ECE 598NSG/498NSU
Deep Learning in Hardware
Fall 2020

DNN Architectures – Accelerators II: DianNao, DNN Engine, and PSP Case Studies

Naresh Shanbhag

Department of Electrical and Computer Engineering

University of Illinois at Urbana-Champaign

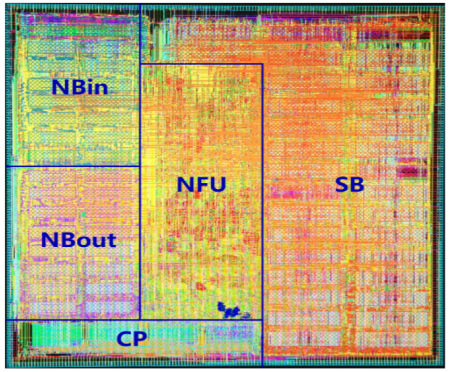
http://shanbhag.ece.uiuc.edu

Outline

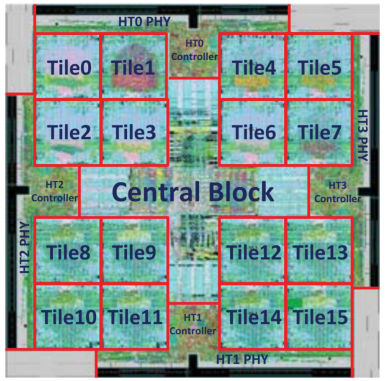
- DianNao Case Study
- DNN Engine Case Study
- PSP Case study

DianNao

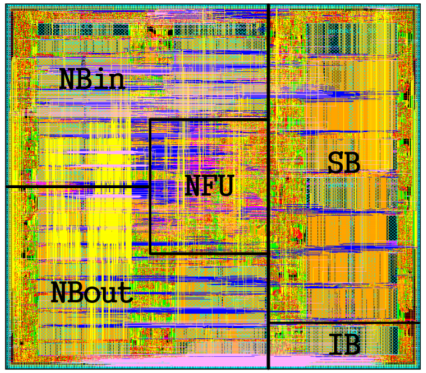
DianNao Family



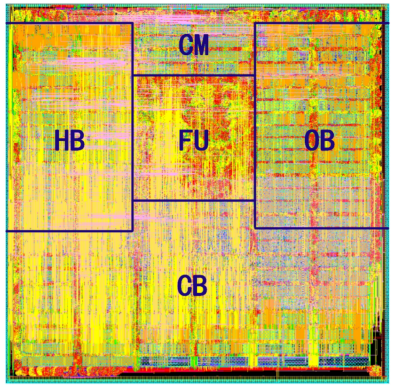
DianNao [ASPLOS'14]



DaDianNao [MICRO'14]



ShiDianNao [ISCA'15]



PuDianNao [ASPLOS'15]

Table 1. Accelerators in the DianNao family.

Name	Process (nm)	Peak performance (GOP/s)	Peak power (W)	Area (mm²)	Applications
DianNao	65	452	0.485	3.02	Neural networks
DaDianNao	28	5585	15.97	67.73	Neural networks
ShiDianNao	65	194	0.32	4.86	Convolutional neural networks
PuDianNao	65	1056	0.596	3.51	Seven representative machine learning techniques

- Family of machine learning accelerators
- Synthesized and not fabricated!

DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning

Tianshi Chen
SKLCA, ICT, China

Zidong Du
SKLCA, ICT, China

Ninghui Sun
SKLCA, ICT, China

Jia Wang
SKLCA, ICT, China

Chengyong Wu
SKLCA, ICT, China

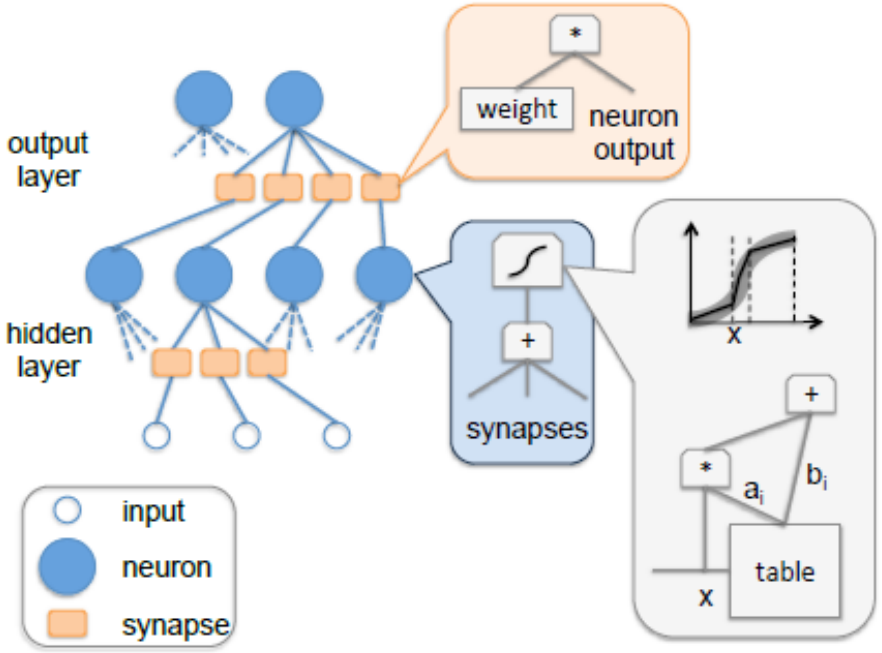
Yunji Chen
SKLCA, ICT, China

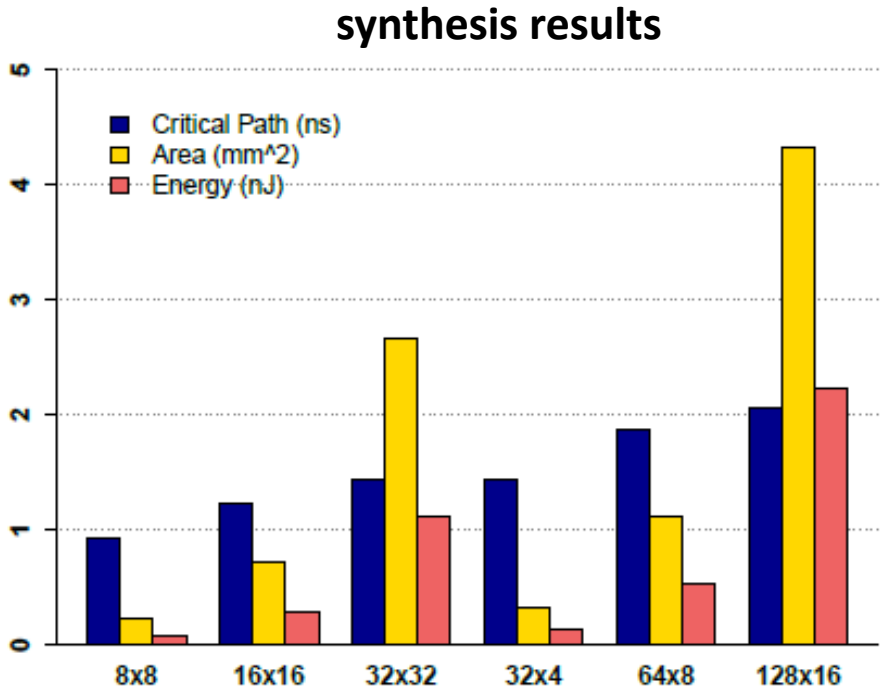
Olivier Temam
Inria, France

Featured in ASPLOS'14!

- Focuses on optimizing **memory usage** for CNN & DNNs
  - one of the earliest works to talk about reuse
- Designed for **MLP** and **CNN inference** only
- Consists of a neural functional unit (NFU) with 256 PEs + adder trees
- Uses 16b **fixed-point** operations

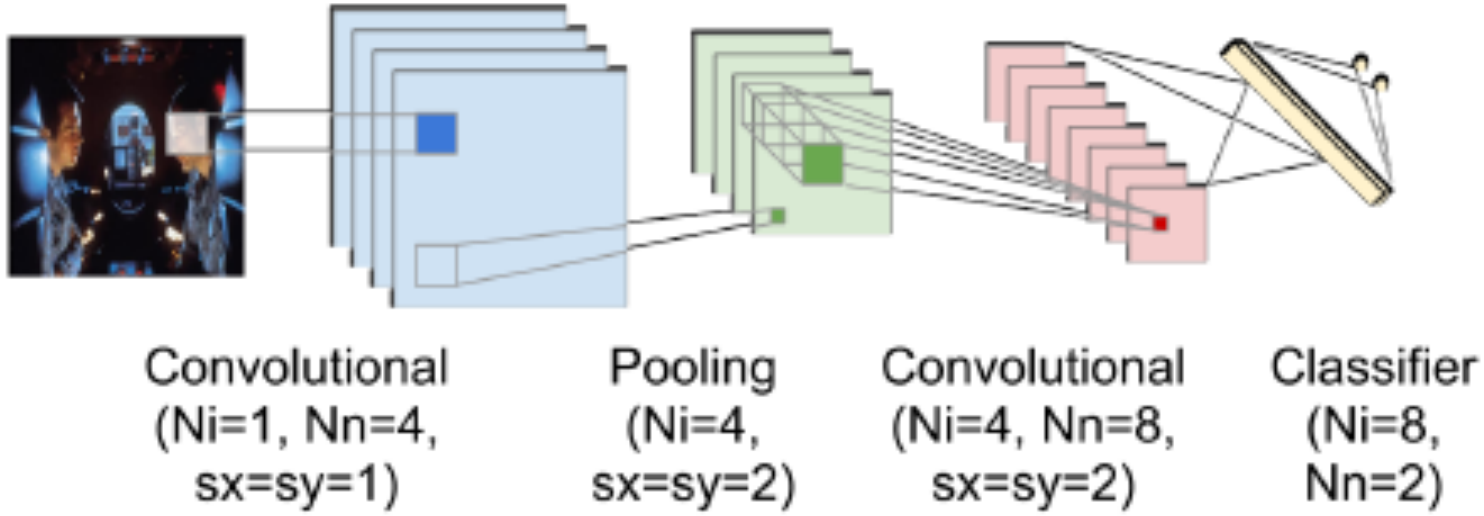
Realizing Small NNs





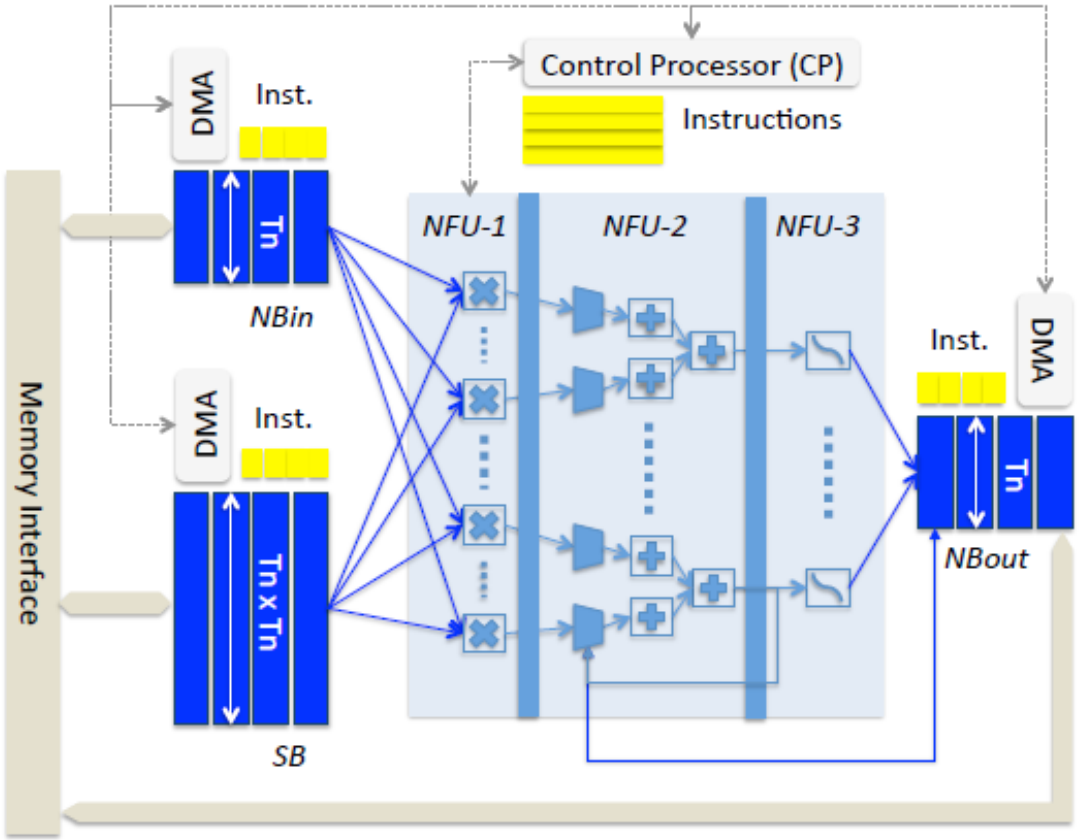
- Direct mapped architecture is not scalable → area, energy, delay increases quadratically with the number of neurons
- Recommends time-shared (**folded**) architecture with proper memory management

Change of Terminology



- Neuron: activation; synapse: kernel weight
- N_i: number of input features; N_n: number of output features
- Conv layers: $K_x \times K_y \times N_i$ kernels
- s_x, s_y: filter stride

DianNao Architecture



- Three memory buffers for **inputs** (NBin), **outputs** (NBout), **synapses** (SB)
- 3-stage (staggered) pipelined Neural Functional Unit (NFUs)
- Convolutional and FC use all three stages; pooling uses NFU-2

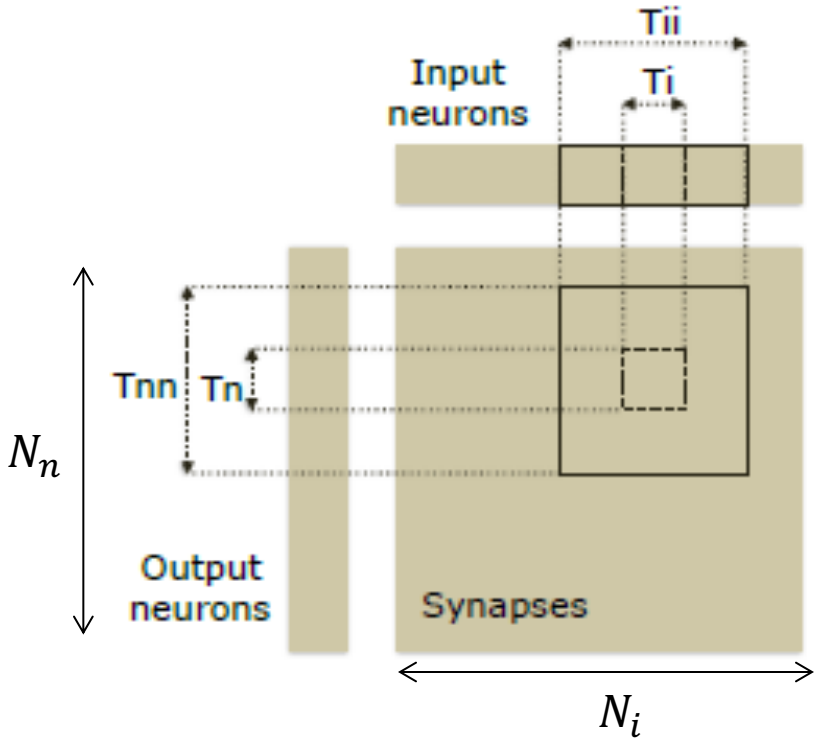
Original FC Layer Code

```
for (int n = 0; n < Nn; n++)
    sum[n] = 0;
for (int n = 0; n < Nn; n++) // output neurons
    for (int i = 0; i < Ni; i++) // input neurons
        sum[n] += synapse[n][i] * neuron[i];
for (int n = 0; n < Nn; n++)
    neuron[n] = sigmoid(sum[n]);
```

MAC operation

- Processes N_n neurons with N_i synapses every cycle
- Total memory transfers → $N_i N_n + N_i N_n + N_n$
  - inputs loaded + synapses loaded + outputs written
- Input neurons **reused** → too large to fit in local buffers
  - solution: use tiling

Tiled Code



- Assume architecture can process $T_n < N_n$ neurons with $T_i < N_i$ synapses simultaneously
- T_{ii}: input tile factor
- T_{nn}: output tile factor
- Trade-off between input and output **reuse**

```
for (int nnn = 0; nnn < Nn; nnn += Tnn){  // tiling for output neurons
    for (int iii = 0; iii < Ni; iii += Tii){  // tiling for input neurons
        for (int nn = nnn; nn < nnn + Tnn; nn += Tn){
            for (int n = nn; n < nn + Tn; n++)
                sum[n] = 0;
            for (int ii = iii; ii < iii + Tii; ii += Ti)
                for (int n = nn; n < nn + Tn; n++)
                    for (int i = ii; i < ii + Ti; i++)
                        sum[n] += synapse[n][i] * neuron[i];
            for (int n = nn; n < nn + Tn; n++)
                neuron[n] = sigmoid(sum[n]);
}   }   }
```

T_n neurons

T_i synapses

Impact of Tiling on Memory BW

- Reduces **input** BW dramatically → slight increase in **output** BW
- Use local storage (L2 cache) to **buffer** synapses (weights)
  - no weight reuse within perceptron layer
  - weights reused for each new input data

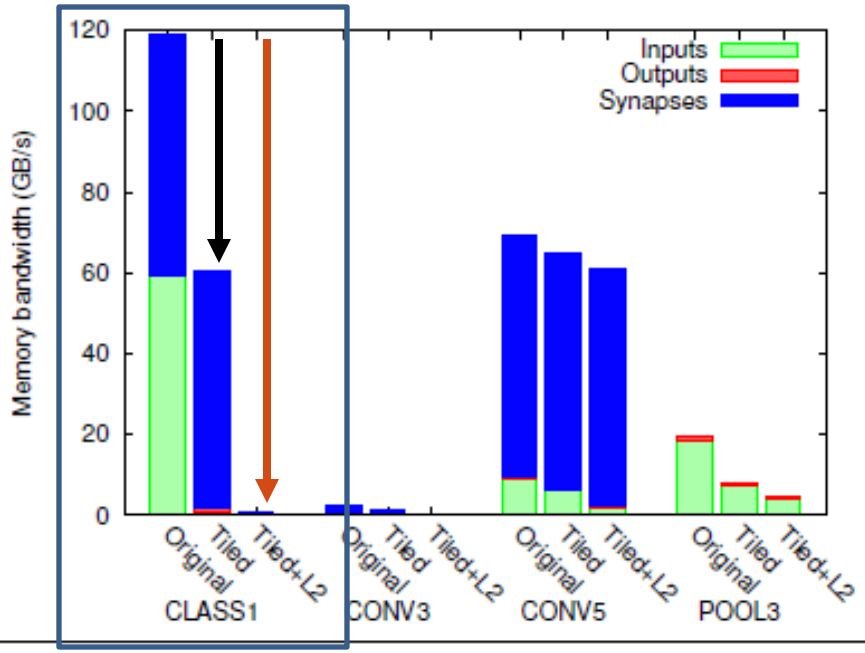
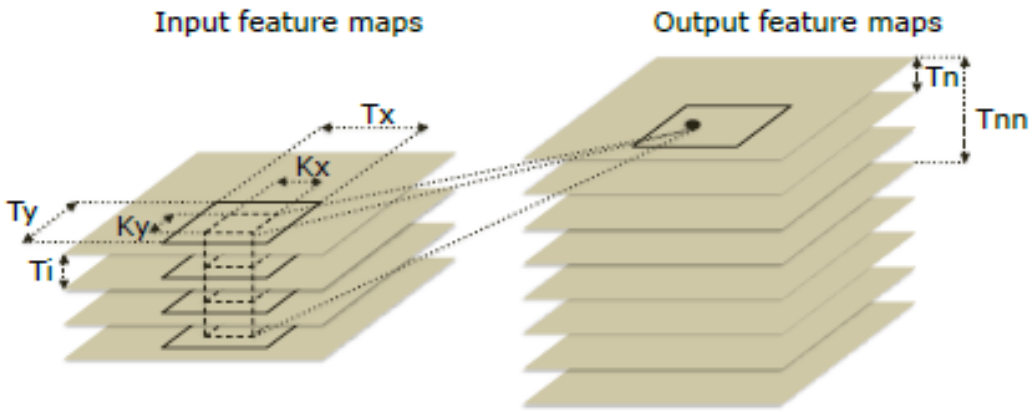


Figure 6. *Memory bandwidth requirements for each layer type (CONV3 has shared kernels, CONV5 has private kernels).*

Tiling Convolutional Layer: Pseudo Code

```
for (int yy = 0; yy ¡ Nyin; yy += Ty)  {
  for (int xx = 0; xx ¡ Nxin; xx += Tx)  {
    for (int nnn = 0; nnn ¡ Nn; nnn += Tnn)  {
      // — Original code — (excluding nn, ii loops)
      int yout = 0;
      for (int y = yy; y < yy + Ty; y += sy) {  // tiling for y;
        int xout = 0;
        for (int x = xx; x < xx + Tx; x += sx) {  // tiling for x;
          for (int nn = nnn; nn < nnn + Tnn; nn += Tn) {
            for (int n = nn; n < nn + Tn; n++)
              sum[n] = 0;
            // sliding window;
            for (int ky = 0; ky < Ky; ky++)
              for (int kx = 0; kx < Kx; kx++)
                for (int ii = 0; ii < Ni; ii += Ti)
                  for (int n = nn; n < nn + Tn; n++)
                    for (int i = ii; i < ii + Ti; i++)
                      // version with shared kernels
                      sum[n] += synapse[ky][kx][n][i]
                                * neuron[ky + y][kx + x][i];
                      // version with private kernels
                      sum[n] += synapse[yout][xout][ky][kx][n][i]}
                                * neuron[ky + y][kx + x][i];
            for (int n = nn; n < nn + Tn; n++)
              neuron[yout][xout][n] = non_linear_transform(sum[n]);
          } xout++; } yout++;
} } } }
```



- Two reuse opportunities

1) Convolutional reuse:
   - input reused $\frac{K_x K_y}{s_x s_y}$ times
2) Input reuse across N_n output FMs

- tile 1) but not 2) → $K_x K_y N_i$ fits in L1 $(10\times10\times1000)$

Precision Requirements

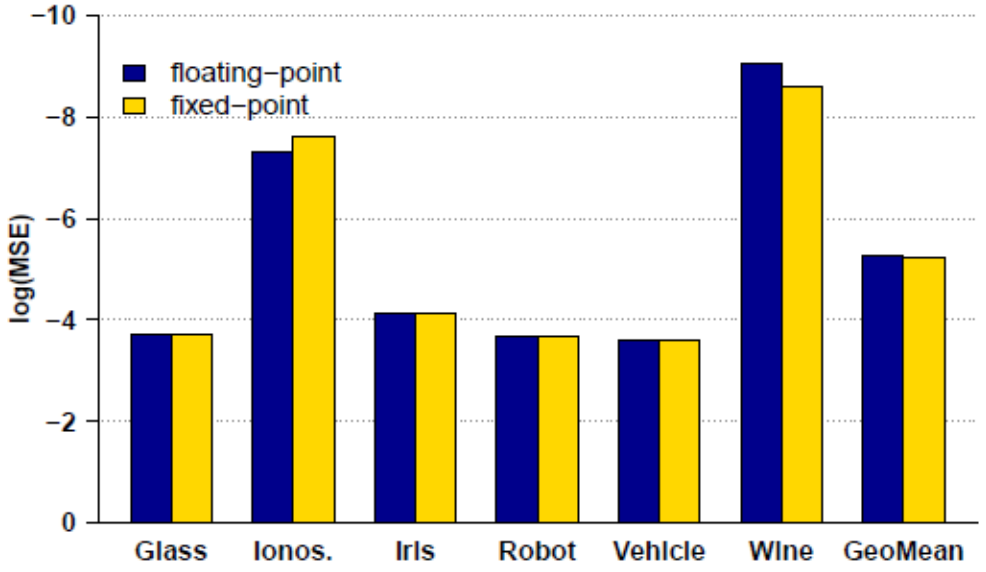


Figure 12. *32-bit floating-point vs. 16-bit fixed-point accuracy for UCI data sets (metric: log(*Mean Squared Error*)).*

- accuracy metric → MSE (SQNR)
- 16-b numbers → 6-b (integer part)+10-b (fractional part)

  "ample evidence that even smaller operators have almost no impact on the accuracy…"
- truncation for arithmetic operations

Type	Error Rate
32-bit floating-point	0.0311
16-bit fixed-point	0.0337

Table 1. *32-bit floating-point vs. 16-bit fixed-point accuracy for MNIST (metric: error rate).*

Type	Area (μm^2)	Power (μW)
16-bit truncated fixed-point multiplier	1309.32	576.90
32-bit floating-point multiplier	7997.76	4229.60

Table 2. *Characteristics of multipliers.*

Experimental Methodology

Since the design is not fabricated, experimentation is done via:

- **accelerator simulator**: custom **cycle-accurate**, **bit-accurate** C++ simulator, plugged to a main memory (DRAM) **model**
- **CAD tools**: **area**, **energy**, and **critical path delay** estimates obtained via simulating Verilog models after synthesis and PnR

The design is benchmarked against:

- **SIMD baseline**: use GEM5+McPAT combination → 4-issue superscalar x86 core with a 128-bit (816-bit) SIMD unit (SSE/SSE2), clocked at 2GHz. Capable of performing up to 8 16-bit operations every cycle

Network Benchmarks

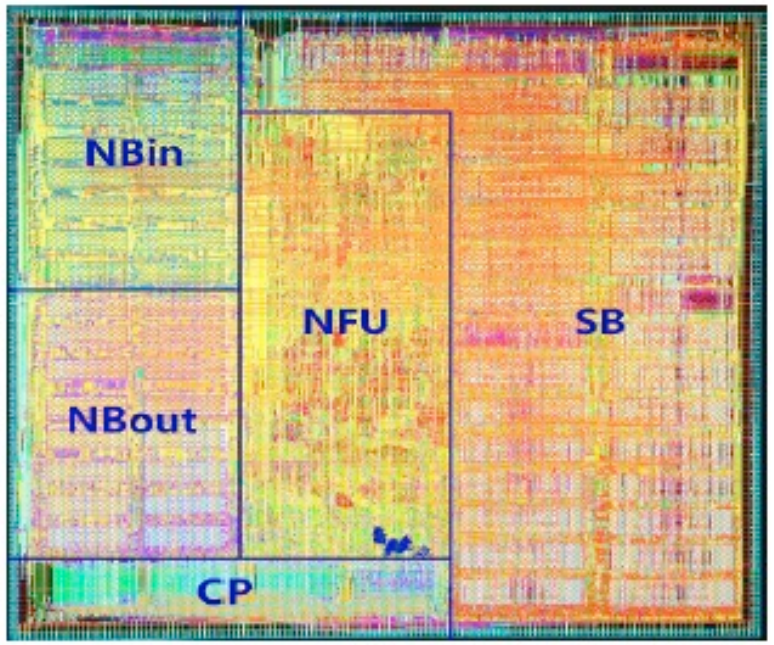
Layer	N_x	N_y	K_x	K_y	N_i	N_o	Description
CONV1	500	375	9	9	32	48	Street scene parsing
POOL1	492	367	2	2	12	-	(CNN) [13], (e.g.,
CLASS1	-	-	-	-	960	20	identifying "building", "vehicle", etc)
CONV2*	200	200	18	18	8	8	Detection of faces in YouTube videos (DNN) [26], largest NN to date (Google)
CONV3	32	32	4	4	108	200	Traffic sign
POOL3	32	32	4	4	100	-	identification for car
CLASS3	-	-	-	-	200	100	navigation (CNN) [36]
CONV4	32	32	7	7	16	512	Google Street View house numbers (CNN) [35]
CONV5*	256	256	11	11	256	384	Multi-Object
POOL5	256	256	2	2	256	-	recognition in natural images (DNN) [16], winner 2012 ImageNet competition

Table 5. *Benchmark layers (CONV=convolutional, POOL=pooling, CLASS=classifier; CONVx* indicates private kernels).*

Post Layout Numbers

Component or Block	Area in μm^2	(%)	Power in mW	(%)	Critical path in ns
ACCELERATOR	3,023,077		485		1.02
Combinational	608,842	(20.14%)	89	(18.41%)	
Memory	1,158,000	(38.31%)	177	(36.59%)	
Registers	375,882	(12.43%)	86	(17.84%)	
Clock network	68,721	(2.27%)	132	(27.16%)	
Filler cell	811,632	(26.85%)			
SB	1,153,814	(38.17%)	105	(22.65%)	
NBin	427,992	(14.16%)	91	(19.76%)	
NBout	433,906	(14.35%)	92	(19.97%)	
NFU	846,563	(28.00%)	132	(27.22%)	
CP	141,809	(5.69%)	31	(6.39%)	
AXIMUX	9,767	(0.32%)	8	(2.65%)	
Other	9,226	(0.31%)	26	(5.36%)	

Table 6. *Characteristics of accelerator and breakdown by component type (first 5 lines), and functional block (last 7 lines).*



- NFU-1: $T_n = 16; T_i = 16 \rightarrow$ 256 16-b multipliers
- NFU-2: 16 adder trees (15 adders each), 16-input shifter & max
- NFU-3: 16, 16-b truncated multipliers+16 adders

Speed-Up

- 118× for CONV and FC (CLASS) layers
- Poor speed-up for POOL layers
  - only adder tree in NFU-2 is used
- Analysis of speed-up provided

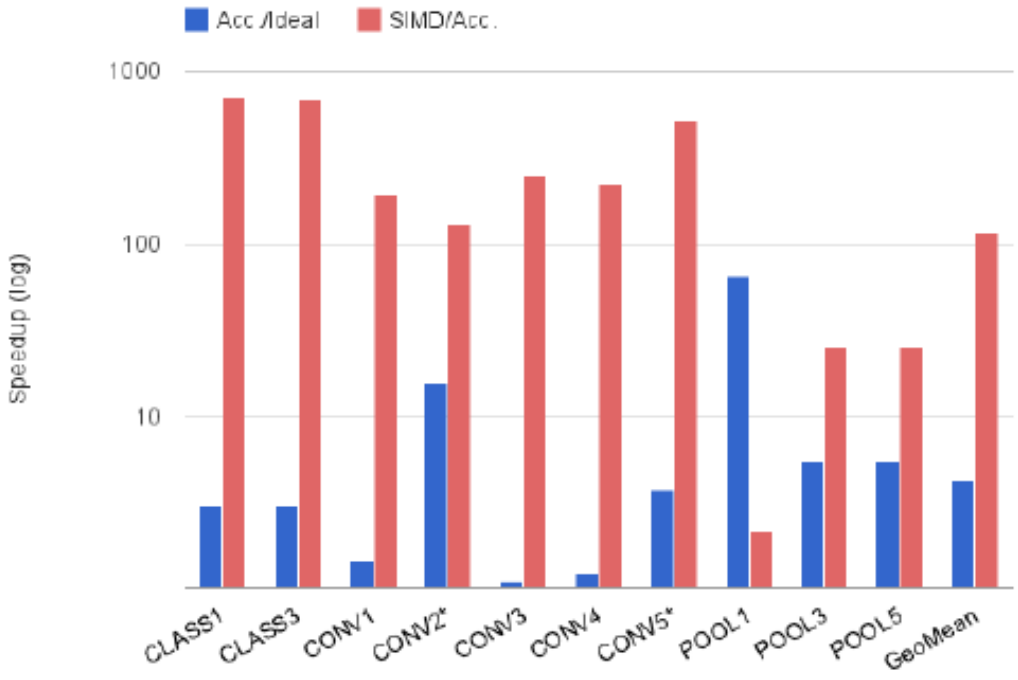
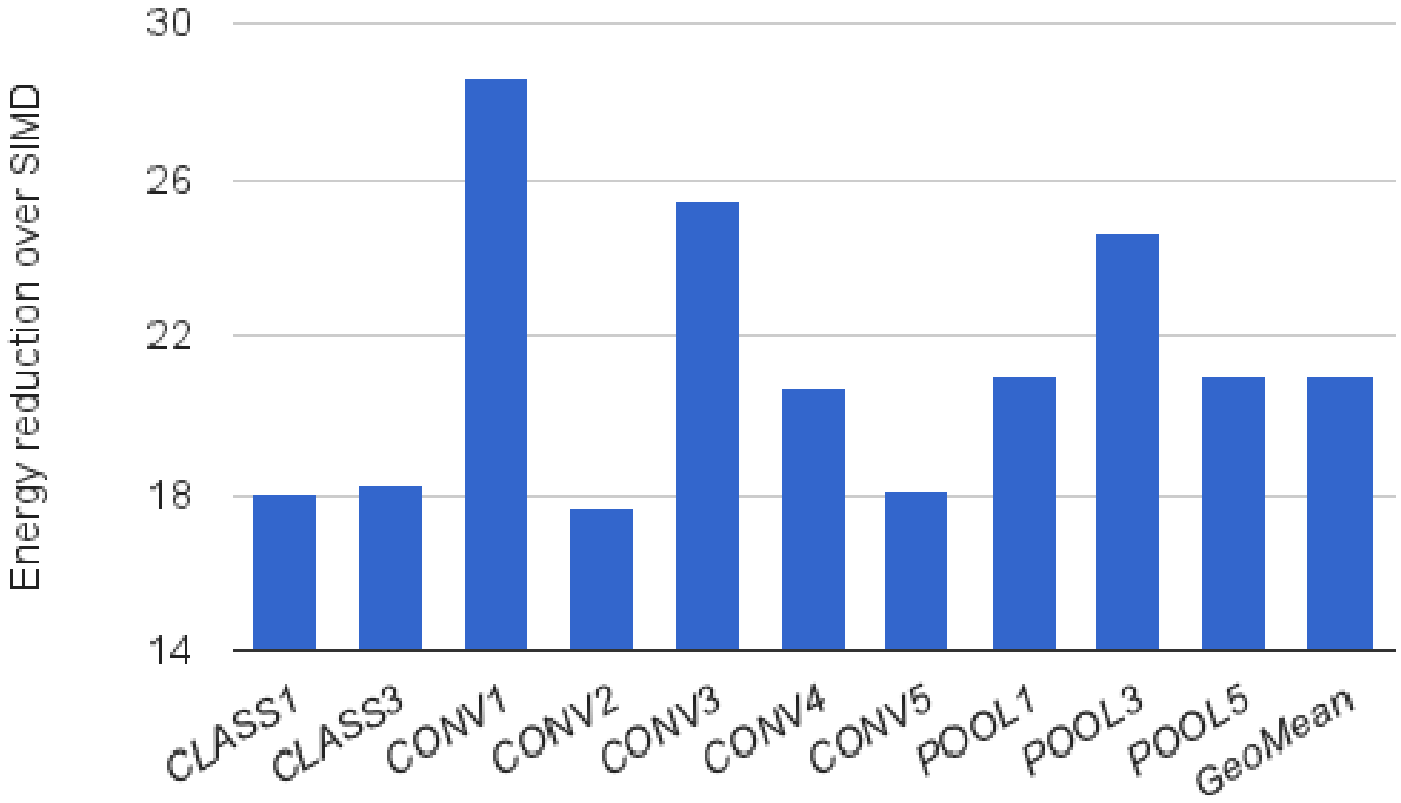


Figure 16. *Speedup of accelerator over SIMD, and of ideal accelerator over accelerator.*

Energy



- Average energy reduction → 21×

Energy

- Accelerator: memory access dominate energy

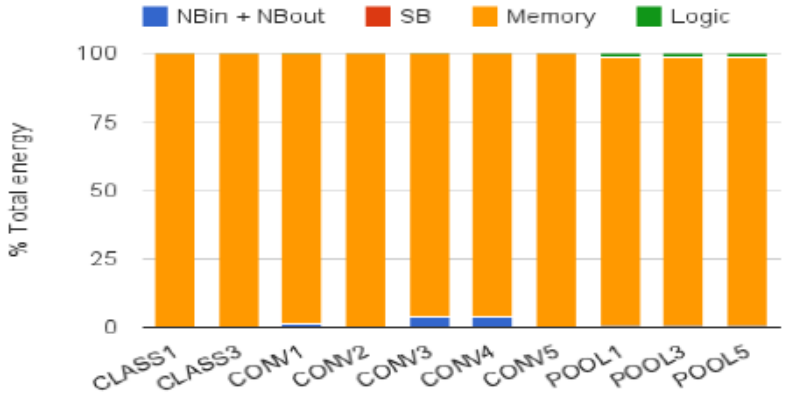


Figure 18. *Breakdown of accelerator energy.*

- SIMD: 2/3 energy in computation and 1/3 in memory access

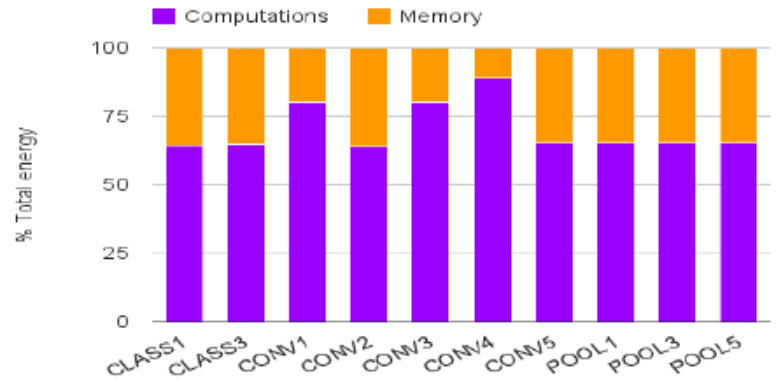
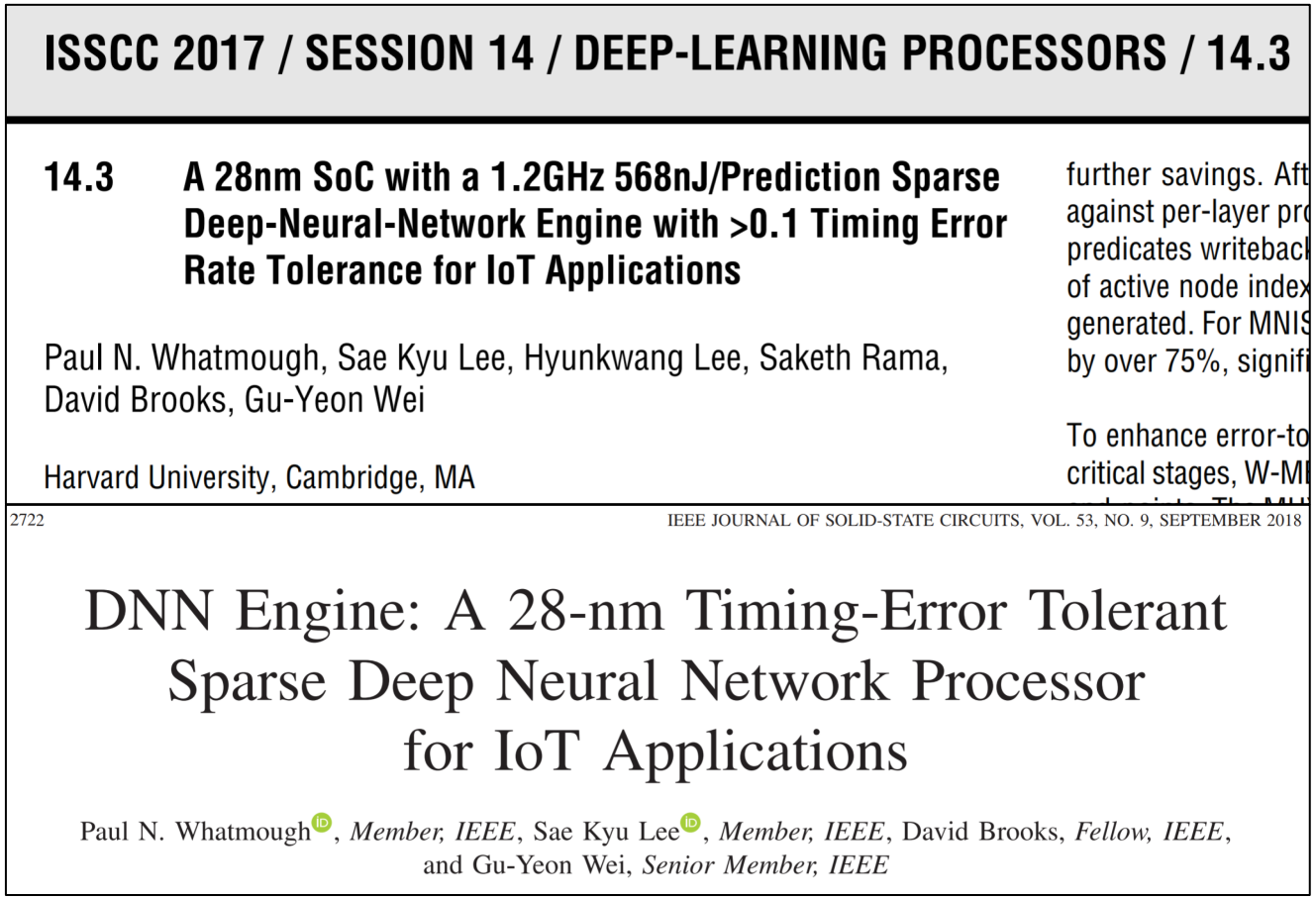


Figure 19. *Breakdown of SIMD energy.*

DNN Engine



ISSCC 2017 / SESSION 14 / DEEP-LEARNING PROCESSORS / 14.3

14.3 A 28nm SoC with a 1.2GHz 568nJ/Prediction Sparse Deep-Neural-Network Engine with >0.1 Timing Error Rate Tolerance for IoT Applications

Paul N. Whatmough, Sae Kyu Lee, Hyunkwang Lee, Saketh Rama, David Brooks, Gu-Yeon Wei

Harvard University, Cambridge, MA

further savings. Aft
against per-layer pr
predicates writebac
of active node index
generated. For MNIS
by over 75%, signifi

To enhance error-to
critical stages, W-MI

2722 IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 53, NO. 9, SEPTEMBER 2018

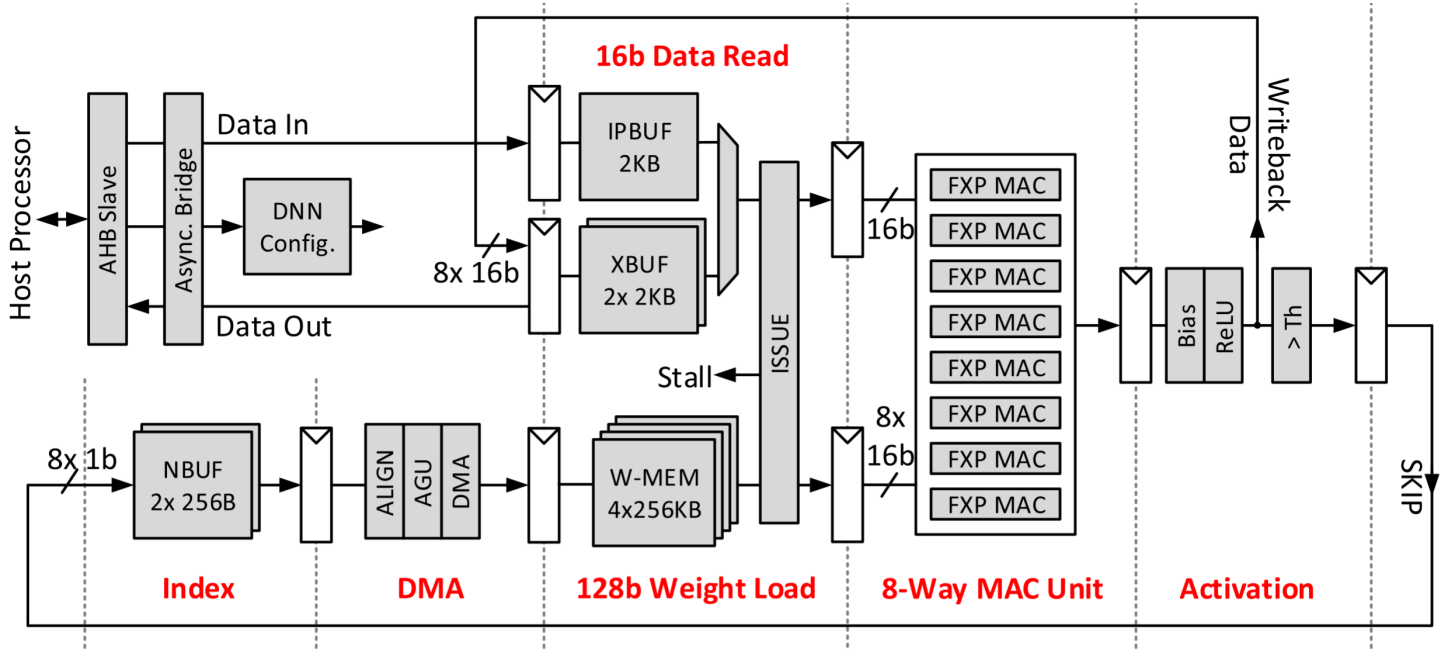
DNN Engine: A 28-nm Timing-Error Tolerant Sparse Deep Neural Network Processor for IoT Applications

Paul N. Whatmough, *Member, IEEE*, Sae Kyu Lee, *Member, IEEE*, David Brooks, *Fellow, IEEE*, and Gu-Yeon Wei, *Senior Member, IEEE*

Featured in ISSCC'17 and JSSC'18!

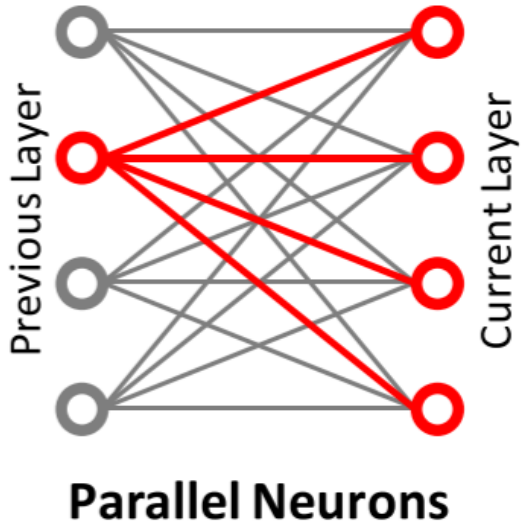
- Designed to accelerate **MLPs** only
- Performs **inference** only, using 8/16b **fixed-point** operations
- Exploits sparsity and noise tolerance for performance gains

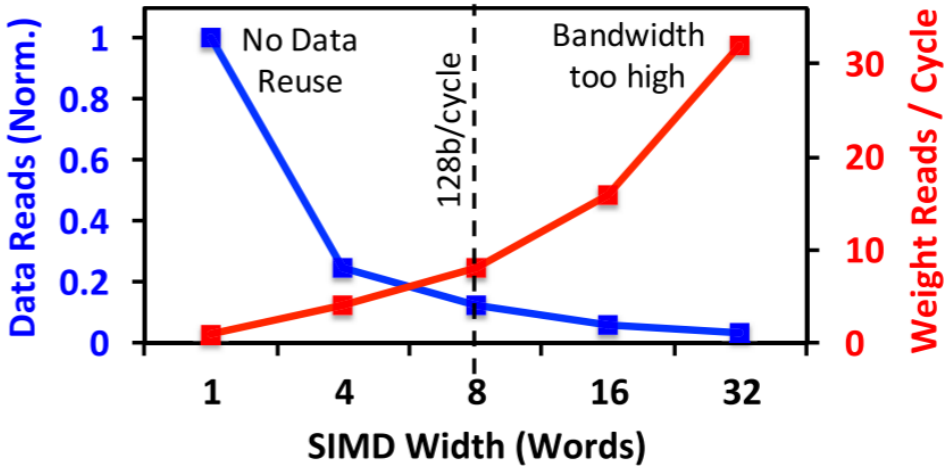
DNN Engine Architecture



- Layer computation carried out by 8 PEs
- Separate on-chip memories for:
  - weights (W-MEM), inputs (IPBUF), and intermediate activations (XBUF)
- Activation unit uses a thresholding function to detect "tiny" values
  - indices of "tiny" activations are stored in NBUF – more on that later

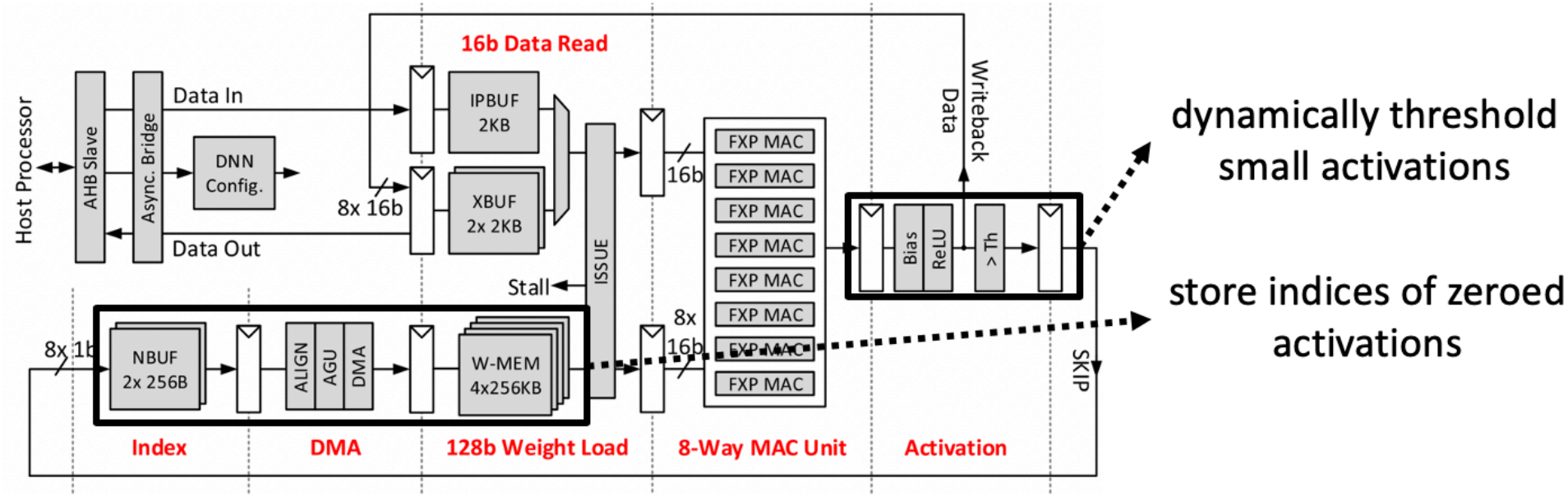
Output Stationary Mapping





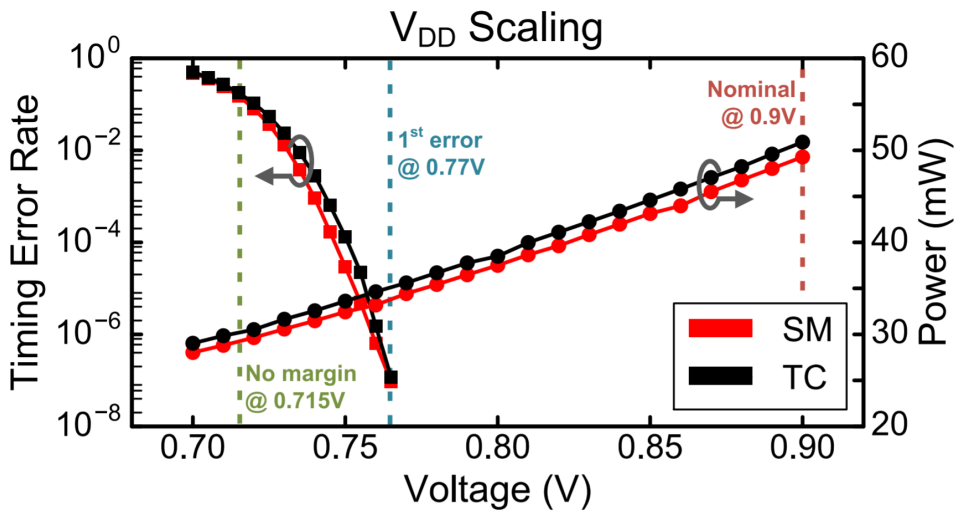
- Fully connected layer do not exhibit weight reuse
- to minimize memory reads, while using N PEs:
  - process each output activation in one PE (**output stationary**)
  - each input pixel is reused N times
- Choice of $N = 8$ justified by memory bandwidth (reuse vs bandwidth tradeoff)

Exploiting Activation Sparsity

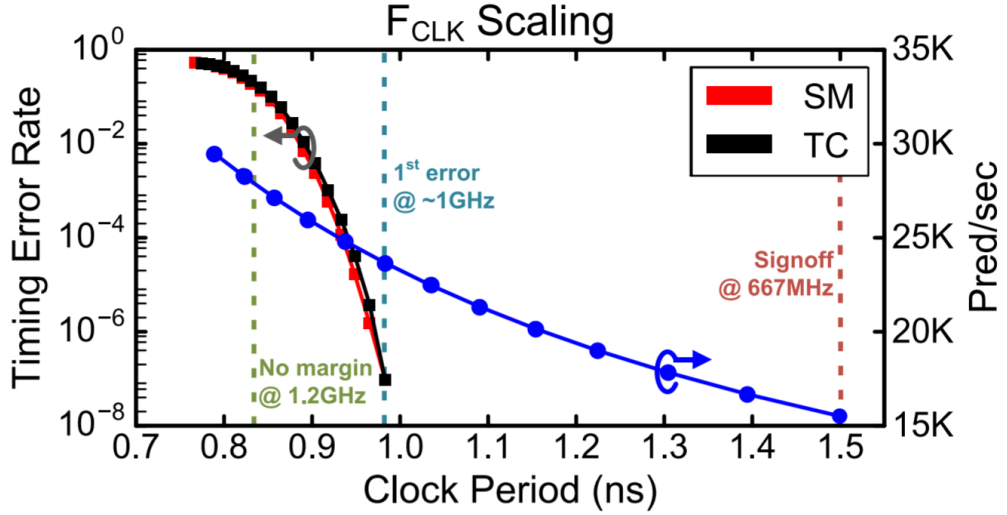


- Clock-gating saves **power**, but not **delay** [Eyeriss]
- In order to completely skip (**elide**) zero and small non-zero values:
  - thresholding to increase sparsity (exploiting noise tolerance of DNNs)
  - Dynamically **eliding** all zero operands at **XBUF writeback** to reduce processing delay
- NBUF stores indices of non-zero activations, which is used to drive the processing of a layer without pipeline stalls (zero **eliding**)

Exploiting DNN Noise Tolerance



Scale supply voltage V_{DD} @ fixed F_{CLK}

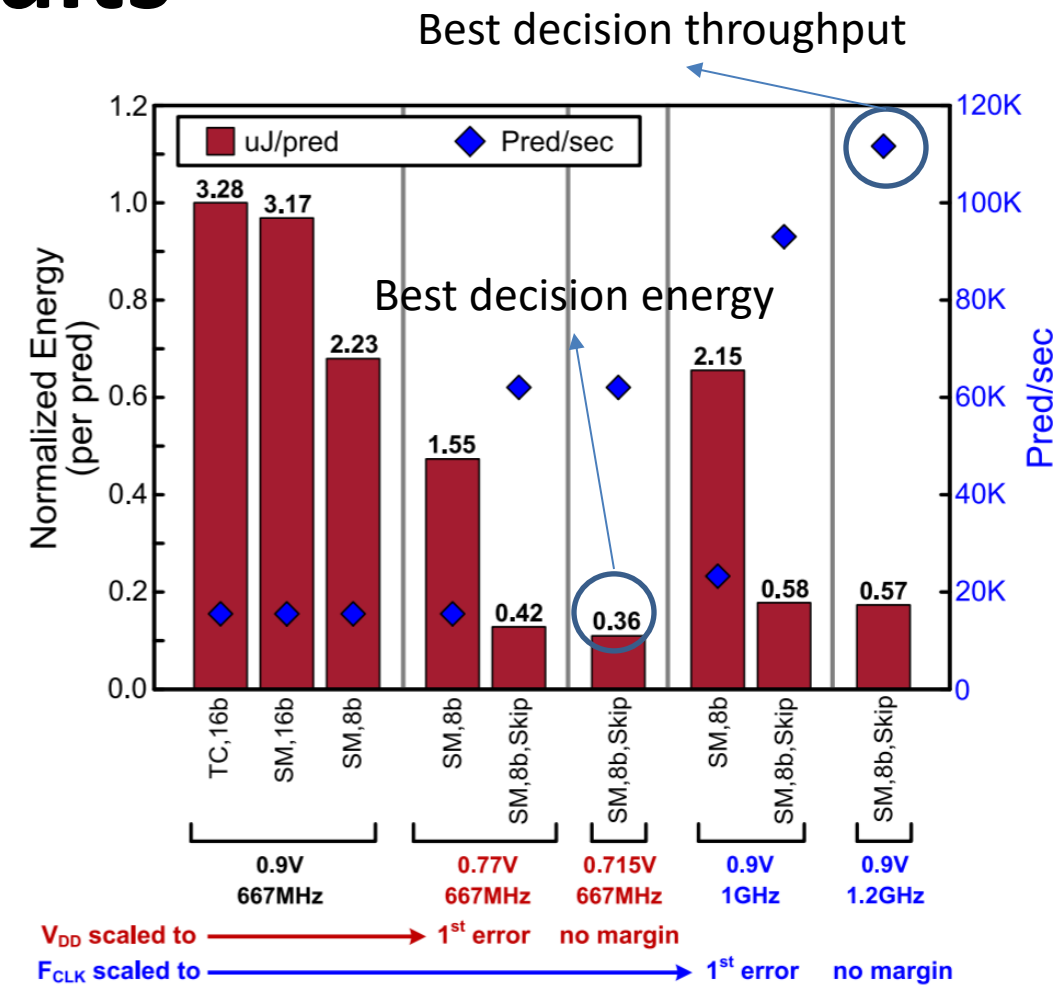


Scale clock frequency F_{CLK} @ fixed V_{DD}

- Idea is to scale either supply voltage or frequency (or both)
  - reduce **power** and increase **throughput**
  - will run into **timing violations** => **errors!**
- Some errors can be absorbed by the DNN's inherit noise tolerance
- Other errors can be mitigated using an advanced technique (**RAZOR**)

Results

- Benchmarked on MNIST with 98.36% accuracy
- Best decision throughput (~115 k decisions/sec) achieved at aggressive frequency scaling (1.2 GHz) + zero skipping
- Best energy/dec (~0.36μJ/dec) achieved at aggressive voltage scaling (0.715 V) + zero skipping

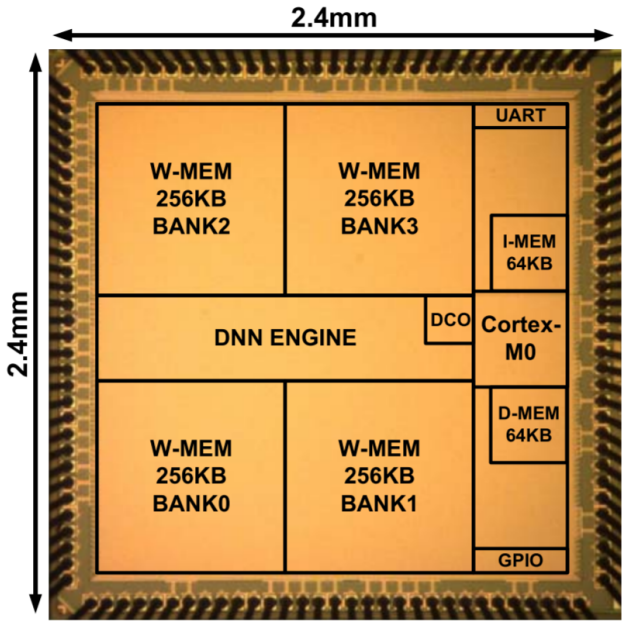


TC: two's complement
SM: signed magnitude

Chip Summary

TEST CHIP SUMMARY

Process Tech.	TSMC 28nm HPC 1P10M
Die Size	2.4mm x 2.4mm
Total SRAM	128KB (SoC), 1MB (WMEM), 6.5KB (DNN)
Total Flip-Flops	8460 (896/8460 RZFF)
ML Model	Fully-Connected Deep Neural Network
Data Types	8/16-bit Weights, 16-bit Act., 32bit Acc.
Model Support	Layers: 0-8, Nodes: 8-1024
Error Tolerance	$>10^{-1}$ @ 98.36% Accuracy
V_{DD}	0.9V (nom.) / 0.6 – 1.1 V (operational)
F_{MAX}	667MHz @ 0.9V (WC sign-off) 1.2GHz @ 0.9V (w/Razor DFS)
Total Power	33.7mW @ 667MHz/0.9V/8b (WC sign-off) 20.3mW @ 667MHz/0.715V/8b (DVS) 63.5mW @ 1.2GHz/0.9V/8b (DFS)
Leakage	3.03mW @ 0.9V



[Whatmough-ISSCC'17]

Precision Scalable Processor (PSP)

A 0.3-2.6 TOPS/W Precision-Scalable Processor for Real-Time Large-Scale ConvNets

Bert Moons, Marian Verhelst
Department of Electrical Engineering, ESAT-MICAS - KU Leuven, Leuven, Belgium
bert.moons@esat.kuleuven.be, marian.verhelst@esat.kuleuven.be

IEEE JOURNAL OF SOLID-STATE CIRCUITS, VOL. 52, NO. 4, APRIL 2017 903

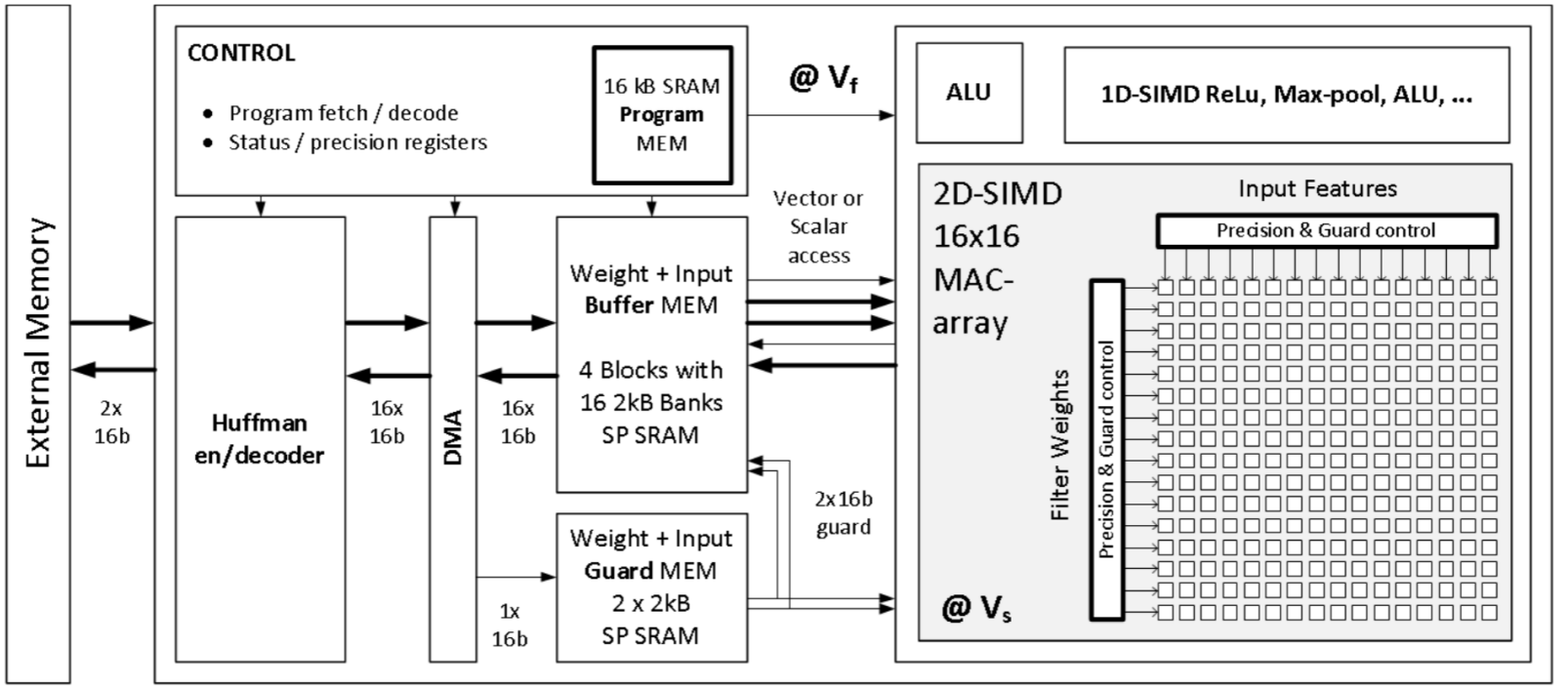
An Energy-Efficient Precision-Scalable ConvNet Processor in 40-nm CMOS

Bert Moons, *Student Member, IEEE*, and Marian Verhelst, *Senior Member, IEEE*

Featured in VLSI'16 and JSSC'17!

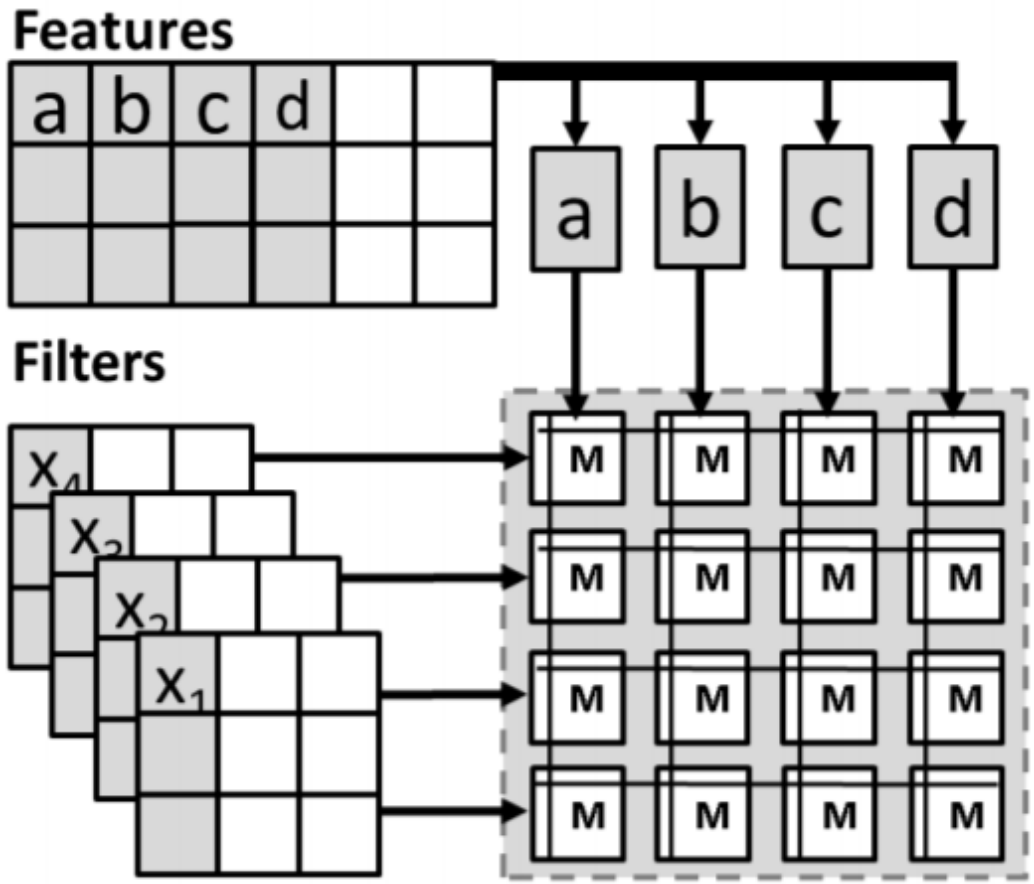
- Designed for CNN **inference** only
- First CNN accelerator to implement precision scalable processor 1b—16b **fixed-point**
- Exploits data statistics
  - zero skipping
  - Huffman encoding
- 76mW or 0.94 real TOPS/W for 47fps **AlexNet**

Chip Architecture

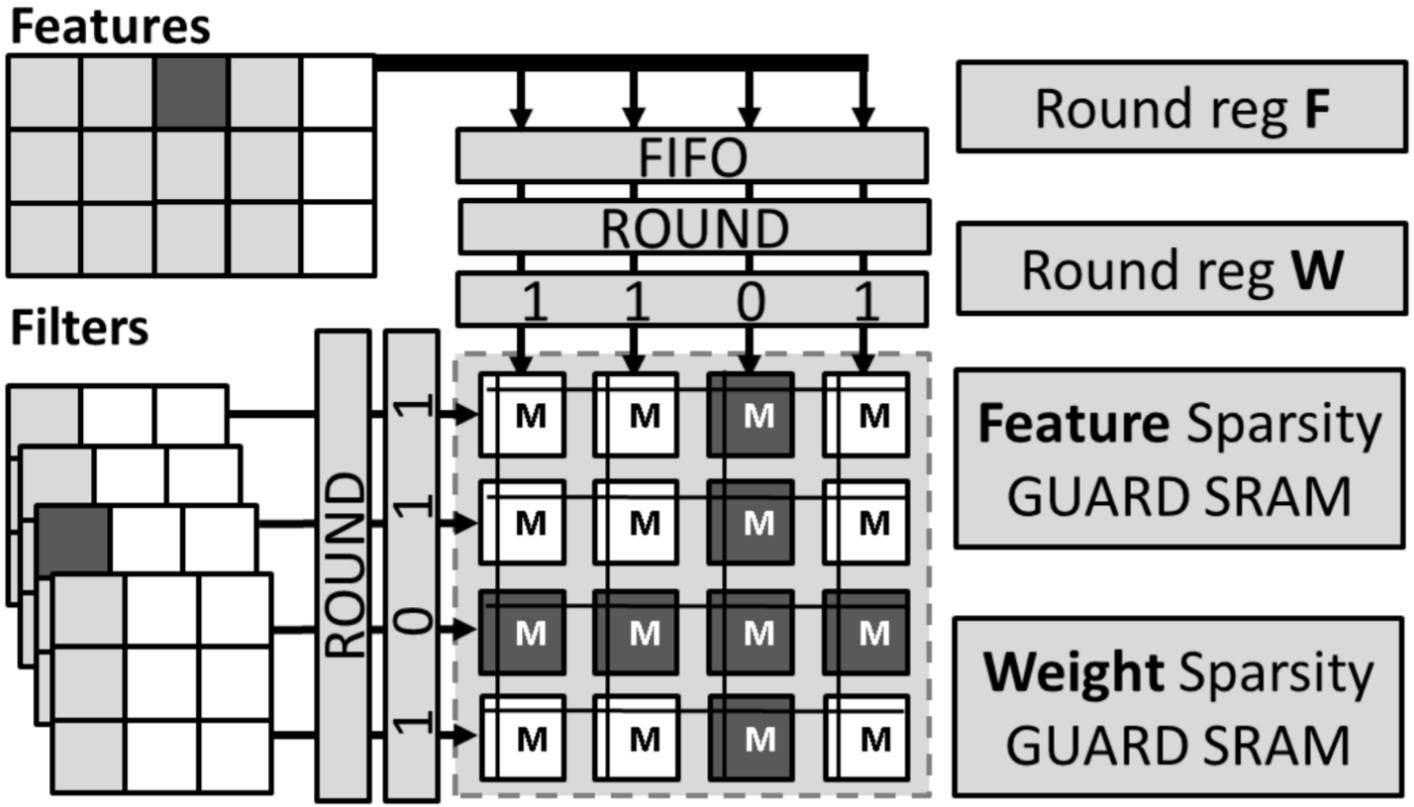


- Systolic architecture: 16×16 PEs
- Two power domains V_s (**scalable**) and V_f (**fixed**)
- Programmable architecture
- Separate memory banks, one for data and one for precision

Output Stationary Mapping

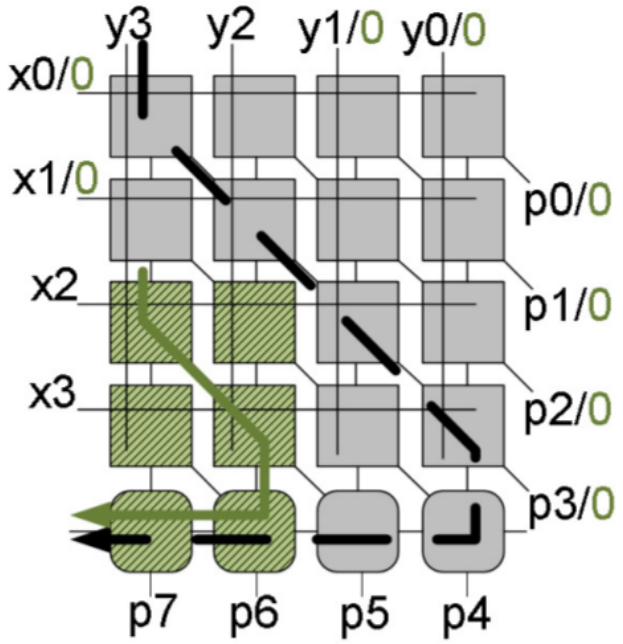


Zero Skipping (GUARD)

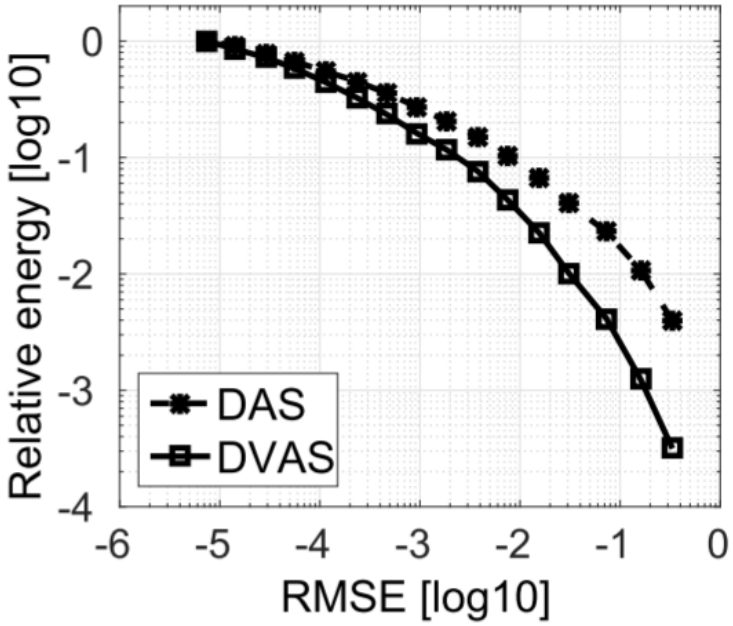


- Guarding Memory Fetches: words are conditionally fetched
- This is simply done by gating the enable signals to the on-chip SRAM memories

Dynamic Voltage – Accuracy Scaling



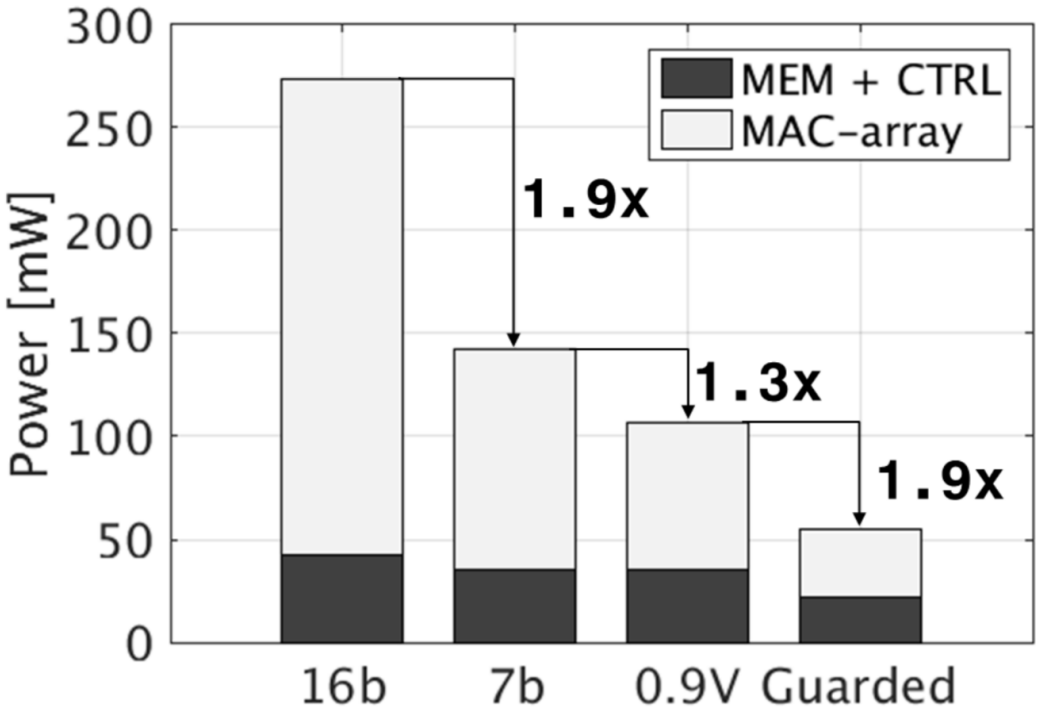
digital Baugh–Wooley multiplier with LSBs disabled



relative energy vs accuracy scaling

- Reducing bit-precision => reduction in critical path
- Can further reduce supply voltage => energy gains

Power Benefits



- Power consumption of AlexNet layer 2
- A total 5× gain in power consumption is achieved through:
  - precision, voltage scaling, and sparse operator guarding

Results

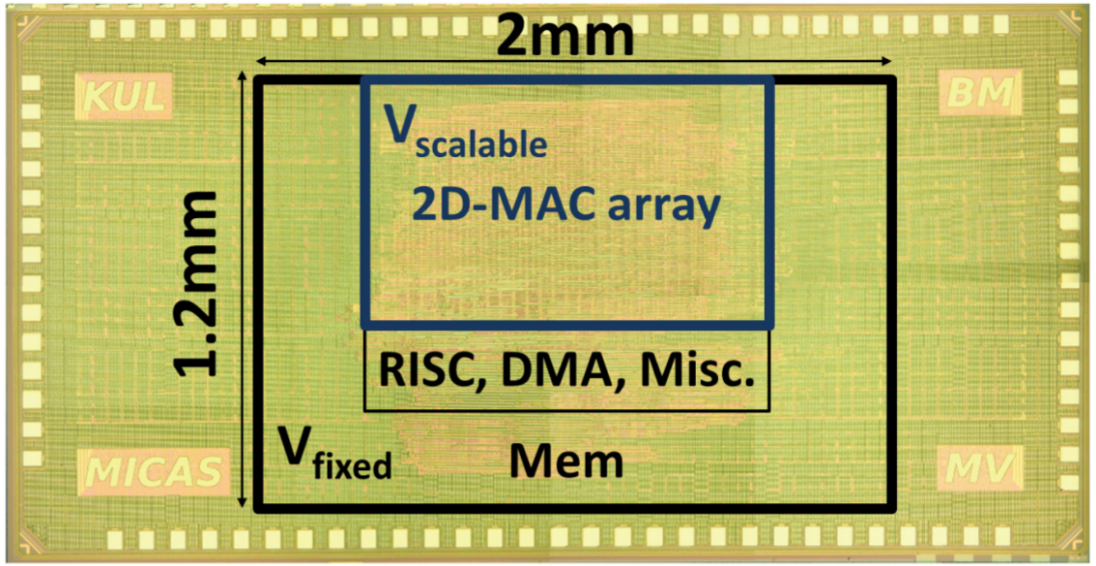
PERFORMANCE COMPARISON OF RELEVANT BENCHMARKS, RUNNING AT 204 MHz

Layer	Weight bits	Input bits	Weight Sparsity (%)	Input Sparsity (%)	Weight BW Reduc.	Input BW Reduc.	IO (MB/f)	HuffIO (MB/f)	Voltage (V)	MMACs/ Frame	Power (mW)	Effective TOPS/W
General CNN	16	16	0%	0%	1.0×	1.0×	–	–	1.1	–	**287**	**0.3**
AlexNet l1	7	4	21%	29%	1.17×	1.3×	1	0.77	0.85	105	85	0.96
AlexNet l2	7	7	19%	89%	1.15×	5.8×	3.2	1.1	0.9	224	55	1.4
AlexNet l3	8	9	11%	82%	1.05×	4.1×	6.5	2.8	0.92	150	77	0.7
AlexNet l4	9	8	04%	72%	1.00×	2.9×	5.4	3.2	0.92	112	95	0.56
AlexNet l5	9	8	04%	72%	1.00×	2.9×	3.7	2.1	0.92	75	95	0.56
Total / avg.	–	–	–	–	–	–	19.8	**10**	–	–	**76**	**0.9**
LeNet-5 l1	3	1	35%	87%	1.40×	5.2×	0.003	0.001	0.7	0.3	25	1.07
LeNet-5 l2	4	6	26%	55%	1.25×	1.9×	0.050	0.042	0.8	1.6	35	1.75
Total / avg.	–	–	–	–	–	–	0.053	**0.043**	–	–	**33**	**1.6**

- Benchmarked on different layers of:
  - AlexNet
  - LeNet

Chip Summary

Technology	40nm LP (1P_8M)
Core Area	1.2mmx2mm
On-Chip MEM Size	144kB
# MAC's	256
# Gates (NAND-2)	1600k
Supply voltage	0.55-1.1 V
Leakage	0.7 mW
Frequency	12-204 MHz
Word bit width	1-16 bit fixed
# Filters	All-programmable
# Channels	All-programmable
Stride	Horizontal: 1-4 Vertical: no limit
Peak performance	102 GOPS
Power (AlexNet)	76 mW
Throughput (AlexNet)	227x227 @ 47fps



Course Web Page

https://courses.grainger.illinois.edu/ece598nsg/fa2020/
https://courses.grainger.illinois.edu/ece498nsu/fa2020/

http://shanbhag.ece.uiuc.edu