Architecture, ISA support, and Software Toolchain for Neuromorphic Computing in ReRAM-Based Main Memory

Yuan Xie

University of California, Santa Barbara

yuanxie@ece.ucsb.edu

Research Overview

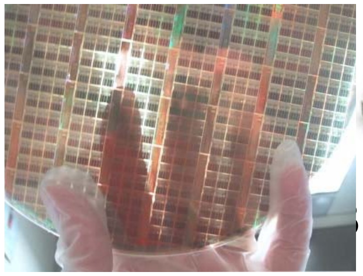
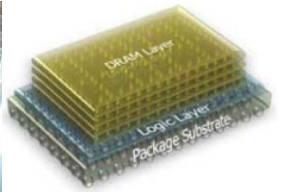
HPC

Mobile/Embedded

Application-Driven Innovations

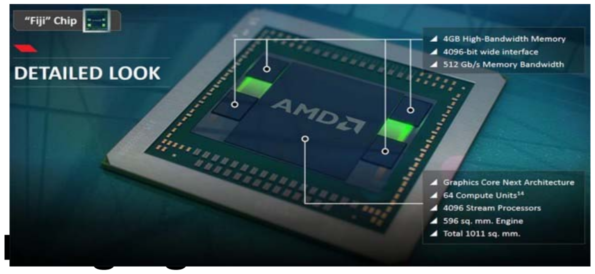
Computer Architecture Innovations

Technology-Driven Innovations

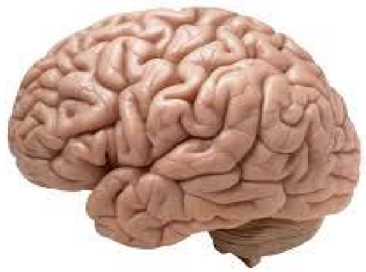
D Integration

Emerging
Technologies



Er

Research Overview

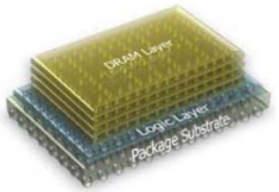


Brain-inspired Computing

Application-Driven Innovations

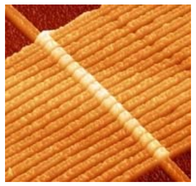
Our brain is a 3D structure with non-volatile memory capability

Technology-Driven Innovations



3D Integration

Emerging Technologies



Emerging NVM

Outline

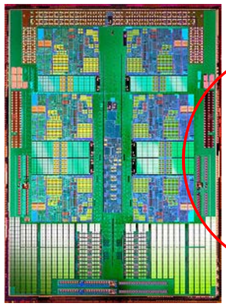
- Introduction and Motivation
- **PRIME:** Morphable Processing-In-Memory Architecture for NN computing
  - ISCA 2016
- **NISA:** Instruction Set Architecture for NN Accelerator
  - ISCA 2016
- **Neutrams:** Software Tool Chain for NN Accelerator
  - MICRO 2016
- Conclusion

Today's Von Neumann Architecture

Computing

Memory/Storage

CPU/GPU



On-chip memory (SRAM)

Off-chip memory (DRAM)

Solid State Disk (Flash Memory)

Secondary Storage (HDD)

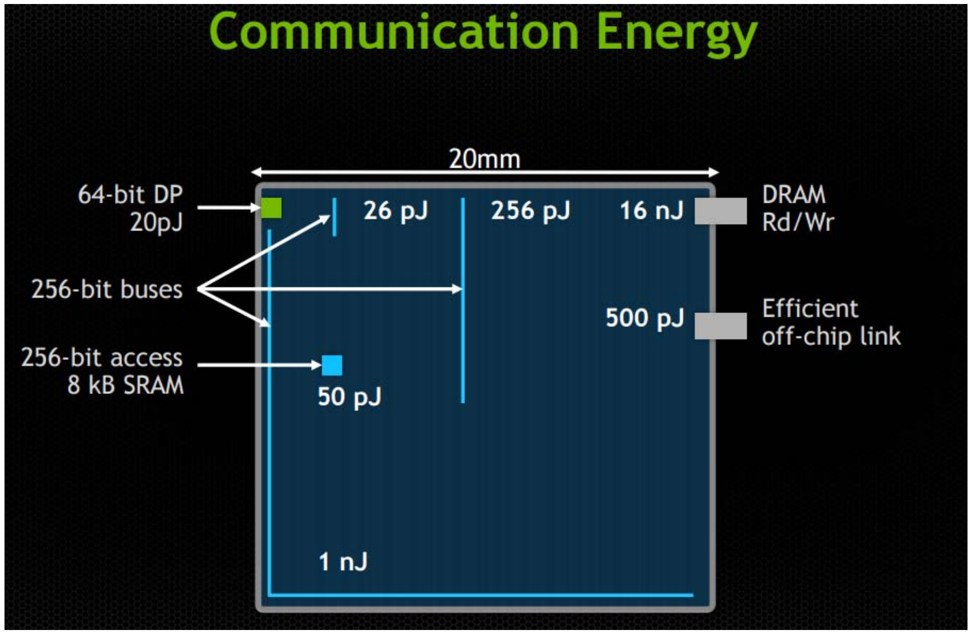
Latency: (Cycles)	On-chip memory (SRAM)	Off-chip memory (DRAM)	Solid State Disk (Flash Memory)	Secondary Storage (HDD)
	1~30	100~300	25000~2000000	>5000000

Challenge:

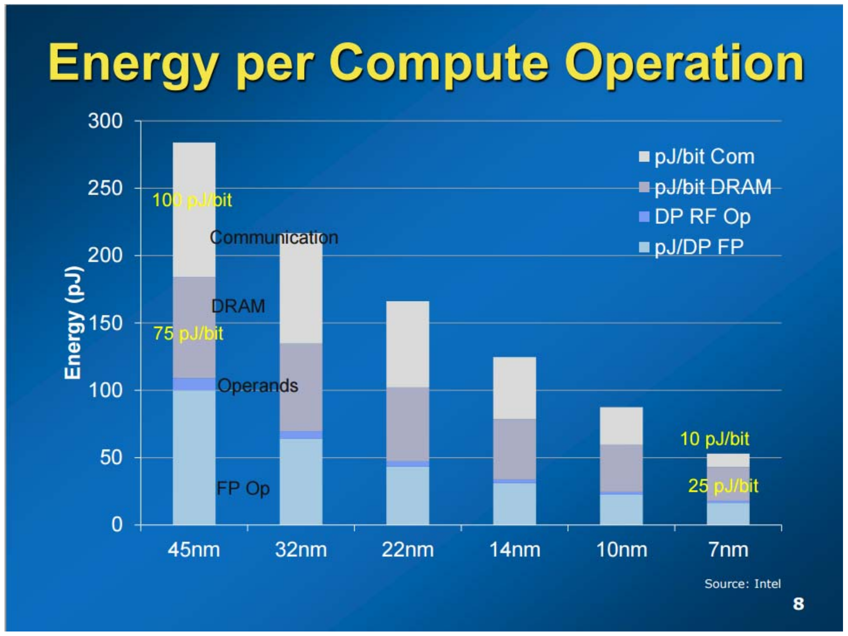
Bridging the Gap Between Computing and Memory Storage

Overhead of Data Movement

- Overhead for Data Movements
  - ~200x times more than floating-point computing itself
  - Technology improvement does not help



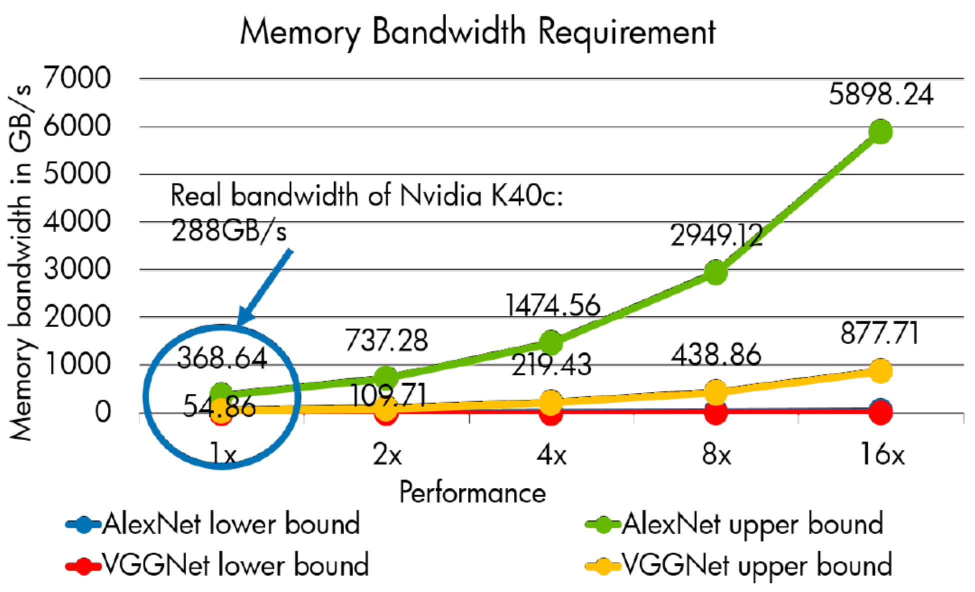
Bill Daily, “The Path to ExaScale”, SC14



Shekhar Borkar, “Exascale Computing—a fact or a fiction?”, IPDPS’13

Today’s NN and DL Acceleration

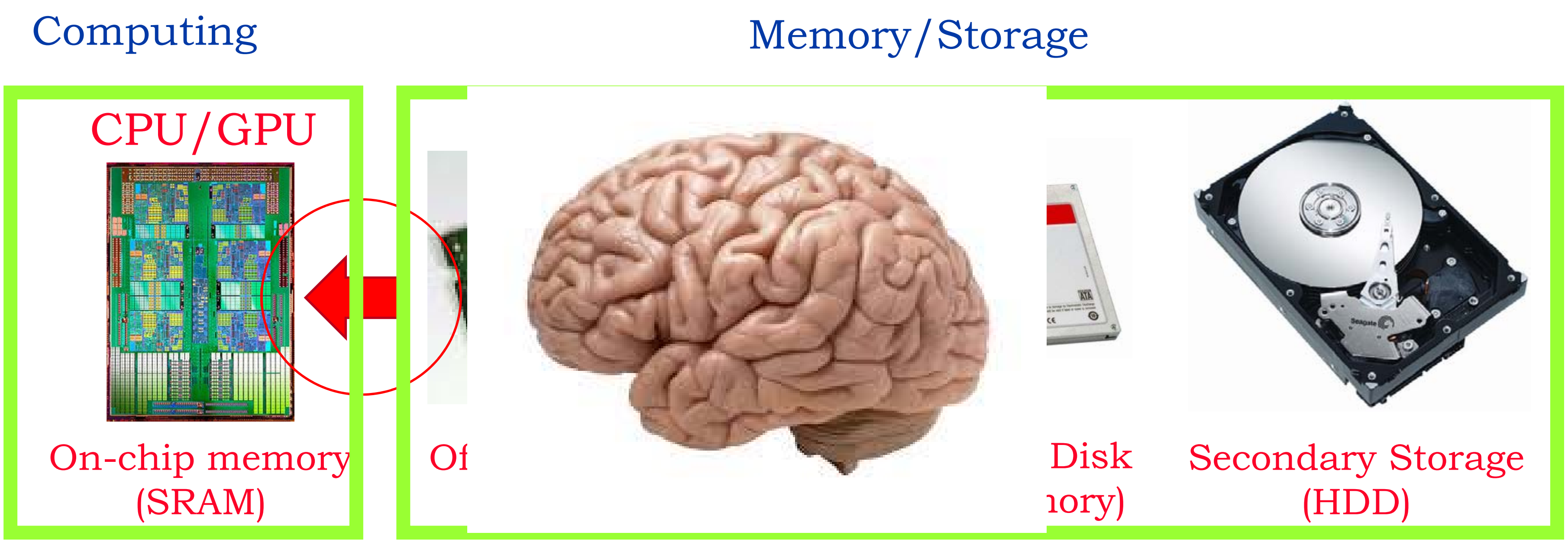
- Neural network (NN) and deep learning (DL)
  - Provide solutions to various applications
  - Acceleration requires high memory bandwidth
    - *PIM is a promising solution*



Deng *et al*, “Reduced-Precision Memory Value Approximation for Deep Learning”, HPL Report, 2015

- The size of NN increases
  - e.g., 1.32GB synaptic weights for Youtube video object recognition
- NN acceleration
  - GPU, FPGA, ASIC

Today's Von Neumann Architecture



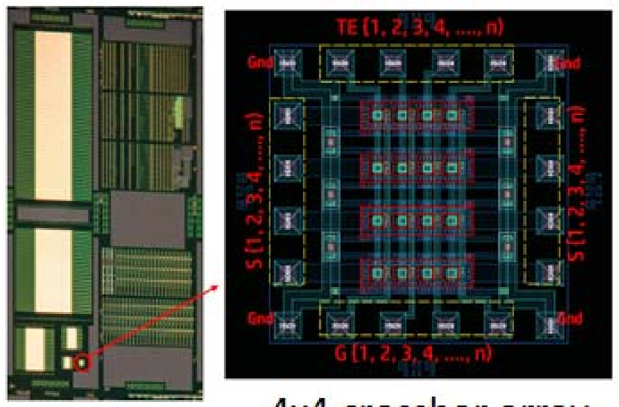
Our brain doesn't have a distinction of compute vs. memory

New Architecture:
In-Memory Computing with ReRAM-based Memory

Using ReRAM for Computing

- Resistive Random Access Memory (ReRAM)
  - Data storage: alternatives to DRAM and flash
  - Computation: matrix-vector multiplication (NN)

Hu *et al*, “Dot-Product Engine (DPE) for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication”, DAC’16.



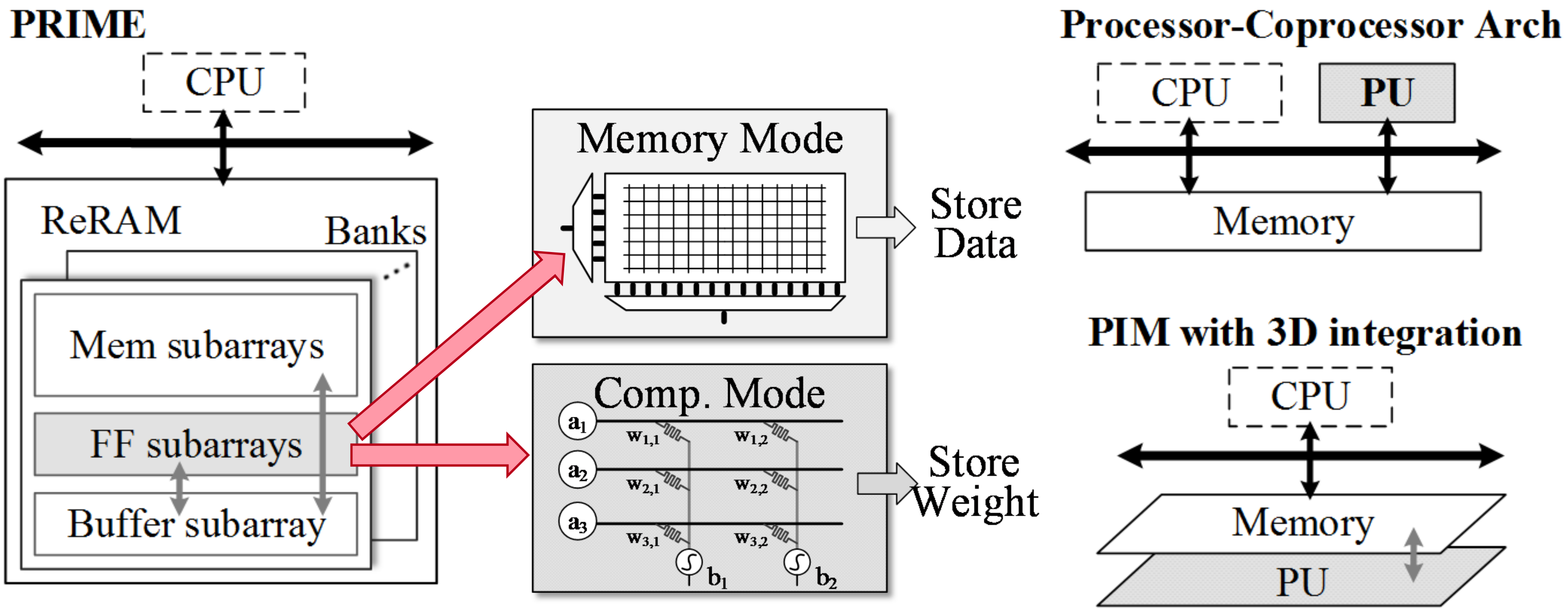
4x4 crossbar array

- Use DPE to accelerate pattern recognition on MNIST
  - no accuracy degradation vs. software approach (99% accuracy) with only 4-bit DAC and ADC requirement
  - 1,000X ~ 10,000X speed-efficiency product vs. custom digital ASIC

Shafiee *et al*, “ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars”, ISCA’16.

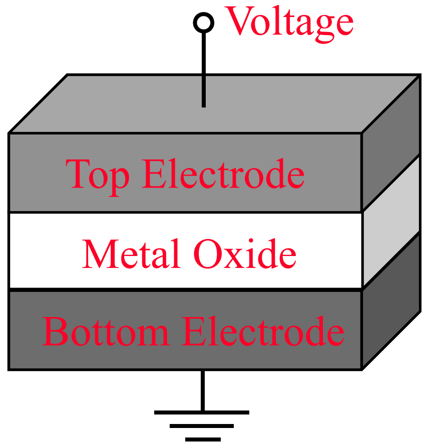
Key idea

- ❑ PRIME: Process-in-ReRAM main memory
  - ● Based on ReRAM main memory design[1]

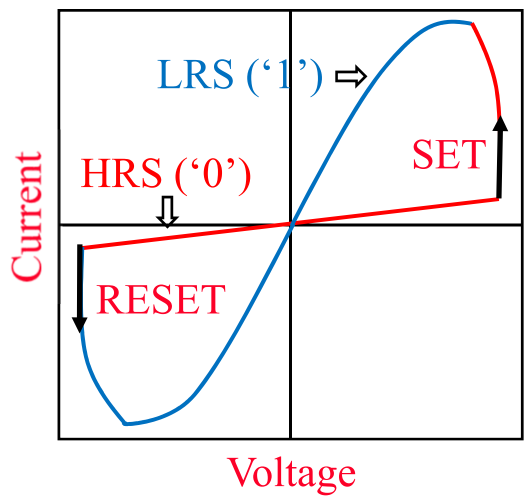


[1] C. Xu *et al*, "Overcoming the challenges of crossbar resistive memory architectures," in HPCA'15.

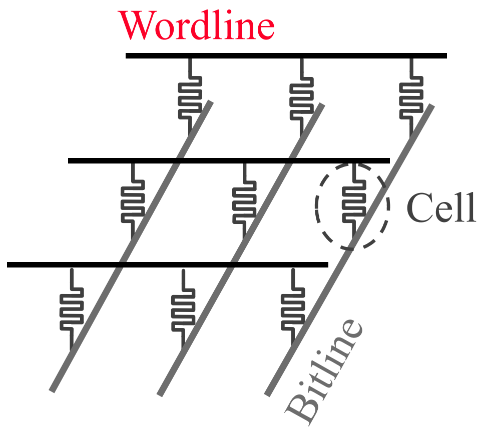
Memristor Basics



(a) Conceptual view of a ReRAM cell



(b) I-V curve of bipolar switching

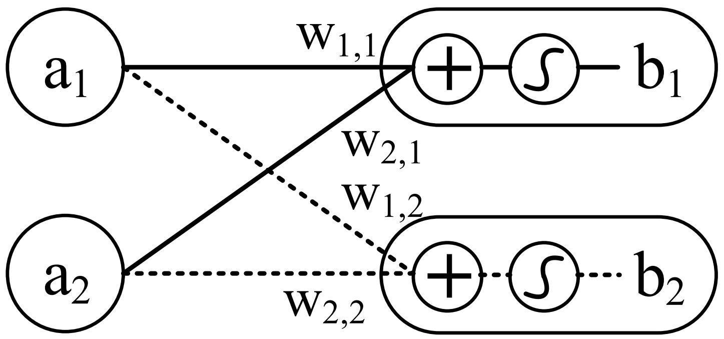


(c) schematic view of a crossbar architecture

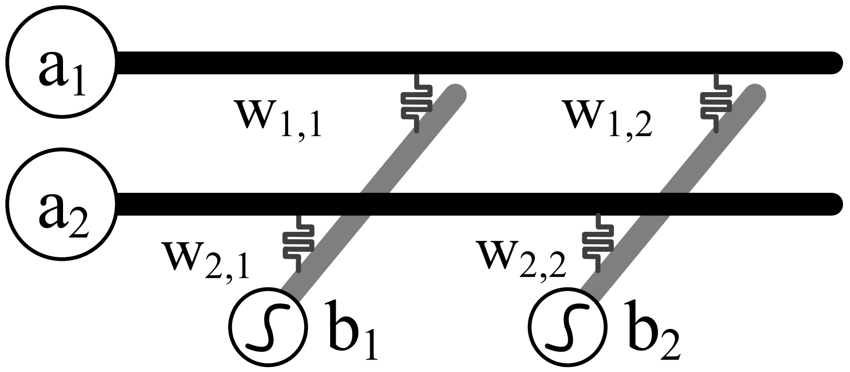
ReRAM Based NN Computation

- Require specialized peripheral circuit design
  - DAC, ADC *etc.*

$$b_j = \sigma(\sum_{\forall i} a_i \cdot w_{i,j})$$

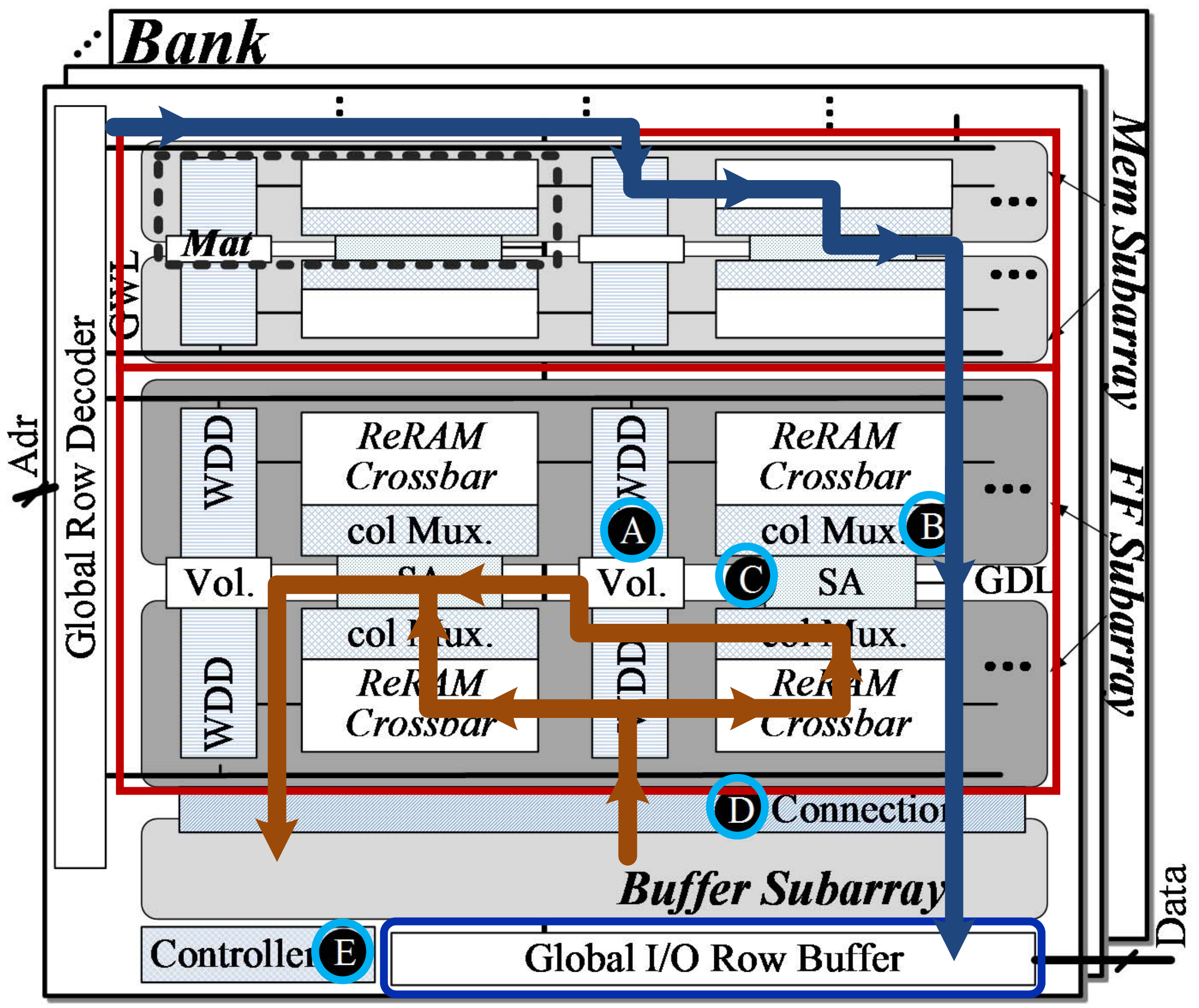


(a) An ANN with one input and one output layer



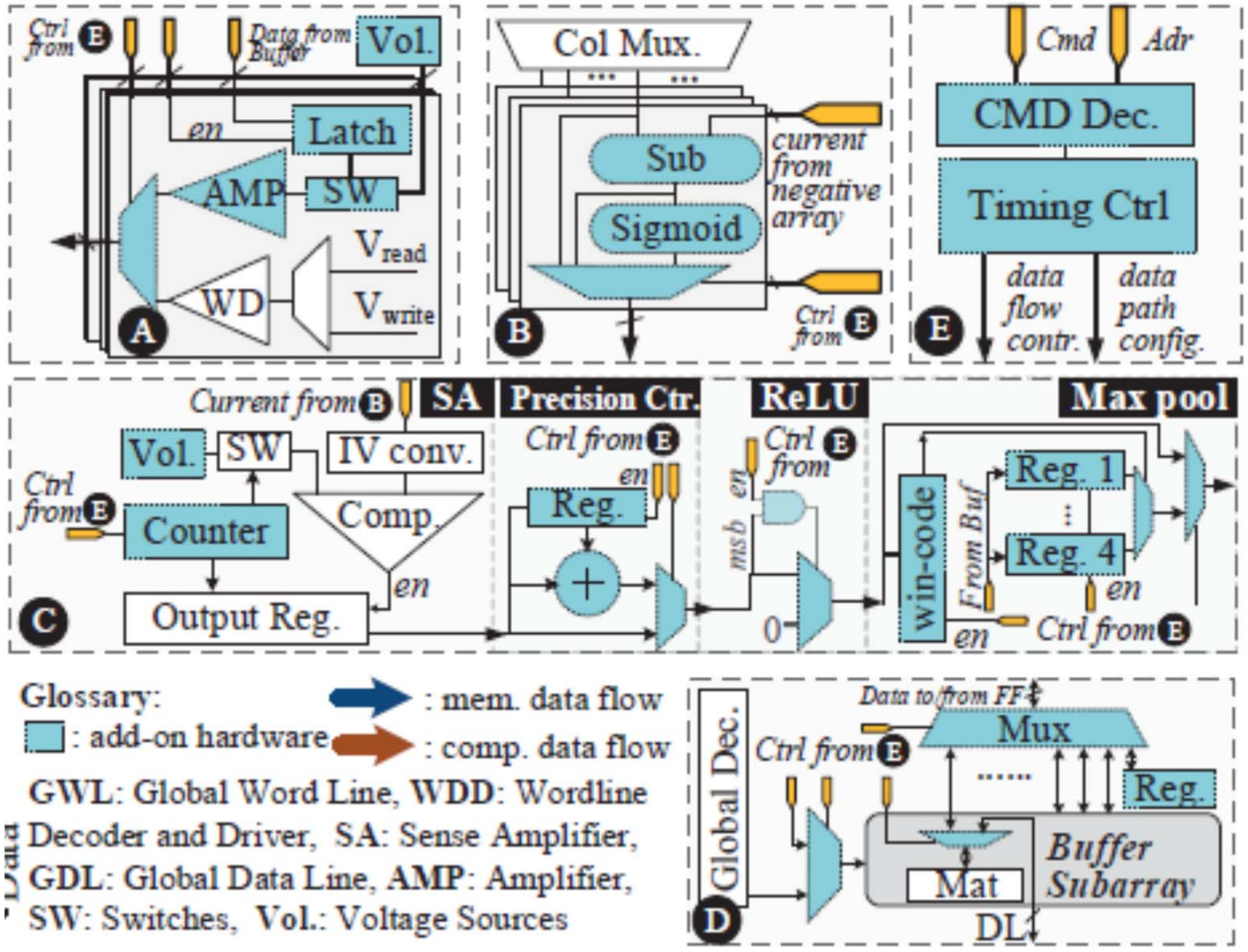
(b) using a ReRAM crossbar array for neural computation

PRIME Architecture Details



- (A) Wordline decoder and driver with multi-level voltage sources;
- (B) Column multiplexer with analog subtraction and sigmoid circuitry;
- (C) Reconfigurable SA with counters for multi-level outputs
- (D) Connection between the FF and Buffer subarrays;

Circuit-level Design Details



(A) Wordline decoder and driver with multi-level voltage sources;

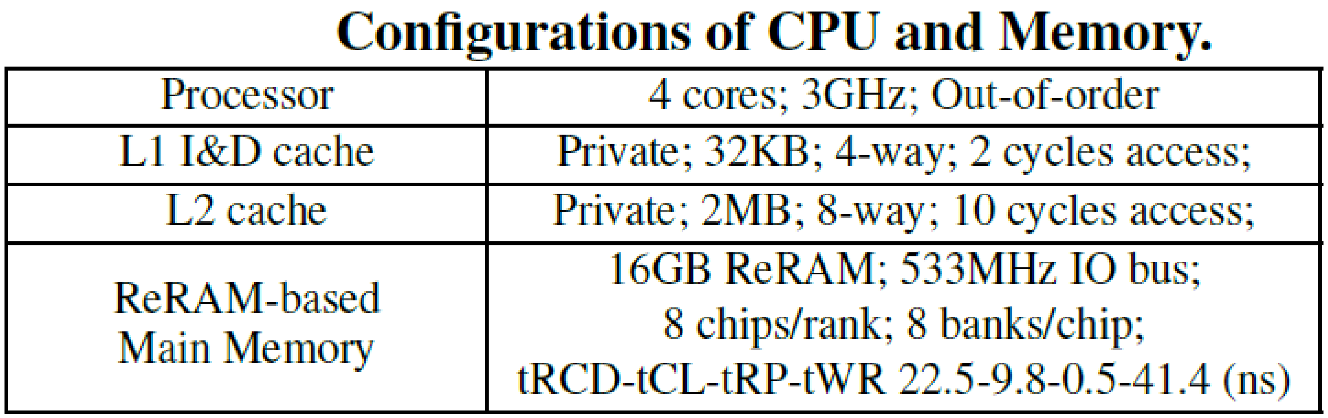
(B) Column multiplexer with analog subtraction and sigmoid circuitry;

(C) Reconfigurable SA with counters for multi-level outputs

(D) Connection between the FF and Buffer subarrays;

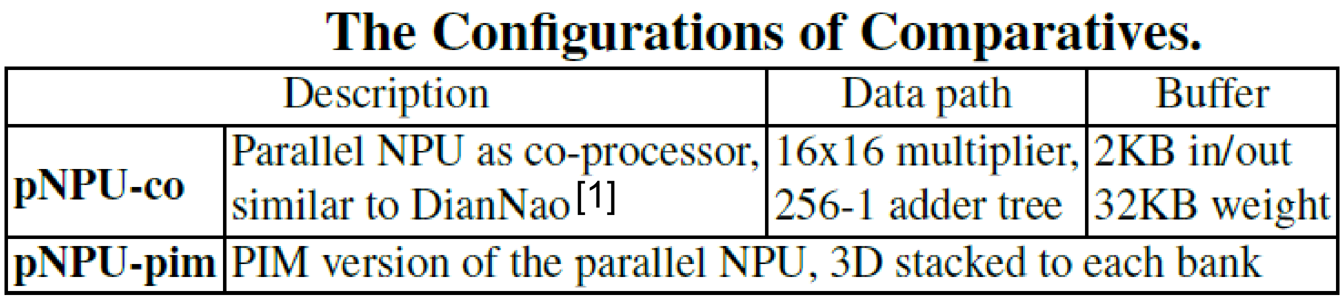
Evaluation

- Comparisons
  - Baseline CPU-only, pNPU-co, pNPU-pim



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)

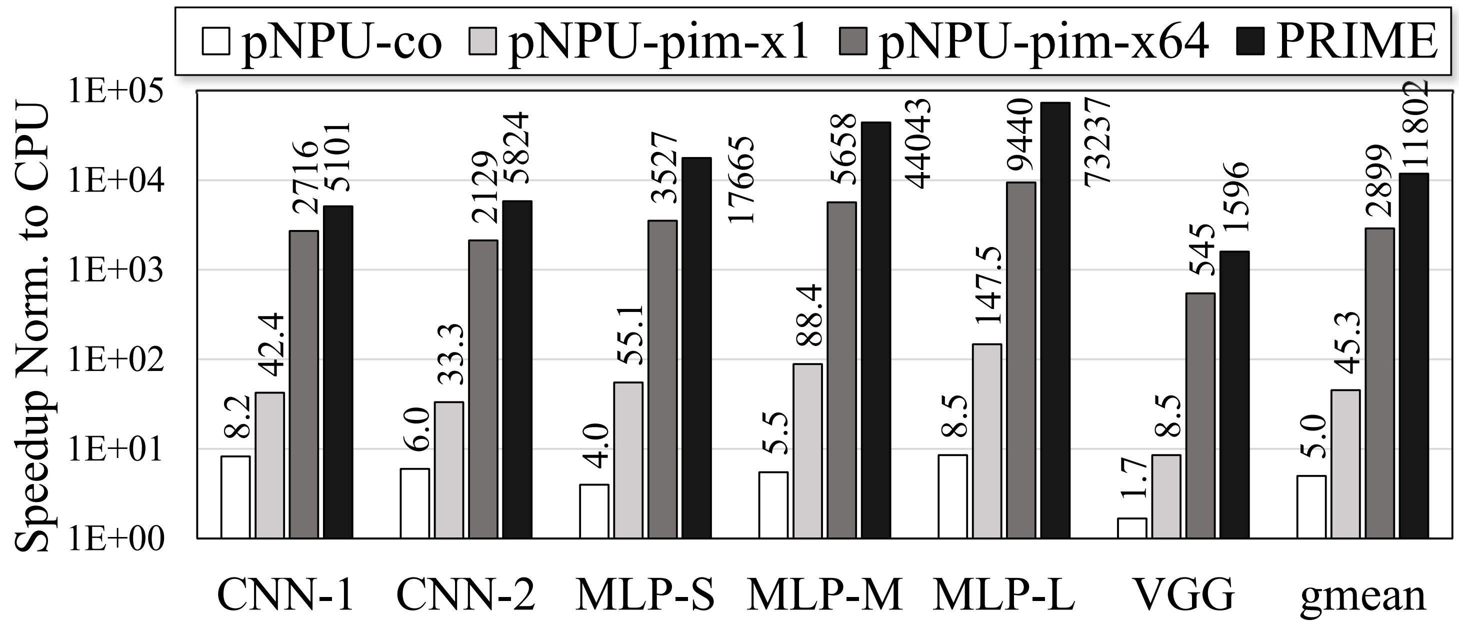


The Configurations of Comparatives.

Description		Data path	Buffer
pNPU-co	Parallel NPU as co-processor, similar to DianNao[1]	16x16 multiplier, 256-1 adder tree	2KB in/out 32KB weight
pNPU-pim	PIM version of the parallel NPU, 3D stacked to each bank		

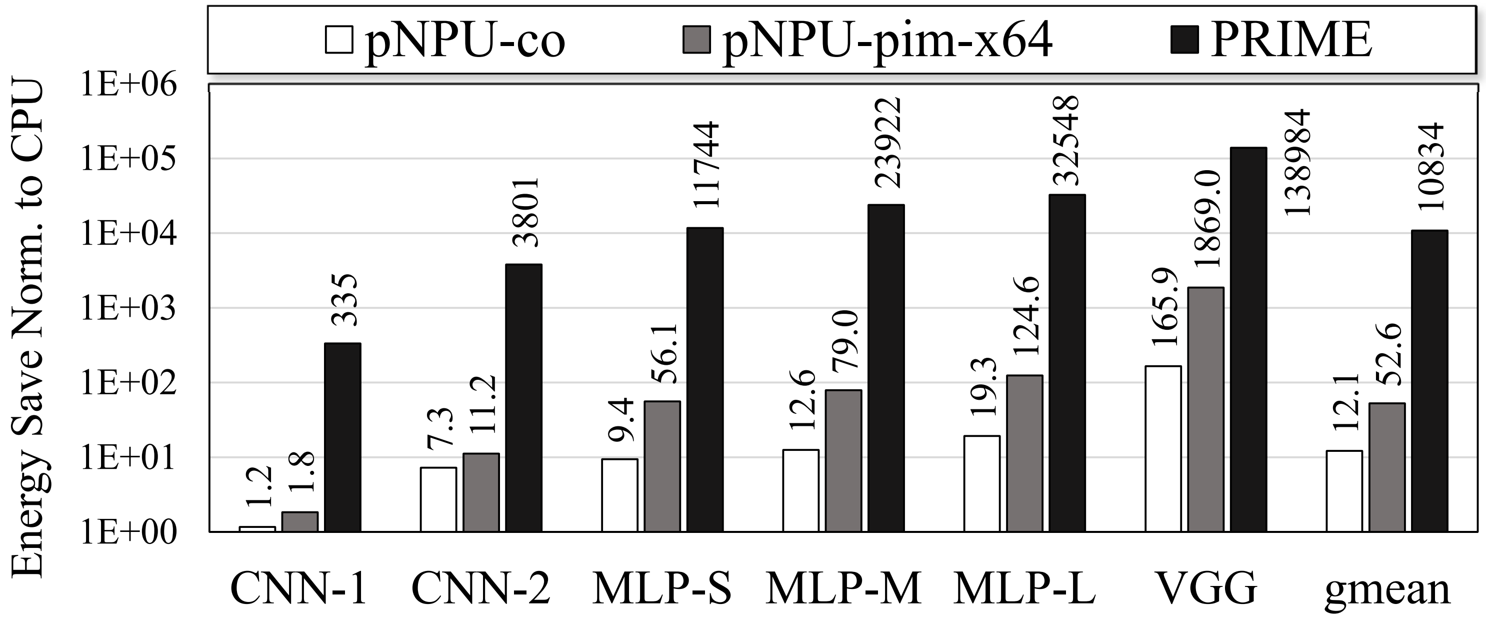
[1] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in ASPLOS'14.

Performance results



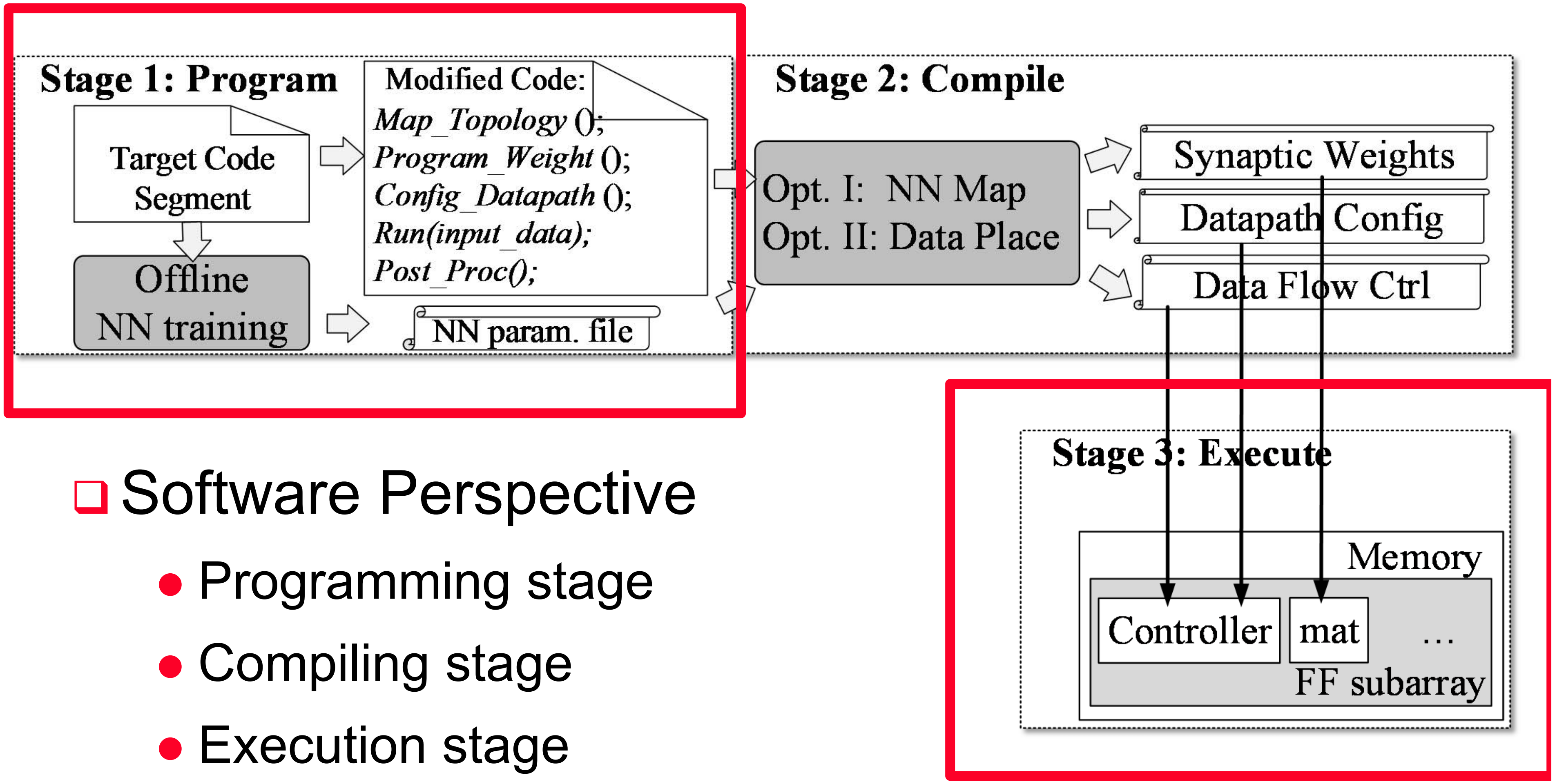
- PRIME is even 4x better than pNPU-pim-x64

Energy results



- PRIME is even 200x better than pNPU-pim-x64

System-Level Design



PIM Architecture

- Software Perspective
  - Programming stage
  - Compiling stage
  - Execution stage

Following RISC ISA design principles

Complex instructions → Short instructions

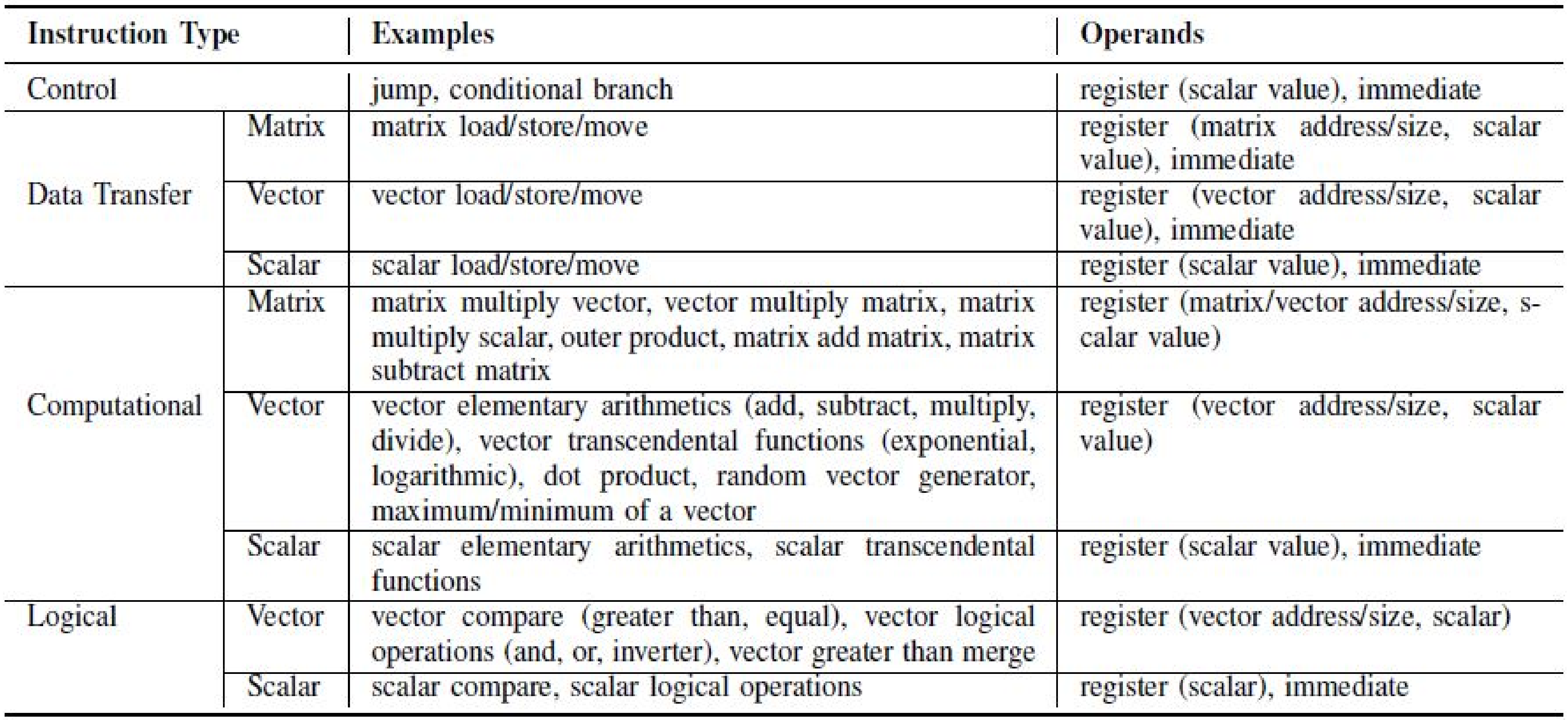
High-level functional blocks → Low-level computational operations

Full connection layer instruction → Matrix/Vector instructions

Lower overhead.

Simple and short instructions significantly reduce design/verification complexity and power/area of the instruction decoder.

An overview of NN instructions



Instruction Type		Examples	Operands
Control		jump, conditional branch	register (scalar value), immediate
Data Transfer	Matrix	matrix load/store/move	register (matrix address/size, scalar value), immediate
	Vector	vector load/store/move	register (vector address/size, scalar value), immediate
	Scalar	scalar load/store/move	register (scalar value), immediate
Computational	Matrix	matrix multiply vector, vector multiply matrix, matrix multiply scalar, outer product, matrix add matrix, matrix subtract matrix	register (matrix/vector address/size, s-calar value)
	Vector	vector elementary arithmetics (add, subtract, multiply, divide), vector transcendental functions (exponential, logarithmic), dot product, random vector generator, maximum/minimum of a vector	register (vector address/size, scalar value)
	Scalar	scalar elementary arithmetics, scalar transcendental functions	register (scalar value), immediate
Logical	Vector	vector compare (greater than, equal), vector logical operations (and, or, inverter), vector greater than merge	register (vector address/size, scalar)
	Scalar	scalar compare, scalar logical operations	register (scalar), immediate

NISA defines a total of 43 64-bit scalar/control/vector/ matrix instructions, and is sufficiently flexible to express all 10 networks.

Code Examples

MLP code:

```
    // $0: input size, $1: output size, $2: matrix size
    //  $3: input address, $4: weight address
    //  $5: bias address, $6: output address
    //  $7-$10: temp variable address

VLOAD    $3, $0, #100          // load input vector from address (100)
MLOAD    $4, $2, #300          // load weight matrix from address (300)
MMV      $7, $1, $4, $3, $0    // Wx
VAV      $8, $1, $7, $5        // tmp=Wx+b
VEXP     $9, $1, $8            // exp(tmp)
VAS      $10, $1, $9, #1       // 1+exp(tmp)
VDV      $6, $1, $9, $10       // y=exp(tmp)/(1+exp(tmp))
VSTORE   $6, $1, #200          // store output vector to address (200)
```

Code Examples

Pooling code:

```
  // $0: feature map size, $1: input data size,
  // $2: output data size, $3: pooling window size – 1
  // $4: x-axis loop num, $5: y-axis loop num
  // $6: input addr, $7: output addr
  // $8: y-axis stride of input

    VLOAD    $6, $1, #100    //  load input neurons from address (100)
    SMOVE    $5, $3          //  init y
L0: SMOVE    $4, $3          //  init x
L1: VGTM     $7, $0, $6, $7
   // ∀ feature map m, output[m]=(input[x][y][m]>output[m])?
   //                                   input[x][y][m]:output[m]
    SADD     $6, $6, $0      //  update input address
    SADD     $4, $4, #-1     //  x--
    CB       #L1, $4         //  if(x>0) goto L1
    SADD     $6, $6, $8      //  update input address
    SADD     $5, $5, #-1     //  y--
    CB       #L0, $5         //  if(y>0) goto L0
    VSTORE   $7, $2, #200    //  stroe output neurons to address (200)
```

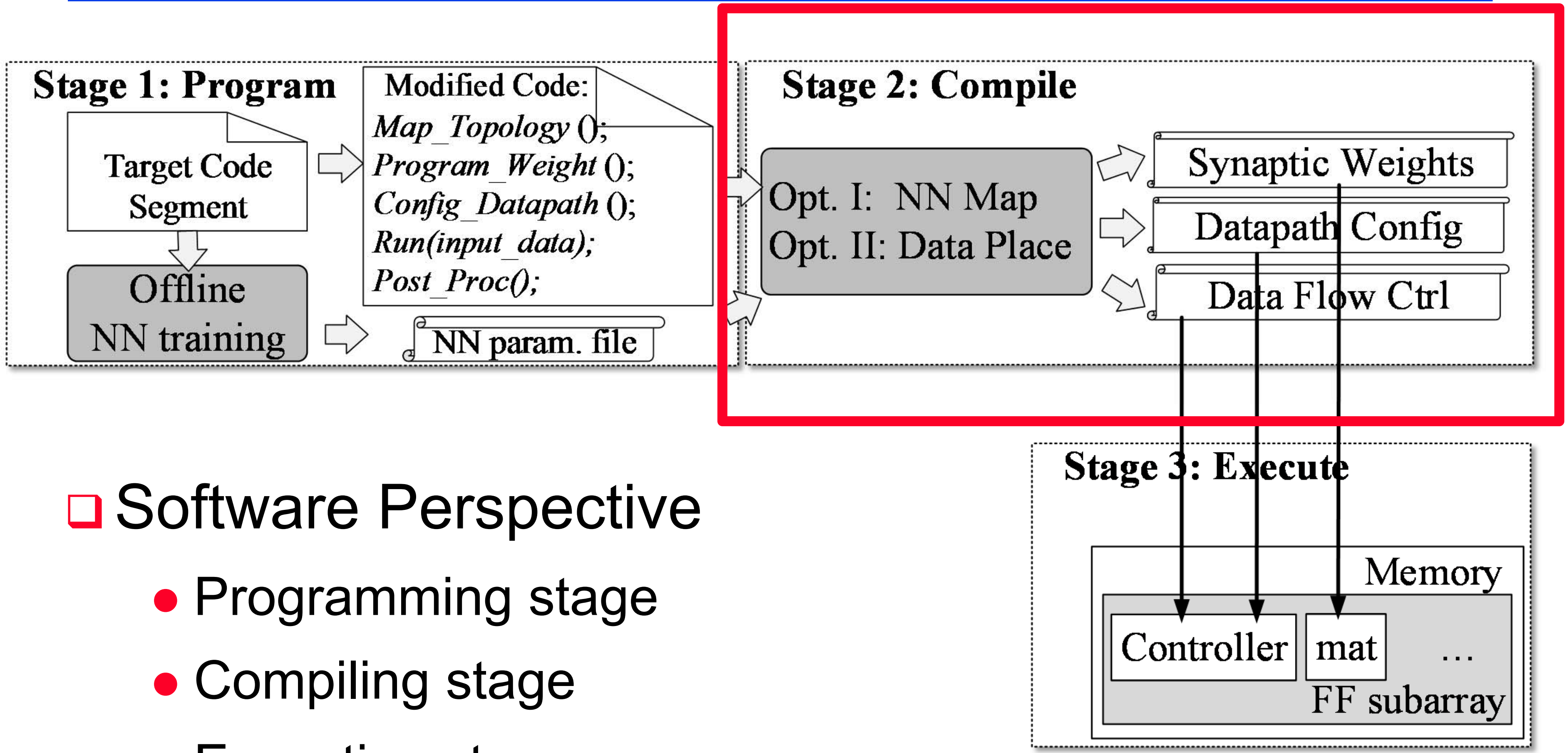
Code Examples

BM code:

```
    // $0: visible vector size, $1: hidden vector size, $2: v-h matrix (W) size
    //  $3: h-h matrix (L) size, $4: visible vector address, $5: W address
    //  $6: L address, $7: bias address, $8: hidden vector address
    //  $9-$17: temp variable address

VLOAD    $4, $0, #100        // load visible vector from address (100)
VLOAD    $9, $1, #200        // load hidden vector from address (200)
MLOAD    $5, $2, #300        // load W matrix from address (300)
MLOAD    $6, $3, #400        // load L matrix from address (400)
MMV      $10, $1, $5, $4, $0 // Wv
MMV      $11, $1, $6, $9, $1 // Lh
VAV      $12, $1, $10, $11   // Wv+Lh
VAV      $13, $1, $12, $7    // tmp=Wv+Lh+b
VEXP     $14, $1, $13        // exp(tmp)
VAS      $15, $1, $14, #1    // 1+exp(tmp)
VDV      $16, $1, $14, $15   // y=exp(tmp)/(1+exp(tmp))
RV       $17, $1             //  ∀i, r[i] = random(0,1)
VGT      $8, $1, $17, $16    //  ∀i, h[i] = (r[i]>y[i])?1:0
VSTORE   $8, $1, #500        // store hidden vector to address (500)
```

System-Level Design

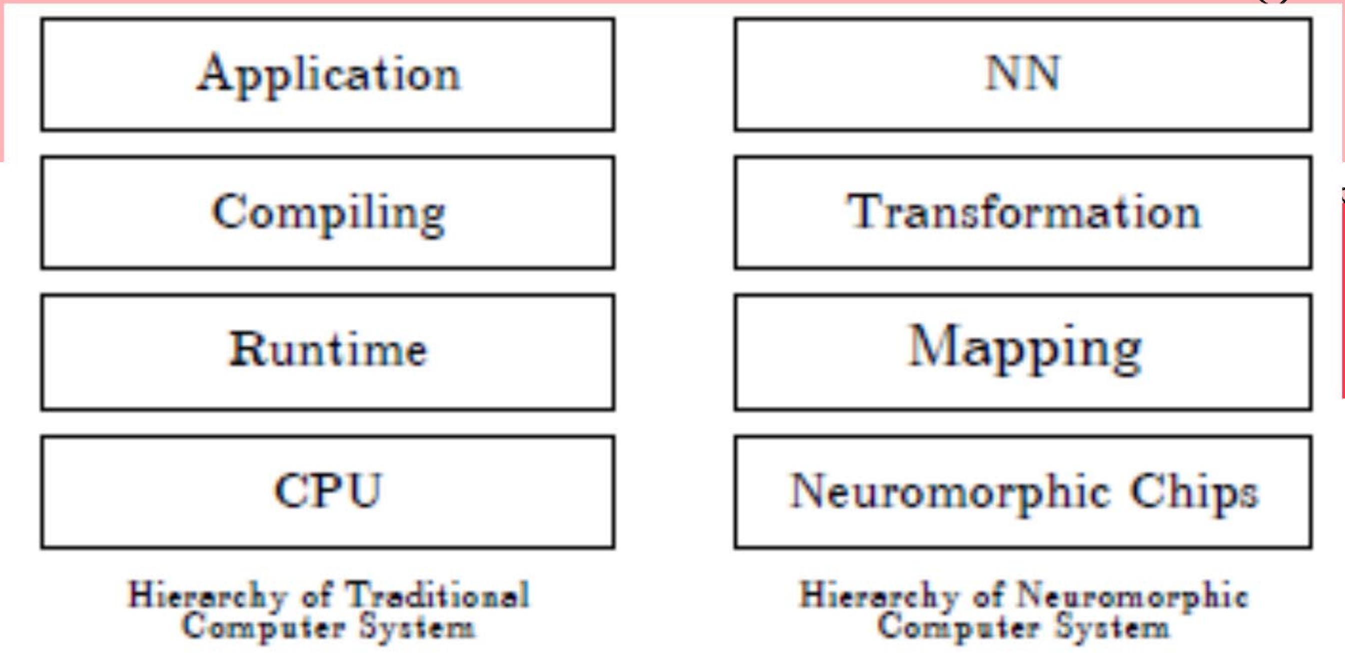


- Software Perspective
  - Programming stage
  - Compiling stage
  - Execution stage

NN Transformation

hardware-independent representation

NN transformation Transformation


Application
Compiling
Runtime
CPU
Hierarchy of Traditional Computer System
NN
Transformation
Mapping
Neuromorphic Chips
Hierarchy of Neuromorphic Computer System

More Details

- "**PRIME:** A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory", Proceedings of the 43rd International Symposium on Computer Architecture (ISCA), 2016

- “**Cambricon:** An Instruction Set Architecture for Neural Networks", in Proceedings of the 43rd ACM/IEEE International Symposium on Computer Architecture (ISCA), 2016

- “**NEUTRAMS**: Neural Network Transformation and Co-design under Neuromorphic Hardware Constraints”, to appear in Intl. Symp. On Microarchitecture (MICRO), 2016

- http://seal.ece.ucsb.edu

Conclusion

- Neuromorphic computing requires new architecture design different from conventional Von Neumann architecture
- New architecture requires a rethinking of Instruction Set Architecture Design to facilitate the software programming and hardware implementation of the new architecture
- Software toolchains are required to help the transformation of high-level NN representation to optimize the mapping of the application to the underlying architecture
- A holistic hardware-software co-design is required for the new computing paradigm.

Thank you!