Machine Learning Accelerators

Eric Chen
Peicheng Tang

In-Datacenter Performance Analysis of a Tensor Processing Unit

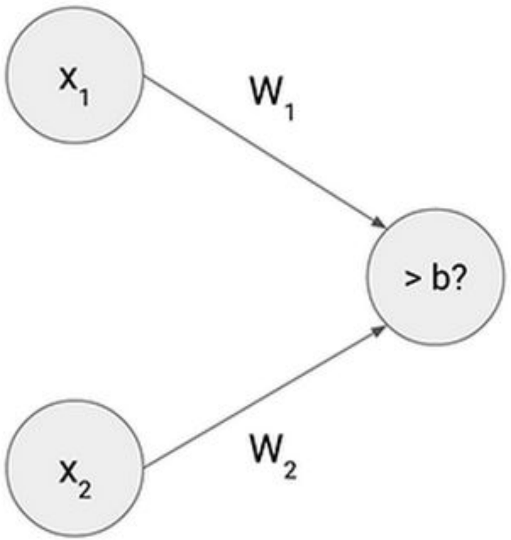
- Motivation
- Background
- TPU Overview
- Benchmarks and Platforms
- Results (Performance)
- Results (Energy)
- Takeaways

Motivation

- Rapidly increasing computation demand on Google's datacenters
- Neural networks are expensive to run on CPUs
- Solution: Develop and deploy an ASIC to accelerate NN inference

Background

- Artificial neurons
  - Nonlinear functions of the weighted sum of inputs
  - Classify data points into one of two kinds
- Performs the following calculations
  - **Multiply** the input data (x) with weights (w) to represent the signal strength
  - **Add** the results to aggregate the neuron's state into a single value
  - Apply an **activation** function (f) to modulate the artificial neuron's activity.



Content referenced from:
https://cloud.google.com/blog/products/gcp/understanding-neural-networks-with-tensorflow-playground
https://cloud.google.com/blog/products/gcp/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu

Neural Networks

- Collect neurons into layers
  - Output of one layer is input into the next
- Two Phases
  - Training
    - Use training datasets to learn the weights and bias
  - Inference
    - Running the network to perform classification
- Three common types:
  - Multi-Layer Perceptron (MLP)
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
    - LSTM is the most common RNN

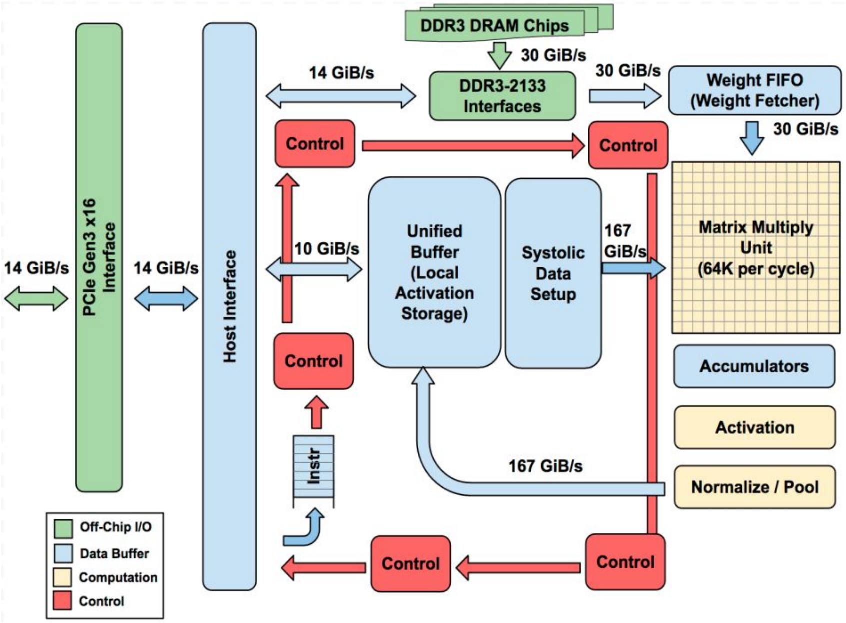
Neural Networks cont.

A very high level overview of inference tasks:

- Fetch inputs and weights
- Perform large-scale matrix multiplication
- Apply activation function on outputs
- Write data back to storage

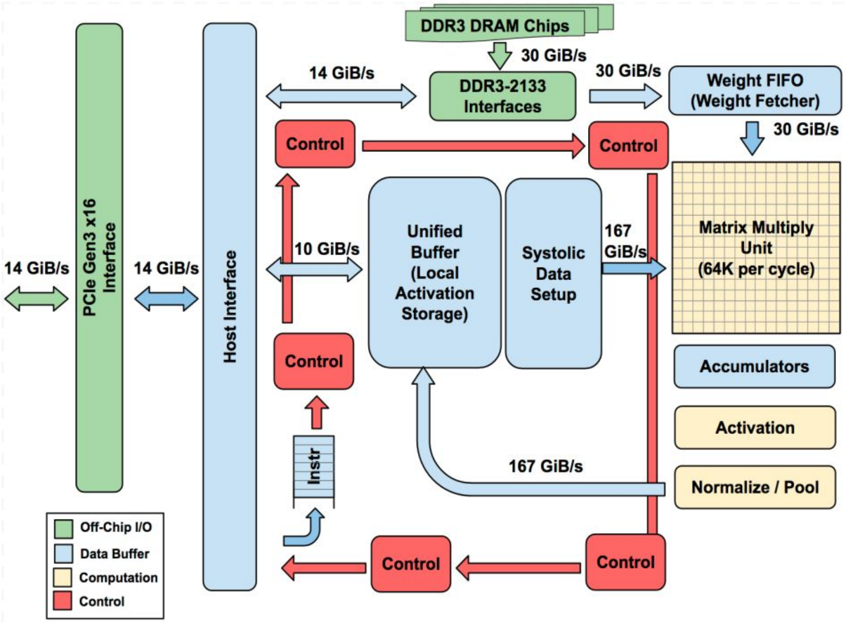
TPU Overview

- Neural Network inference accelerator
- Coprocessor on PCIe bus
- CISC-based instruction set
- Instructions sent by server, no fetching
- Primary components:
  - Matrix Multiply Unit
  - Accumulators
  - Weight Memory and FIFO
  - Activation Unit
  - Unified Buffer
- Instructions are 4-stage pipelined
  - Keep matrix unit busy
  - Hide other instructions by overlapping execution with the matrix multiply



TPU Operation

- Fetch inputs and weights
  - Input data from CPU host memory -> buffer
  - Weights from weight memory -> FIFO
- Perform large-scale matrix multiplication
  - Pass inputs and weights through systolic array, output stored in accumulator
- Apply activation function on outputs
  - Store results in unified buffer
- Write data back to storage
  - Write back from buffer to host memory



Benchmarks and Platforms

Workload

Name	*LOC*	*Layers*					*Nonlinear function*	*Weights*	*TPU Ops / Weight Byte*	*TPU Batch Size*	*% of Deployed TPUs in July 2016*
		FC	*Conv*	*Vector*	*Pool*	*Total*					
MLP0	100	5				5	ReLU	20M	200	200	61%
MLP1	1000	4				4	ReLU	5M	168	168	
LSTM0	1000	24		34		58	sigmoid, tanh	52M	64	64	29%
LSTM1	1500	37		19		56	sigmoid, tanh	34M	96	96	
CNN0	1000		16			16	ReLU	8M	2888	8	5%
CNN1	1000	4	72		13	89	ReLU	100M	1750	32	

Platforms

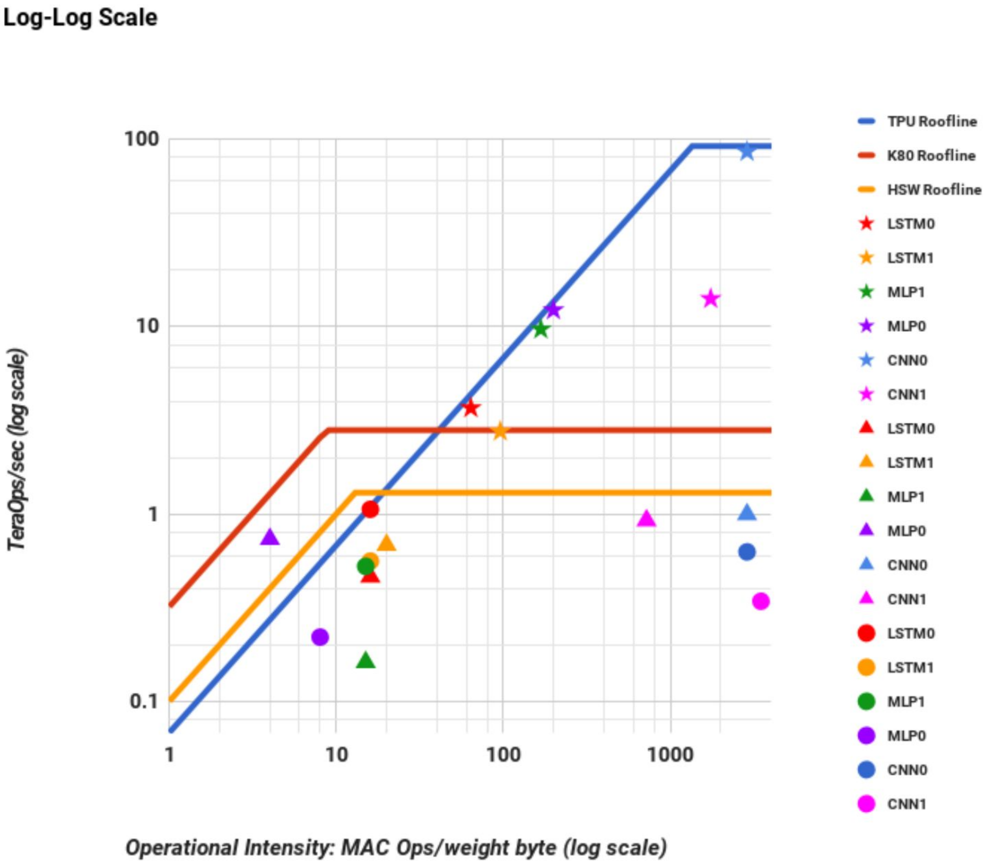
Model	*Die*										*Benchmarked Servers*				
	mm^2	*nm*	*MHz*	*TDP*	*Measured*		*TOPS/s*		*GB/s*	*On-Chip Memory*	*Dies*	*DRAM Size*	*TDP*	*Measured*	
					Idle	*Busy*	8b	FP						*Idle*	*Busy*
Haswell E5-2699 v3	662	22	2300	145W	41W	145W	2.6	1.3	51	51 MiB	2	256 GiB	504W	159W	455W
NVIDIA K80 (2 dies/card)	561	28	560	150W	25W	98W	--	2.8	160	8 MiB	8	256 GiB (host) + 12 GiB x 8	1838W	357W	991W
TPU	<331*	28	700	75W	28W	40W	92	--	34	28 MiB	4	256 GiB (host) + 8 GiB x 4	861W	290W	384W

Results (Performance)

- Gap between data points and ceiling shows potential benefits of performance tuning
- Using weighted mean of workloads, TPU is 15.3x faster than GPU

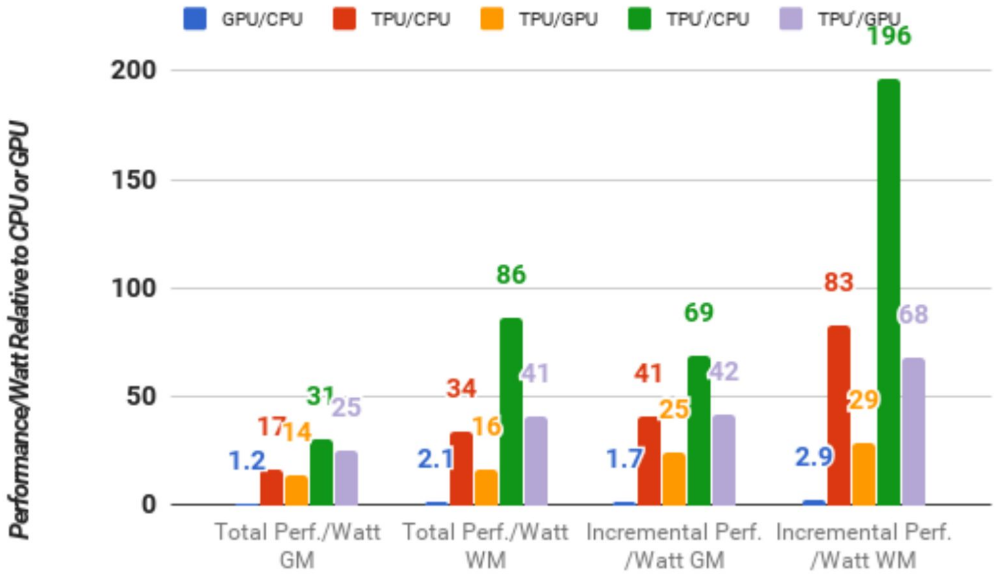
Type	DNN		LSTM		CNN		GM	WM
	0	1	0	1	0	1		
GPU	2.5	0.3	0.4	1.2	1.6	2.7	1.1	1.9
TPU	41.0	18.5	3.5	1.2	40.3	71.0	14.5	29.2
Ratio	16.7	60.0	8.0	1.0	25.4	26.3	13.2	15.3

Table 6. K80 GPU die and TPU die performance relative to CPU for the NN workloads. GM and WM are geometric and weighted mean (using the mix from Table 1). Relative performance for the GPU and TPU includes host server overhead.



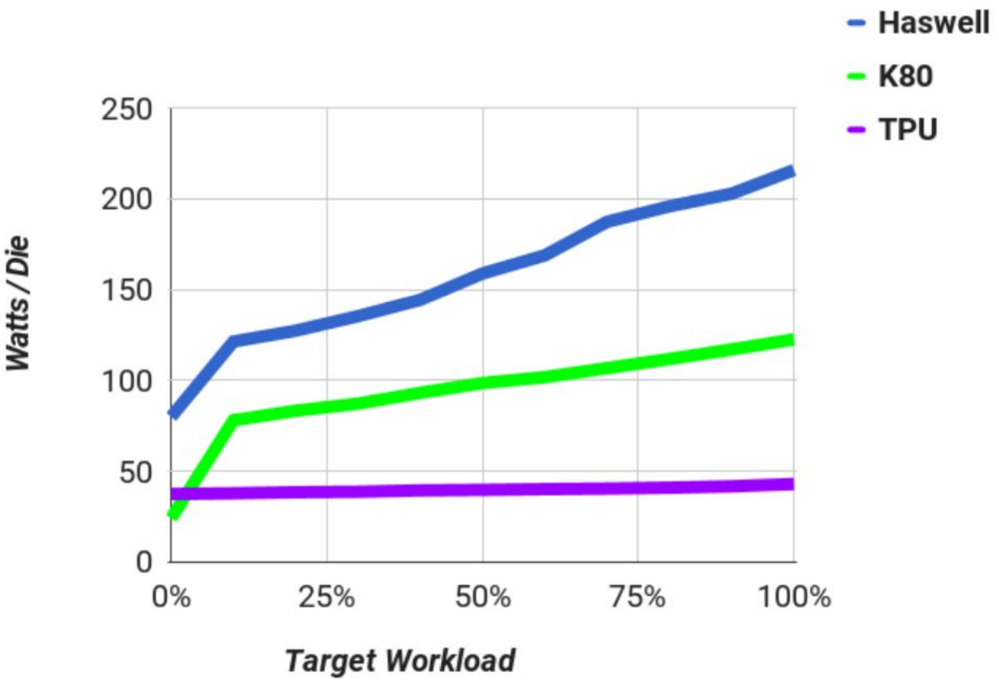
Results (energy)

- **17-34x** better **total** perf/watt over GPU
- **25-29x** better **incremental** perf/watt over GPU
    - Incremental excludes host CPU power consumption



Energy Proportionality

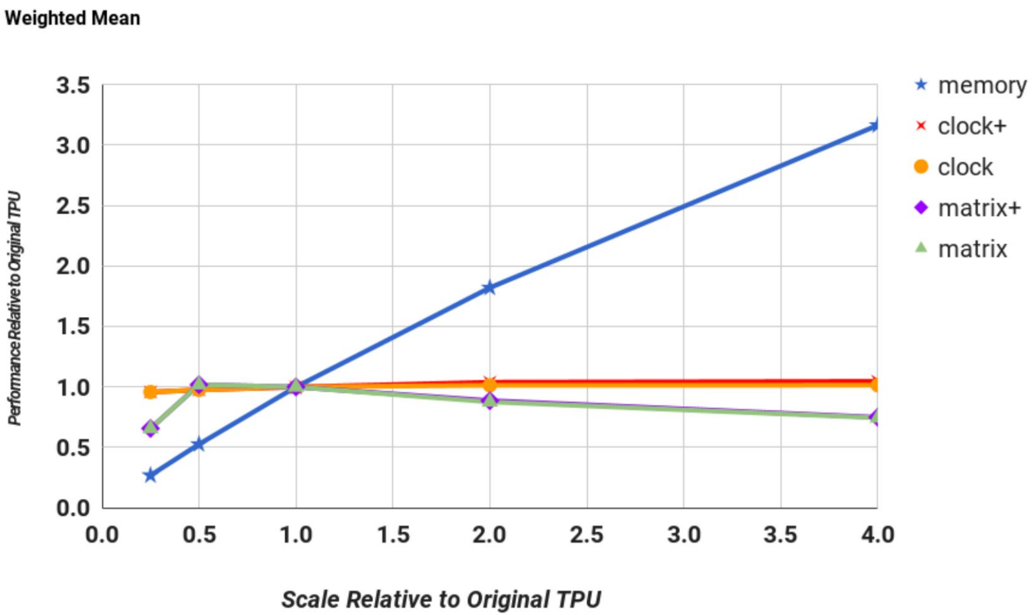
- Servers are not always busy - ideally power should be proportional to workload
- Graph normalized per die, server has 2 CPUs and either 8 GPUs or 4 TPUs



Takeaways

- Memory bandwidth has the greatest impact on perf.
  - 4/6 applications were memory bound
- CNNs are common on edge devices, but MLPs and LSTMs make up the bulk of datacenter workload
- Inferences per second is a poor metric
- History is important for designing domain-specific architectures

Performance Scaled w/ parameters



DaDianNao: A Machine-Learning Supercomputer

- Motivation
- Main Contribution
- Implementation details
- Evaluation

Motivation

- Neural Network has the trend to have larger size
  - Increasing number of parameters
  - 1 billion parameters(64bits/each) = 8GB
- Existing accelerators have ***size*** limitations
  - Only small neural network can be executed
  - Intermediate data (learned parameters, synapses) stored in main memory
- Improve DianNao?

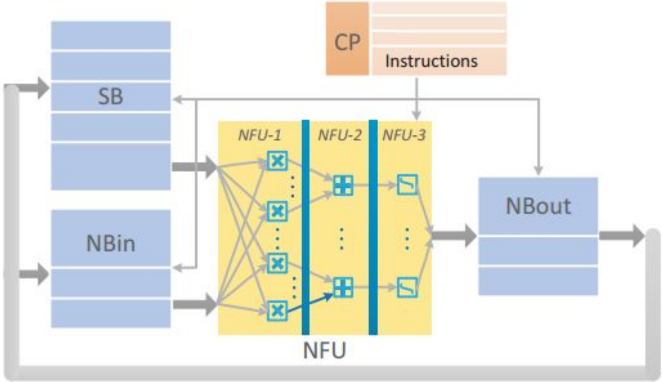


Figure 3: *Block diagram of the DianNao accelerator [5].*

Main problem:

Memory bandwidth/storage

Contribution

DaDianNao----A multi-chip system that maps memory footprint to on-chip storage

1. Synapses are stored close to the neuron
2. Asymmetric architecture where each node footprint is massively biased towards storage rather than computations
3. Transfer neuron results instead of synapses (Low external bandwidth needed)
4. Break down local storage into tiles (High internal bandwidth)

Implementation Detail----Node

1. Synapses Close to Neurons
   a. Both inference and training
   b. Low energy/latency data transfers
   c. Use eDRAM to store data
   d. Split eDRAM into four banks

2. High Internal Bandwidth
   a. Tiled based design
   b. Tiles connected via fat tree

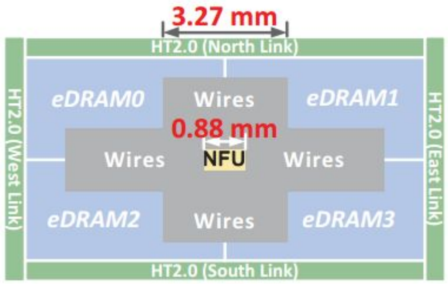


Figure 4: *Simplified floorplan with a single central NFU showing wire congestion.*

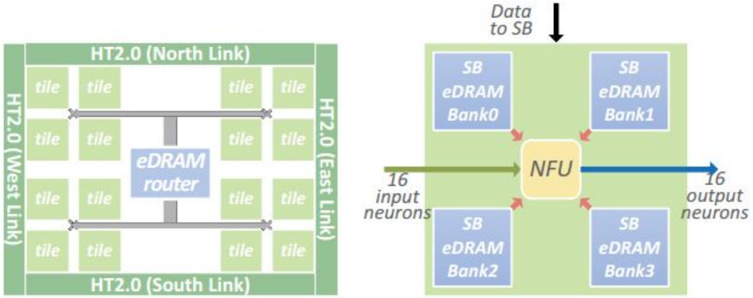


Figure 5: *Tile-based organization of a node (left) and tile architecture (right). A node contains 16 tiles, two central eDRAM banks and fat tree interconnect; a tile has an NFU, four eDRAM banks and input/output interfaces to/from the central eDRAM banks.*

Implementation Detail----Node

3. Configurability (Layers, Inference vs. Training)
   - Pipeline configuration
   - Block: Aggregation of 16-bit operators
     i. 16 bits work most of time, but fail in training

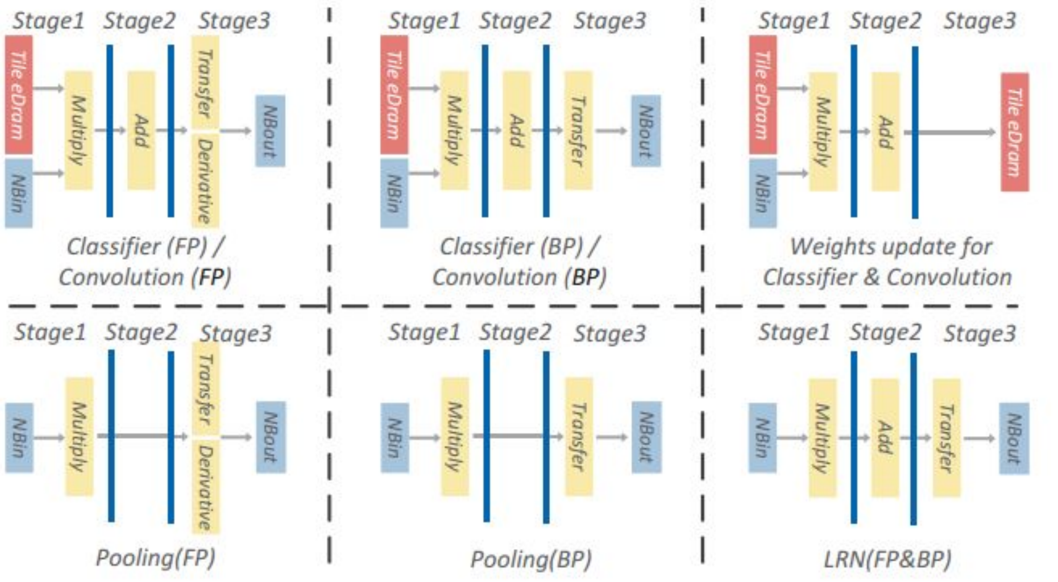


Figure 7: *Different pipeline configurations for CONV, LRN, POOL and CLASS layers.*

Implementation Detail----Overall Characteristics

Parameters	Settings	Parameters	Settings
Frequency	606MHz	tile eDRAM latency	$\sim$3 cycles
# of tiles	16	central eDRAM size	4MB
# of 16-bit multipliers/tile	256+32	central eDRAM latency	$\sim$10 cycles
# of 16-bit adders/tile	256+32	Link bandwidth	6.4x4GB/s
tile eDRAM size/tile	2MB	Link latency	80ns

Table III: *Architecture characteristics.*

Implementation Detail
Programming, Code Generation

1. Programming, Control and Code Generation

Class	Class	Class	Class	CP
LOAD	LOAD	LOAD	LOAD	central eDRAM
STORE	WRITE	WRITE	WRITE	
192	128	64	0	
256	256	256	256	
0	0	0	0	
4	4	4	4	
64	64	64	64	
4	4	4	4	
READ	READ	READ	READ	SB
NULL	NULL	NULL	NULL	
768	512	256	0	
1	1	1	1	
READ	READ	READ	READ	NBin
NULL	NULL	NULL	NULL	
0	0	0	0	
1	1	1	1	
READ	READ	READ	NULL	NBout
WRITE	WRITE	WRITE	WRITE	
0	0	0	0	
1	1	1	1	
MUL	MUL	MUL	MUL	NFU
ADD	ADD	ADD	ADD	
SIGMOID	NULL	NULL	NULL	
1	1	1	0	
0	1	1	1	

Table V: *An example of classifier code (N_i = 4096, N_o = 4096, 4 nodes).*

Inst Name	CP
READ OP	central eDRAM
WRITE OP	
READ ADDR	
WRITE ADDR	
READ STRIDE	
WRITE STRIDE	
READ ITER	
WRITE ITER	
READ OP	SB
WRITE OP	
ADDR	
STRIDE	
READ OP	NBin
WRITE OP	
ADDR	
STRIDE	
READ OP	NBout
WRITE OP	
ADDR	
STRIDE	
NFU-1 OP	NFU
NFU-2 OP	
NFU-3 OP	
NFU-2-IN	
NFU-2-OUT	

Table IV: *Node instruction format.*

Implementation Detail
Multi-Node Mapping

1. Multi-Node Mapping
   a. Convolutional and pooling layers
   b. Local response normalization layers
   c. Classifier layers

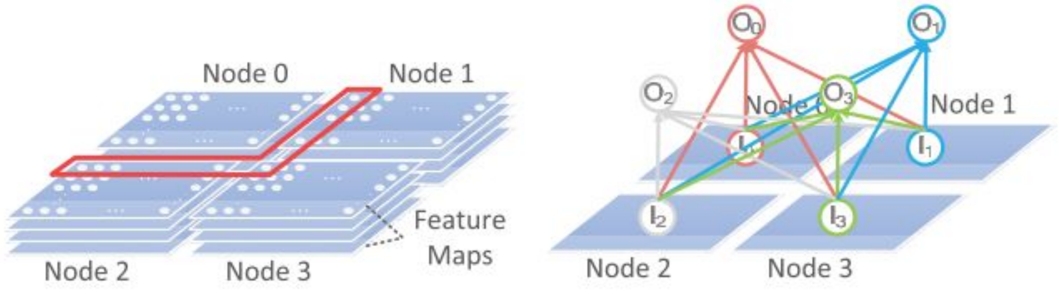


Figure 8: *Mapping of (left) a convolutional (or pooling) layer with 4 feature maps; the red section indicates the input neurons used by node 0; (right) a classifier layer.*

Evaluation - Performance

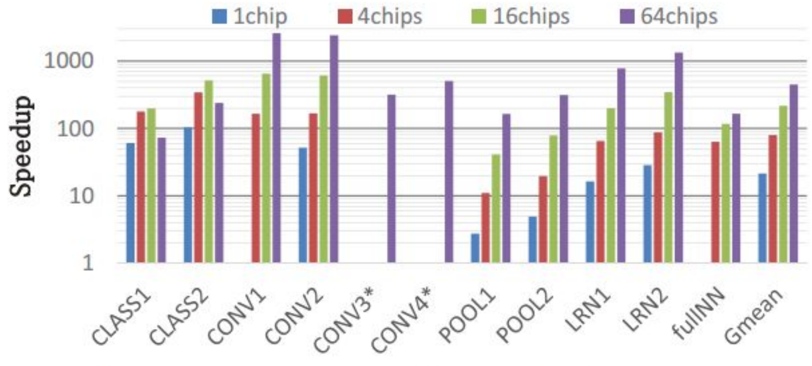


Figure 10: *Speedup w.r.t. the GPU baseline (inference). Note that CONV1 and the full NN need a 4-node system, while CONV3* and CONV4* even need a 36-node system.*

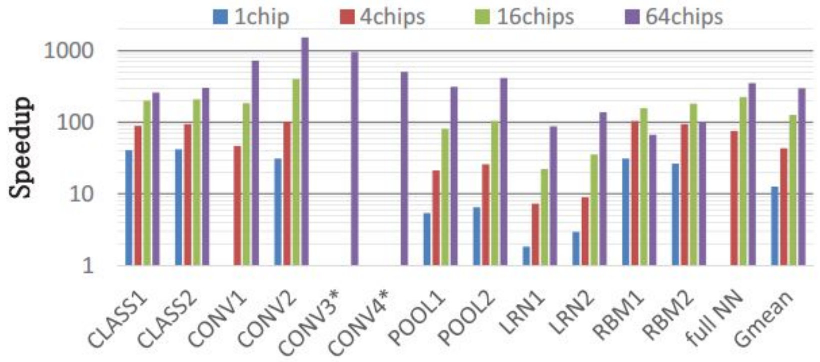


Figure 12: *Speedup w.r.t. the GPU baseline (training).*

With 64 nodes:
Inference: outperforms a single GPU by up to 450.65x
Training: 300.04x

Evaluation - Power

Component/Block	Area (μm^2)	(%)	Power (W)	(%)
WHOLE CHIP	67,732,900		15.97	
Central Block	7,898,081	(11.66%)	1.80	(11.27%)
Tiles	30,161,968	(44.53%)	6.15	(38.53%)
HTs	17,620,440	(26.02%)	8.01	(50.14%)
Wires	6,078,608	(8.97%)	0.01	(0.06%)
Other	5,973,803	(8.82%)		
Combinational	3,979,345	(5.88%)	6.06	(37.97%)
Memory	32207390	(47.55%)	6.12	(38.30%)
Registers	3,348,677	(4.94%)	3.07	(19.25%)
Clock network	586323	(0.87%)	0.71	(4.48%)
Filler cell	27,611,165	(40.76%)		

Table VI: *Node layout characteristics.*

With 64 nodes:
Inference: reduce energy by up to 150.31x
Training: 66.94x

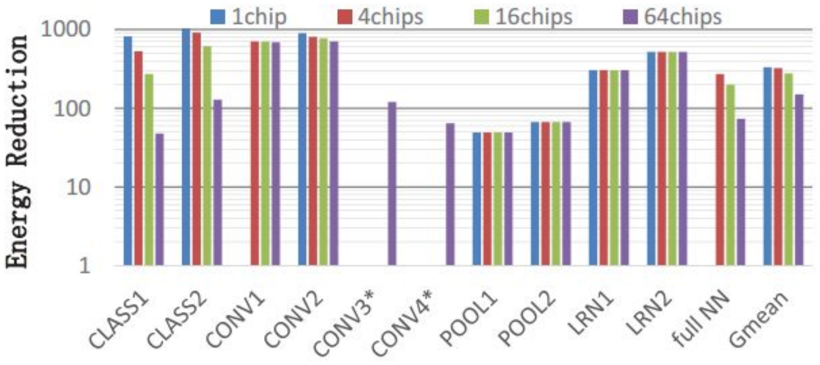


Figure 13: *Energy reduction w.r.t. the GPU baseline (inference).*

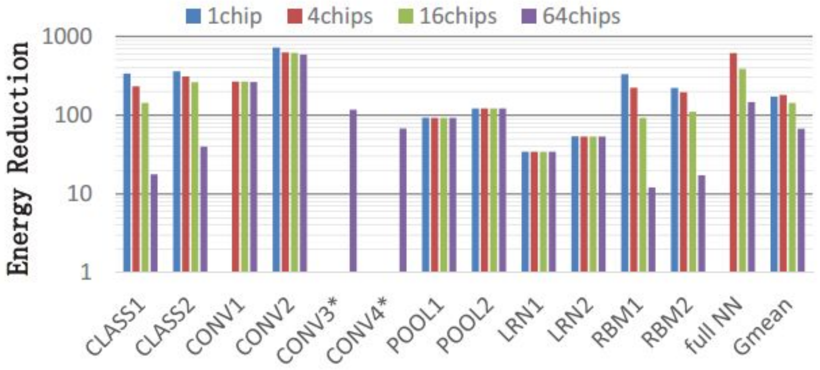


Figure 14: *Energy reduction w.r.t. the GPU baseline (training).*