NeuroSim: A Benchmark Framework of Compute-in-Memory Hardware Accelerators from Devices/Circuits to Architectures/Algorithms

Prof. Shimeng Yu
School of Electrical and Computer Engineering,
Georgia Institute of Technology

Email: shimeng.yu@ece.gatech.edu
Web: https://shimeng.ece.gatech.edu/
Hardware Accelerators for AI

- GPU still dominates the training in cloud, FPGA is good for inference for fast prototyping
- TPU (or similar digital ASIC) is ramping up in cloud as well as edge

<table>
<thead>
<tr>
<th>GPU</th>
<th>FPGA</th>
<th>TPU</th>
<th>Compute-in-memory (CIM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conventional computing platforms</td>
<td>Digital CMOS ASICs</td>
<td>Analog CMOS (or eNVMs)</td>
<td></td>
</tr>
<tr>
<td>~ 0.1 TOPS/W Floating-point</td>
<td>~ 1-10 TOPS/W Fixed-point</td>
<td>~ 10-100 TOPS/W</td>
<td></td>
</tr>
</tbody>
</table>

- To further improve energy efficiency (TOPS/W), analog CIM (possibly with eNVMs) is promising especially in the edge inference where the model is pre-trained.
- CIM chip could also support incremental learning with continuous (possibly unlabeled) new data when deployed to the field.
CIM Basics: Mixed-Signal Compute

Layer $i$  
\[\text{IN}[1] \quad \text{IN}[2] \quad \text{IN}[3] \quad \ldots \quad \text{IN}[N]\]

Layer $i+1$  
\[\text{W}_11 \quad \text{W}_12 \quad \text{W}_13 \quad \ldots \quad \text{W}_{mn}\]

\(\text{IN}[1] \xrightarrow{\text{Forward}} \text{W}_11\)
\(\text{IN}[2] \quad \text{IN}[3] \quad \ldots \quad \text{IN}[N]\)

\(\text{mapped to memory}\)

\(\text{WL Driver}\)
\(\text{SL/BL header}\)

\(\text{SL}\)\(1\)\(T\)\(1\)\(R\)
\(\text{WL}\)

\(\text{ADC}\)

\(\text{Shift-add}\)
Needs for Early Design Exploration

Versatile candidates as “analog” synaptic devices

CIM macro demonstration

Critical Needs: Evaluate system-level multi-macro performance with new synaptic devices
NeuroSim: Open-Source Simulator for CIM

Pre-RTL simulator for early-stage research

https://github.com/neurosim

MLP_NeuroSim_V3.0
Benchmark framework of synaptic device technologies for a simple neural network

DNN_NeuroSim_V2.1
Benchmark framework of compute-in-memory based accelerators for deep neural network (on-chip training chip focused)

DNN_NeuroSim_V1.3
Benchmark framework of compute-in-memory based accelerators for deep neural network (inference engine focused)

3D_NeuroSim_V1.0
Benchmark framework of 3D integrated CIM accelerators for popular DNN inference, support both monolithic and heterogeneous 3D integration
NeuroSim History

The first appearance:

Contributors:
Pai-Yu Chen, Xiaochen Peng
Shanshi Huang, Yandong Luo
Anni Lu, etc...

MLP+NeuroSim (2-layer MLP)
P.-Y. Chen, IEDM’17, P.-Y. Chen, TCAD’18

DNN+NeuroSim V1 (Inference)
X. Peng, IEDM’19

DNN+NeuroSim V2 (Training)
X. Peng, TCAD’20

Interface w/ PyTorch
Impact of NeuroSim Family

CURRENT GLOBAL USERS > 300

Industry R&D
Research Center

University

Resulted in publications 100+

Intel, tsmc, Samsung, IBM, Western Digital, SK hynix, NOVATEK, GigaDevice, ACM, IEEE, AASC, AS58, IDEM, Nature, Georgia Tech School of Electrical and Computer Engineering
DNN+NeuroSim Methodologies (Python & C++)

➢ End-to-end frameworks for: 1* inference engine [1], 2* online training system [2], 3* 3D-CIM design [3]
➢ Input parameters related to: device → circuits → architecture hierarchy and data-flow → algorithm
➢ Software-hardware co-simulation reports: inference/training accuracy, TOPS, TOPS/W and TOPS/mm²

Algorithm accuracy estimation based on WAGE method [4]
• Hardware-aware quantization for weight, activation, gradient, error, as well as partial sum quantization based on ADC precision.
• Support various network models for CIFAR-10/-100 and ImageNet

Hardware metrics estimation based on analytic models that are calibrated with SPICE at module-level.
• Massive options (tech-node 130nm to 7nm, memory device type, $R_{\text{ON}}$, ADC option...)
• Analog modules (e.g. ADC) calibrated with Cadence custom simulation;
• Digital modules estimated with standard cell area and logic gate delay/dynamic power/leakage power;
• Interconnect modules (e.g. H-tree) estimated with parasitic RC delay and power;

Area Estimation

1: Get array size
- e.g. cell size=4F*4F, synaptic array size=128*128
- Array area = (128*4F) * (128*4F)

2: Auto-define peripheries
- According to the synaptic array size 128*128
  → Define switch matrix or decoder specs (e.g. 7-bit)
- According to the wire loading cap
  → Define transmission gate (TG) size (avoid large IR drop)

Note
- Assume \( R_{TG} = R_{ON} \times \text{Drop}_{tolerance} \) (default = 0.1)
- Define TG size according to \( R_{TG} \)

3: Other user-defined peripheries
- e.g. ADC (MLSA/SAR-ADC) User define: precision, ADC-mode, ADC-mux-sharing
- Align area according to the synaptic array (w/ width or height)
Latency Estimation

1: RC delay (digital block)
- Tech-file provides temperature-dependent $I_{ON}$ & $I_{OFF} \rightarrow$ get R (based on transistor size)
- Similarly, get C (loading cap can be wire cap)

1*: Cadence fitting function (analog ADC)
- Column resistance $R_{COL}$ is calculated based on real-trace (mapped conductance and input vector)
- $R_{COL}$ as input of latency fitting function

2: Consider operation scheme
- Each block needs to consider number of operations (i.e. $T=N*t$)
- Total latency needs to be “$\Sigma$” or “MAX” according to the operation scheme (depends on working in sequence or in parallel)
Dynamic Energy Estimation

1: CV^2 dynamic energy (array)
Loading wire in array:
E += C_{WL} * V_{dd}^2 * \#WL_{Selected}
E += C_{BL} * V_{read}^2 * \#BL_{Selected}
E *= \#op

2: CV^2 dynamic energy (digital block)
Assume critical operation scheme, count number of transistors that need to be charged-up;
Sum-up CV^2 for all charged-up transistors;
Also times \#op (number of operation).

3: Cadence fitting function (analog block)
Column resistance R_{COL} is calculated based on real-trace (mapped conductance and input vector)
R_{COL} as input of power fitting function
Energy is calculated based on latency and power fitting function
DNN+NeuroSim V1 for Inference Engine

(a) DNN Setup

(b) Retention Model

(c) Network Structure

(d) NeuroSim Core

(e) Hierarchical Simulation

DNN+NeuroSim V1 for Inference Engine

Fig. 1 Architecture hierarchy of CIM accelerator (defined in DNN+NeuroSim V1.0).

Fig. 2 System-level benchmark (VGG-8) with various design options (4-bit/cell).

Sweep array size & cell precision & ADC precision on CIFAR-10 accuracy

Fig. 3 Inference accuracy with different ADC quantization: sweep subarray size and cell precision.
ADC Design Considerations

For partial sum precision larger than 4 bit, SAR ADC provides better tradeoffs than Flash ADC.

S. Yu. et al. CICC 2020
Weight Mapping and Duplication

Conventional mapping

Novel mapping

Input reuse in novel mapping

Weight duplication for speed up

X. Peng TCAS-I 2020

- IFM size = W*W*D, Kernel size = K*K*D*N, OFM size = W*W*N
- Number of sub-matrix = K*K

X. Peng TCAS-I 2020
DNN+NeuroSim V1 Reveals Key Factors in CIM

Table 1 Benchmark CIM accelerators on 8-bit VGG-8, across versatile device technologies.

<table>
<thead>
<tr>
<th>Technology node (LSTP)</th>
<th>7nm</th>
<th>22nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>8T-SRAM</td>
<td>8T-SRAM</td>
</tr>
<tr>
<td>MLSA-ADC precision</td>
<td>4-bit</td>
<td>5-bit</td>
</tr>
<tr>
<td>Cell Precision</td>
<td>1-bit</td>
<td>2-bit</td>
</tr>
<tr>
<td>Ron (Ω)</td>
<td>\</td>
<td>6k</td>
</tr>
<tr>
<td>On/Off Ratio</td>
<td>\</td>
<td>17</td>
</tr>
<tr>
<td>Inference Accuracy (%)</td>
<td>\</td>
<td>91%</td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>13.61</td>
<td>59.05</td>
</tr>
<tr>
<td>Memory Utilization (%)</td>
<td>98.73%</td>
<td>96.86%</td>
</tr>
<tr>
<td>L-by-L Latency (ms)</td>
<td>0.63</td>
<td>0.76</td>
</tr>
<tr>
<td>L-by-L Dynamic Energy (µJ)</td>
<td>22.86</td>
<td>58.56</td>
</tr>
<tr>
<td>L-by-L Leakage power (mW)</td>
<td>1.47</td>
<td>1.11</td>
</tr>
<tr>
<td>Energy Efficiency (TOPS/W)</td>
<td>51.10</td>
<td>21.36</td>
</tr>
<tr>
<td>Throughput (FPS)</td>
<td>1589.24</td>
<td>1311.87</td>
</tr>
</tbody>
</table>

- Desired $R_{ON}$: 100k ~ 1M Ω
- eNVM must be “multilevel” to beat SRAM @ advanced tech-node
- For practical eNVM (e.g. Intel RRAM), can out-perform 7nm-SRAM at edge device (with low compute activity).

Fig. 1 Energy efficiency as a function of computation vs. standby ratio of CIM accelerators (across versatile device technologies).
DNN+NeuRoSim V2 for Online Training

- Extends online training hardware and software analysis
- Non-ideal weight-update: asymmetry and non-linearity, device & cycle variation

DNN+NeuroSim V2 for Online Training

- Transposable synaptic array to support feed-forward & back-propagation (error calculation)
- Add weight gradient calculation (by SRAM-CIM), need frequent data reload (from DRAM)

Fig. 1 CIM architecture supports online training.

* SRAM-CIM for weight gradient calculation (need frequent data reload)

Fig. 2 Transposable synaptic array support both of feed-forward and back-propagation (take example of 3-terminal eNVM).
Asymmetry and Nonlinearity in Analog Synapse

- Asymmetry makes devices statistically easier to converge to the middle range of the conductance than approaching $G_{max}$ or $G_{min}$.
- Frequent sign flipping causes a large unwanted conductance change towards the middle range.
Online Training: Nonlinearity & Asymmetry

Fig. 1 Analysis of nonlinearity and asymmetry (8-bit VGG-8 for CIFAR-10), w/ and w/o momentum optimization. Results show at +5/-5, the accuracy is still >80% (w/ momentum optimization).

Mommentum optimization: \( \Delta W(t) = \beta \Delta W(t-1) + (1 - \beta) \cdot (-\frac{\partial L}{\partial W}) \)
Device & Cycle Variation Behavior Model

Device-to-device variation introduce different nonlinearity (NL).

Randomly generates NLs to different synapse with a standard deviation \((\sigma)\) respect to the mean NL \((\mu)\).

Cycle-to-cycle variation introduce variation in conductance change at every weight-update.

Express C2C variation standard deviation \((\sigma)\) in terms of percentage of entire weight range.
Online Training: Device & Cycle Variation

Fig. 1 Analysis of device-to-device variation under different non-linearity, w/ momentum optimization. Behavior model randomly generates NLs to different synapse with a standard deviation ($\sigma$) respect to the mean NL ($\mu$).

Fig. 2 Analysis of device-to-device variation under different non-linearity, w/ momentum optimization. Behavior model generate variation in G change at every programming pulse; introduce standard deviation ($\sigma$) in terms of percentage of entire weight range.
Online Training: System Performance

- **ADC (6-bit) & weight gradient units (SRAM-based)** dominant in total area
- **Buffer latency & DRAM energy** is the bottleneck of performance
- Weight gradient computation is the bottleneck in the entire learning (require frequent DRAM access)

Fig. 16 The data shows the 100th epoch of FeFET-based CIM online training accelerator. (a) area breakdown by main components; (b) latency and (c) energy breakdown by main components; (d) latency and (e) energy breakdown by operations; (f) peak latency and (g) peak energy breakdown by operations.
## DNN+NeuroSim V2 Key Insights

### VGG-8 on CIFAR10, with Novel Weight Mapping and Dataflow, on-chip training with 256 epochs

<table>
<thead>
<tr>
<th>Technology node (LSTP)</th>
<th>7 nm</th>
<th>32 nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>SRAM</td>
<td>SRAM</td>
</tr>
<tr>
<td># of Conductance States</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADC precision</td>
<td>Sequential 4-bit</td>
<td>Sequential 4-bit</td>
</tr>
<tr>
<td>Weight / ΔWeight / Cell Precision</td>
<td>5-bit / 5-bit / 1-bit</td>
<td>5-bit / 5-bit / 1-bit</td>
</tr>
<tr>
<td>Ron (n)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>On/Off Ratio</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nonlinearity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2C Variation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Pulse Voltage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write Pulse Width</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Area (mm²)</td>
<td>11.12</td>
<td>13.05</td>
</tr>
<tr>
<td>Memory Utilization (%)</td>
<td>94.62%</td>
<td></td>
</tr>
<tr>
<td>Training Accuracy (%)</td>
<td>91.00%</td>
<td></td>
</tr>
<tr>
<td>Training Latency (s) / Epoch</td>
<td>244.50</td>
<td>105.51</td>
</tr>
<tr>
<td>Training Dynamic Energy (J) / Epoch</td>
<td>195.60</td>
<td>92.70</td>
</tr>
<tr>
<td>Training Peak Latency (s) / Epoch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Peak Dynamic Energy (J) / Epoch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Throughput (TOPS)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training Energy Efficiency (TOPS/W)</td>
<td>1.74</td>
<td>2.28</td>
</tr>
<tr>
<td>Training Peak Throughput (TOPS)</td>
<td>1.04</td>
<td>4.96</td>
</tr>
<tr>
<td>Training Peak Energy Efficiency (TOPS/W)</td>
<td>13.03</td>
<td>40.80</td>
</tr>
</tbody>
</table>

### Key Insights

1. Ron still plays an important role to achieve better “peak” performance.
2. When write pulse width is long (> 1 us), the weight-update will affect the throughput (batch-size=200).
3. Desired nonlinearity: below +3/-3; cycle-to-cycle variation: <1%; to guarantee learning accuracy.
4. 7nm Parallel-SRAM shows superior performance; while FeFET is a promising synaptic device to achieve high-performance CIM training architecture.
5. DRAM access dominates the overall energy consumption, resulting in ~2 TOPS/W regardless of device technologies.
NeuroSim Validation with Real Chip Implementation

After calibration, the prediction with post-layout simulation error rate is less than 2%.

A. Lu, et al. AICAS 2021

Adjusted factors introduced to calibrate:
• transistor sizing
• wiring area
• gate switching activity
• post-layout performance drop
NeuroSim Extension Roadmap

One custom chip supports various DNN models that may have larger on-chip memory capacity than the chip could hold.

- Reconfigurable NeuroSim [p1]
- Monolithic 3D NeuroSim [p2]
- Heterogeneous 3D NeuroSim [p3]
- Cryogenic NeuroSim [p4]

Monolithic 3D partition between memory at legacy node with BEOL oxide transistors and peripheral logic at advanced 7nm node.

3D stacking with multi-tier memory interleaved with logic-tier by hybrid bonding and micro-TSV. Thermal modeling supported.

77K and 4K cryogenic transistor technology files calibrated for data-center computing and quantum peripheral control.

3D NAND based architecture for GB-level model for language, graph, genome, and recommendation system.

[p1] A. Lu et al., T-VLSI, 2021 & DATE 2020
[p2] X. Peng et al., IEDM, 2020
[p3] X. Peng et al., TED, 2021
[p4] P. Wang et al., ISCAS, 2021
[p5] W. Shim et al., EDL 2021
Reconfigurable CIM Design

If chip area is constrained, off-chip weight reloading has two options:
Option 1: reload weights (with sequential processing)
Option 2: reload inputs (with batch processing)

Go off-chip significantly lowers the throughput and energy efficiency ✖

A. Lu et al., T-VLSI, 2021 & DATE 2020
Monolithic 3D

BEOL oxide transistor

Partition the design into M3D:
- Si FEOL for logic and ADC modules at 7nm
- Oxide BEOL for memory and its periphery at 45nm

X. Peng et al., IEDM, 2020
Heterogeneous 3D

➢ TSV and hybrid bonding for 5-tiers
➢ Two Scheme: “Layer-by-Layer” vs. “Pipeline”
➢ SRAM: both logic and memory tier @ 7nm
➢ RRAM: 2-bit/cell, RON is 6kΩ (on/off=150), memory tier @ 22nm, logic tier @ 7nm
➢ TSV diameter’s sweet spot: 1~3um

X. Peng et al., TED, 2021
Cryogenic NeuroSim

Exp Data

Vth engineering

SRAM based CIM accelerator

P. Wang et al., ISCAS, 2021
3D NAND based GB model

\[ Y_j = \sum_{i=1}^{d} X_iW_{ij} \]

Hybrid bonding with 7nm logic die

Google’s MoE model

Latency = 61.7 \( \mu \)s (not change with sparsity)

Google’s MoE model

LSTM

Latency = 61.7 \( \mu \)s (not change with sparsity)

NAND wafer
CMOS wafer

Input buffer
XDEC

BL MUX / ADC
Adder&Shifter / Output buffer

Area (mm\(^2\))

Input Sparsity (%)

Energy Efficiency (TOPS/W)

WL energy ratio (%)

Latency = 61.7 \( \mu \)s (not change with sparsity)
Examples of Running DNN+NeuroSim V1.3

- **NeuroSim core: device features in param.cpp**
  
  ```
  memcelltype = 2;  // 1: cell.memCellType = Type::SRAM
                      // 2: cell.memCellType = Type::RRAM
                      // 3: cell.memCellType = Type::FeFET
  cellBit = 2;      // precision of memory device
  
  /*** parameters for SRAM ***/
  // due the scaling, suggested SRAM cell size above 22nm: 160F^2
  // SRAM cell size at 14nm: 300F^2
  // SRAM cell size at 10nm: 400F^2
  // SRAM cell size at 7nm: 600F^2
  heightInFeatureSizeSRAM = 10;    // SRAM Cell height in feature size
  widthInFeatureSizeSRAM = 28;    // SRAM Cell width in feature size
  widthSRAMCellNMOS = 2;
  widthSRAMCellPMOS = 1;
  widthAccessCMOS = 1;
  minSenseVoltage = 0.1;
  
  /*** parameters for analog synaptic devices ***/
  heightInFeatureSizeITIR = 4;    // ITIR Cell height in feature size
  widthInFeatureSizeITIR = 12;    // ITIR Cell width in feature size
  heightInFeatureSizeCrossbar = 2; // Crossbar Cell height in feature size
  widthInFeatureSizeCrossbar = 1; // Crossbar Cell width in feature size
  resistanceOn = 6e3; // Ron resistance at Vr in the reported measurement data (need to
                      //  maxConductance = (double) 1/resistanceOn;
                      //  minConductance = (double) 1/resistanceOff;
  resistanceOff = 6e3*150; // Roff resistance at Vr in the reported measurement dat (need to
  readVoltage = 0.5; // On-chip read voltage for memory cell
  readPulseWidth = 10e-9; // read pulse width in sec
  accessVoltage = 1.1;  // Gate voltage for the transistor in ITIR
  resistanceAccess = resistanceOn*IR DROP TOLERANCE; // resistance of access CMOS in ITIR
  writeVoltage = 2;     // Enable level shifter if writeVoltage > 1.8V
  ```

- **SRAM:**

- **eNVM:**
Examples of Running DNN+NeuroSim V1.3

• NeuroSim core: circuit features in param.cpp

```cpp
// technode: 130  --> wireWidth: 175
// technode: 90  --> wireWidth: 110
// technode: 65  --> wireWidth: 105
// technode: 45  --> wireWidth: 80
// technode: 32  --> wireWidth: 56
// technode: 22  --> wireWidth: 40
// technode: 14  --> wireWidth: 25
// technode: 10, 7  --> wireWidth: 10

// Technology
technode = 22;
featuresize = 40e-9;  // Wire width for subArray simulation
wireWidth = 40;       // wireWidth of the cell for Accuracy calculation
```

• Technology:

```cpp
numRowSubArray = 128;  // # of rows in single subArray
numColSubArray = 128;  // # of columns in single subArray
```

• Sub-array size:

• ADC type/sharing/resolution:

```cpp
SARADC = false;       // false: MLSA
                      // true: sar ADC
currentMode = true;   // false: MLSA use VSA
                      // true: MLSA use CSA
numColMuxed = 8;      // How many columns share 1 ADC (for eHVM and FeFET) or parallel SRAM
levelOutput = 32;     // # of levels of the multilevelSenseAmp output, should be in 2^N forms; e.g. 32 levels --> 5-bit ADC
```
Examples of Running DNN+NeuroSim V1.3

• Python wrapper that interfaces with PyTorch
  • train.py (optional): some pre-trained models provided to skip training VGG-8 for CIFAR-10, DenseNet-40 for CIFAR-10, ResNet-18 for ImageNet
  • inference.py: accuracy estimation w/ hardware effects

Model/dataset/mode

<table>
<thead>
<tr>
<th>Model/dataset/mode</th>
<th>Resolution</th>
<th>Hardware properties</th>
<th>Nonideal effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-8 for CIFAR-10</td>
<td></td>
<td></td>
<td>Variation, retention</td>
</tr>
<tr>
<td>DenseNet-40 for CIFAR-10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResNet-18 for ImageNet</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

parser.add_argument('--dataset', default='cifar10', help='cifar10/cifar100/imagenet')
parser.add_argument('--model', default='VGG', help='VGG8/DenseNet40/ResNet18')
parser.add_argument('--mode', default='WAGE', help='WAGE/FP')

parser.add_argument('--wl_weight', default=0)
parser.add_argument('--wl_grad', default=0)
parser.add_argument('--wl_activate', default=0)
parser.add_argument('--wl_error', default=0)

# Hardware Properties
# if do not consider hardware effects, set inference=0
parser.add_argument('--inference', default=0, help='run hardware inference simulation')
parser.add_argument('--subArray', default=128, help='size of subArray (e.g. 128*128)')
parser.add_argument('--ADPtodac', default=True, help='ADC precision (e.g. 5-bit)')
parser.add_argument('--cellBin', default=1, help='cell precision (e.g. 4-bit/cell)')
parser.add_argument('--convOffRatio', default=10, help='device on/off ratio (e.g. 0.05/0.95 = 3)')

# if do not run the device retention / conductance variation effects, set vari=0, v=0
parser.add_argument('--vari', default=0, help='conductance variation (e.g. 0.1 standard deviation to generate random variation)')
parser.add_argument('--v', default=0, help='drift coefficient')

parser.add_argument('--detect', default=0, help='if 1, fixed-direction drift, if 0, random drift')
parser.add_argument('--target', default=0, help='drift target for fixed-direction drift')

current_time = datetime.now().strftime("%Y_%m_%d_%H_%M_%S")
Examples of Running DNN+NeuroSim V1.3

• Optional: self-defined network models
  • Create your own file in “models” folder: Replace nn.Conv2d / nn.Linear as QConv2d / QLinear
    self.conv1 = make_layers([(C, 3, num_planes, 3, 'same', 1)], args, logger)
  • make_layers function (just call it, no need to change)
Examples of Running DNN+NeuroSim V1.3

- Optional: self-defined network models
  - Create your own NetWork.csv
  - Example: VGG-8

<table>
<thead>
<tr>
<th>Layer</th>
<th>IFM Length</th>
<th>IFM Width</th>
<th>IFM Channel Depth</th>
<th>Kernel Length</th>
<th>Kernel Width</th>
<th>Kernel Depth</th>
<th>Followed by pooling or not?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Layer 1</td>
<td>32</td>
<td>32</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>128</td>
<td>0</td>
</tr>
<tr>
<td>Layer 2</td>
<td>32</td>
<td>32</td>
<td>128</td>
<td>3</td>
<td>3</td>
<td>128</td>
<td>1</td>
</tr>
<tr>
<td>Layer 3</td>
<td>16</td>
<td>16</td>
<td>128</td>
<td>3</td>
<td>3</td>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>Layer 4</td>
<td>16</td>
<td>16</td>
<td>256</td>
<td>3</td>
<td>3</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>Layer 5</td>
<td>8</td>
<td>8</td>
<td>256</td>
<td>3</td>
<td>3</td>
<td>512</td>
<td>0</td>
</tr>
<tr>
<td>Layer 6</td>
<td>8</td>
<td>8</td>
<td>512</td>
<td>3</td>
<td>3</td>
<td>512</td>
<td>1</td>
</tr>
<tr>
<td>Layer 7</td>
<td>1</td>
<td>1</td>
<td>8192</td>
<td>1</td>
<td>1</td>
<td>1024</td>
<td>0</td>
</tr>
<tr>
<td>Layer 8</td>
<td>1</td>
<td>1</td>
<td>1024</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>
Examples of Running DNN+NeuroSim V1.3

• Workflow:
  - Change param.cpp
  - Compile c++ files
  - Create model file (optional)
  - Run train.py
  - Create NetWork.csv
  - Run inference.py

• Output example:
  - Accuracy
  - Floorplan

Test set: Average loss: 1.5665, Accuracy: 9007/10000 (90%)

Tile and PE size are optimized to maximize memory utilization (= memory mapped by synapse / total memory on chip)

Desired Conventional Mapped Tile Storage Size: 1024x1024
Desired Conventional PE Storage Size: 512x512
Desired Novel Mapped Tile Storage Size: 9x512x512
User-defined SubArray Size: 128x128

--- # of tile used for each layer ---
layer1: 1
layer2: 1
...

--- Speed-up of each layer ---
layer1: 16
layer2: 4
...

--- Utilization of each layer ---
layer1: 0.210938
layer2: 1
...
Memory Utilization of Whole Chip: 96.8584 %

--- FloorPlan Done ---
Examples of Running DNN+NeuroSim V1.3

• Output example:
  • each layer performance breakdown

<table>
<thead>
<tr>
<th>Hardware Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Estimation of Layer 1</td>
<td></td>
</tr>
<tr>
<td>layer1’s readLatency is: 377932ns</td>
<td></td>
</tr>
<tr>
<td>layer1’s readDynamicEnergy is: 1.89268e+06pJ</td>
<td></td>
</tr>
<tr>
<td>layer1’s leakagePower is: 8.47407uW</td>
<td></td>
</tr>
<tr>
<td>layer1’s leakageEnergy is: 147340pJ</td>
<td></td>
</tr>
<tr>
<td>layer1’s buffer latency is: 337150ns</td>
<td></td>
</tr>
<tr>
<td>layer1’s buffer readDynamicEnergy is: 22681.5pJ</td>
<td></td>
</tr>
<tr>
<td>layer1’s ic latency is: 24831ns</td>
<td></td>
</tr>
<tr>
<td>layer1’s ic readDynamicEnergy is: 487920pJ</td>
<td></td>
</tr>
</tbody>
</table>

-------------------------- Breakdown of Latency and Dynamic Energy --------------------------

<table>
<thead>
<tr>
<th>Hardware Performance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC (or S/As and precharger for SRAM) readLatency is: 7385.06ns</td>
<td></td>
</tr>
<tr>
<td>Accumulation Circuits (subarray level: adders, shiftAdd; PE/Tile/Global level: accumulation units) readLatency is: 7385.06ns</td>
<td></td>
</tr>
<tr>
<td>Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readLatency is: 363212ns</td>
<td></td>
</tr>
<tr>
<td>ADC (or S/As and precharger for SRAM) readDynamicEnergy is: 1.03834e+06pJ</td>
<td></td>
</tr>
<tr>
<td>Accumulation Circuits (subarray level: adders, shiftAdd; PE/Tile/Global level: accumulation units) readDynamicEnergy is: 155498pJ</td>
<td></td>
</tr>
<tr>
<td>Other Peripheries (e.g. decoders, mux, switchmatrix, buffers, IC, pooling and activation units) readDynamicEnergy is: 698849pJ</td>
<td></td>
</tr>
</tbody>
</table>

-------------------------- Breakdown of Latency and Dynamic Energy --------------------------

Latency & energy of:
- Whole layer
- Buffer
- Interconnection
- ADC
- Accumulation
- Other
- Leakage
Examples of Running DNN+NeuroSim V1.3

- Output examples:
  - System-level summary

<table>
<thead>
<tr>
<th>Component</th>
<th>Latency &amp; Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whole chip</td>
<td>Area</td>
</tr>
<tr>
<td>Buffer</td>
<td>Latency</td>
</tr>
<tr>
<td>Interconnection</td>
<td>Energy Efficiency</td>
</tr>
<tr>
<td>ADC</td>
<td>Throughput</td>
</tr>
<tr>
<td>Accumulation</td>
<td>Compute Efficiency</td>
</tr>
<tr>
<td>Other</td>
<td></td>
</tr>
<tr>
<td>Leakage</td>
<td></td>
</tr>
</tbody>
</table>

- Chip Area: 4.5354e+07um²
- Chip total CM array: 1.5408e+06um²
- Total IC area on chip (global and PE local): 0.17954e+06um²
- Total ADC (or S/A and precharger for SRAM) Area on chip: 1.6862e+06um²
- Total accumulation circuits (subarray level: adders, shiftAdd; PE/Tile/Layer: accumulation units) on chip: 3.13922e+06um²
- Other Peripherals (e.g., decoders, mux, switchmatrix, buffers, pooling and activation units): 2.00545e+07um²

- Chip clock period (t) = 2.0534ns
- Chip layer-by-layer (latency per (edge)) = 1.45613e+06ns
- Chip total readDynamicEnergy = 4.03921e+07J
- Chip total leakage energy = 1.1901e+06J
- Chip total power = 628.78mW
- Chip buffer latency = 1.0008e+09ns
- Chip buffer readDynamicEnergy = 412965J
- Chip readLatency = 1.41384ns
- Chip readDynamicEnergy = 8.54262e+06J

- **Breakdown of Latency and Dynamic Energy**

- **Breakdown of Latency and Dynamic Energy**

- **Performance**
  - Throughput TOPS (Layer-by-Layer Process): 0.845963
  - Throughput FLOPS (Layer-by-Layer Process): 886.71
  - Compute efficiency TOPS/mW² (Layer-by-Layer Process): 0.0185705

- **Simulation Performance**
  - Total run-time of NeuroSim: 151 seconds
Acknowledgement

• NeuroSim project contributors: Pai-Yu Chen, Xiaochen Peng, Shanshi Huang, Yandong Luo, Anni Lu, etc...