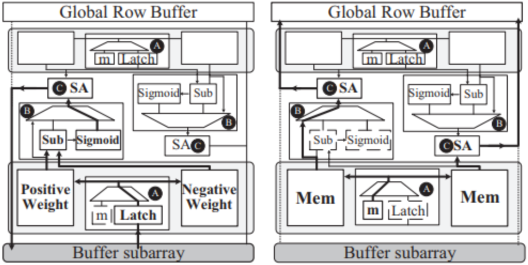
Figure 2. (a) An ANN with one input/output layer; (b) using a ReRAM crossbar array for neural computation.

Architecture and System Design

High Level Overview of Architecture

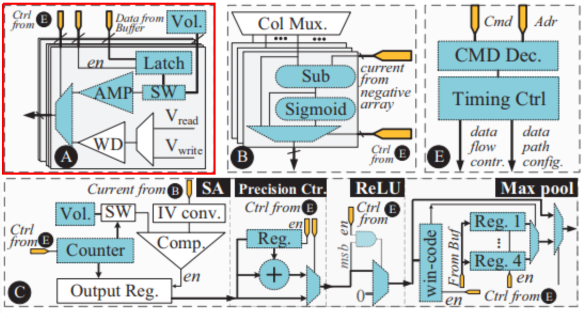


Choosing between NN Computation and Memory Mode

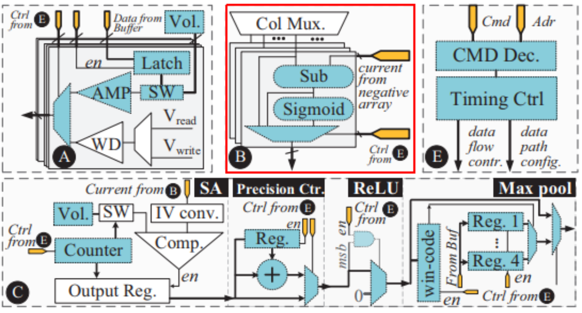


(a)
(b)

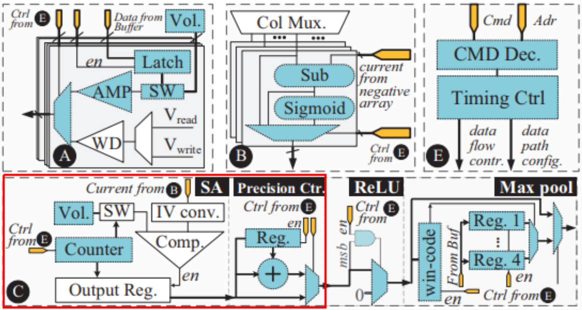
MicroArchitecture of FF SubArray: Decoder



MicroArchitecture of FF SubArray: Col Mux



MicroArchitecture of FF SubArray: SA



System Level Design

- Small-Scale NN: Replication
- Medium-Scale NN: Split-Merge
- Large-Scale NN: Inter-Bank Communication

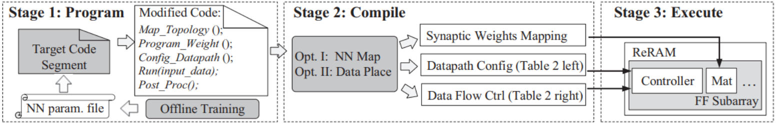


Figure 7. The software perspective of PRIME: from source code to execution.

Evaluations and Results

Experimental Setup

Table III
THE BENCHMARKS AND TOPOLOGIES.

MlBench		*MLP-S*	784-500-250-10
CNN-1	conv5x5-pool-720-70-10	*MLP-M*	784-1000-500-250-10
CNN-2	conv7x10-pool-1210-120-10	*MLP-L*	784-1500-1000-500-10
VGG-D	conv3x64-conv3x64-pool-conv3x128-conv3x128-pool conv3x256-conv3x256-conv3x256-pool-conv3x512 conv3x512-conv3x512-pool-conv3x512-conv3x512 conv3x512-pool-25088-4096-4096-1000		

Table IV
CONFIGURATIONS OF CPU AND MEMORY.

Processor	4 cores; 3GHz; Out-of-order
L1 I&D cache	Private; 32KB; 4-way; 2 cycles access;
L2 cache	Private; 2MB; 8-way; 10 cycles access;
ReRAM-based Main Memory	16GB ReRAM; 533MHz IO bus; 8 chips/rank; 8 banks/chip; tRCD-tCL-tRP-tWR 22.5-9.8-0.5-41.4 (ns)

Table V
THE CONFIGURATIONS OF COMPARATIVES.

Description		Data path	Buffer
pNPU-co	Parallel NPU [17] as co-processor	16×16 multiplier 256-1 adder tree	2KB in/out 32KB weight
pNPU-pim	PIM version of parallel NPU, 3D stacked to each bank		

Performance

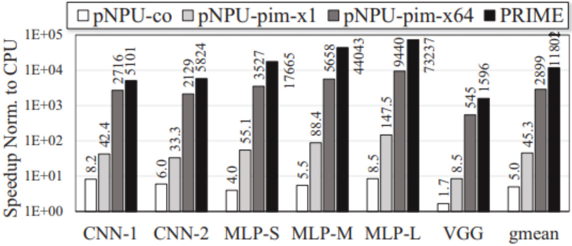


Figure 8. The performance speedups (*vs.* CPU).

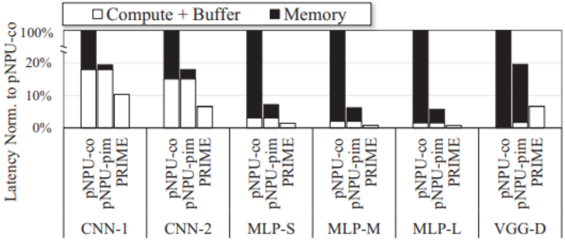


Figure 9. The execution time breakdown (*vs.* pNPU-co).

Energy

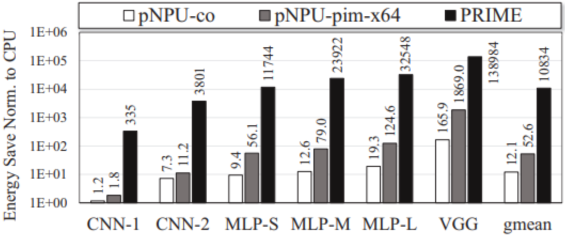


Figure 10. The energy saving results (*vs.* CPU).

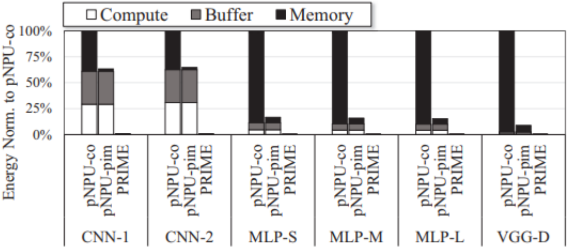


Figure 11. The energy breakdown (*vs.* pNPU-co).

Area

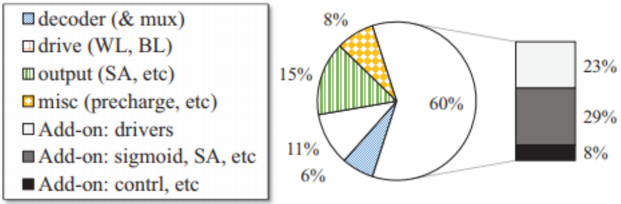


Figure 12. Area Overhead of PRIME.

Discussion

Critique

1. PRIME only supports unsigned input vectors
2. Tough to support recurrent NNs/LSTMs architectures
3. Opportunity to exploit sparsity of NN for energy savings
    - Introduce some logic into pipeline to check how many non-zero weights in crossbar
    - If below some defined threshold, skip the computation

Conclusion

- PRIME is the solution to the data movement and high memory bandwidth problem
- Using ReRAM crossbar accelerates NN computation
- Circuit/microarchitectural changes as well as software interface enables wide spectrum of NN applications
- Very little area overhead and extra processing elements

References

- P. Chi, et al. (2016, Sept. 4). *PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory* [Online]. Available: http://ieeexplore.ieee.org/document/7551380/citations?tabFilter=papers
- A. Shafiee, et al. (2016, Aug. 25). *ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars* [Online]. Available: http://ieeexplore.ieee.org/document/7551379/

Thank you and Questions

Appendix

Precision Issues

- Input precision
- Weight precision
- Output precision

- Multiple low-precision input signals
- Multiple cells to make one high precision weight
- Multiple phases for one computation

Precision Graph

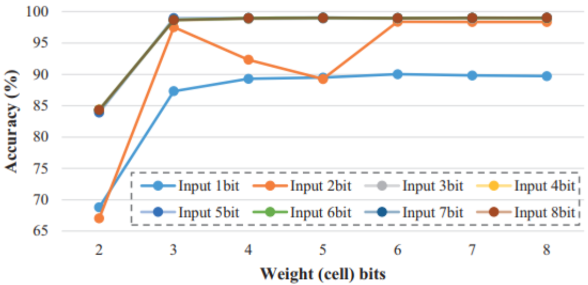


Figure 6. The precision result.

Math for precision

$$R_{\text{full}} = \sum_{i=1}^{2^{P_N}} \left(\sum_{k=1}^{P_{\text{in}}} I_k^i 2^{k-1} \cdot \sum_{k=1}^{P_w} W_k^i 2^{k-1} \right)$$

$$R_{\text{target}} = R_{\text{full}} \gg (P_{\text{in}} + P_w + P_N - P_o).$$

$$\text{input: } \sum_{k=1}^{P_{\text{in}}} I_k^i 2^{k-1} = \sum_{k=1}^{P_{\text{in}}/2} (Ih_k^i 2^{k-1} \cdot 2^{P_{\text{in}}/2} + Il_k^i 2^{k-1}) \quad (4)$$

$$\text{weight: } \sum_{k=1}^{P_w} W_k^i 2^{k-1} = \sum_{k=1}^{P_w/2} (Wh_k^i 2^{k-1} \cdot 2^{P_w/2} + Wl_k^i 2^{k-1}). \quad (5)$$

Math for precision

$$
\begin{aligned}
R_{\text{full}} = \sum_{i=1}^{2^{P_N}} \{ 2^{\frac{P_w+P_{\text{in}}}{2}} \cdot & \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Ih_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wh_k^i 2^{k-1}}_{HH-part} \\
+ 2^{\frac{P_w}{2}} \cdot & \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Il_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wh_k^i 2^{k-1}}_{HL-part} \qquad (6) \\
+ 2^{\frac{P_{\text{in}}}{2}} \cdot & \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Ih_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wl_k^i 2^{k-1}}_{LH-part} + \underbrace{\sum_{k=1}^{P_{\text{in}}/2} Il_k^i 2^{k-1} \sum_{k=1}^{P_w/2} Wl_k^i 2^{k-1}}_{LL-part} \}.
\end{aligned}
$$

$$
R_{\text{full}} = 2^{\frac{P_w+P_{\text{in}}}{2}} \cdot R_{\text{full}}^{\text{HH}} + 2^{\frac{P_w}{2}} \cdot R_{\text{full}}^{\text{HL}} + 2^{\frac{P_{\text{in}}}{2}} \cdot R_{\text{full}}^{\text{LH}} + R_{\text{full}}^{\text{LL}}.
$$

Math for precision

$$R_{tar}^{HH} = P_0 bitsof R_{full}^{HH}$$

$$R_{tar}^{HL} = P_0 - (P_{in}/2) bitsof R_{full}^{HL}$$

$$R_{tar}^{LH} = P_0 - (P_w/2) bitsof R_{full}^{LH}$$

$$R_{tar}^{LL} = P_0 - ((P_{in} - P_w)/2) bitsof R_{full}^{LL}$$

$$R_{tar} = R_{tar}^{HH} + R_{tar}^{HL} + R_{tar}^{LH} + R_{tar}^{LL}$$