Eyeriss : A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks

Motivation

Convolutions dominate for over 90% of the CNN operations and dominate runtime. Although these operations can leverage highly-parallel compute paradigms, throughput might not scale accordingly due to the accompanying bandwidth requirement and the energy consumption remains high as data movement can be more expensive than computation.

- Need to develop dataflows that support parallel processing with minimal data movement
- Differences in data movement energy cost based on where the data is stored also needs to be accounted for

Spatial Architecture

Spatial architectures are a class of accelerators that can exploit high compute parallelism using direct communication between an arrayof simple processing engines(PEs).

- Fine-grained SAs - in the form of an FPGA
- Coarse-grained SAs – tiled arrays of ALU style PEs connected together via on-chip networks

Coarse-grained SAs are a popular implementation choice for CNN accelerators-

- Operations in a CNN layer are uniform and exhibit high parallelism which can be computed with parallel ALU-style PE
- Direct inter-PE communication can be used very effectively for passing partial sums and sharing input data

The system provides four levels of storage hierarchy for data accesses –
DRAM
global buffer
array (inter-PE communication)
RF
Accessing data from a different level implies a different energy cost

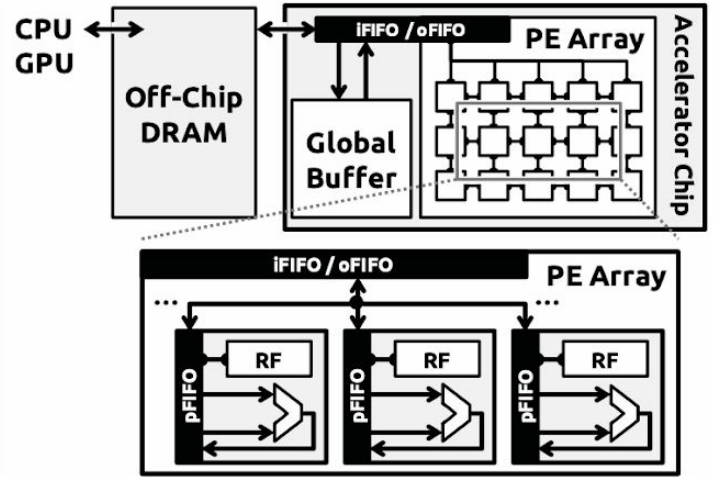
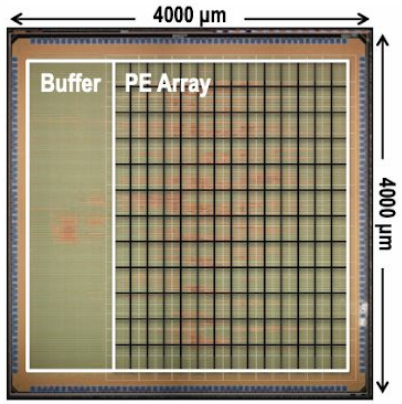


Figure 1. Block diagram of a general CNN accelerator system consisting of a spatial architecture accelerator and an off-chip DRAM. The zoom-in shows the high-level structure of a PE.



Process	65nm CMOS
# of PEs	168
RF Size/PE	0.5 kB
Buffer Size	108 kB
Clock Rate	200 MHz
Precision	16-bit Fixed-Point

Figure 4. Die photo and spec of the Eyeriss chip [41].

CNN Background

- A convolutional neural network (CNN) is constructed by stacking multiple computation layers as a directed acyclic graph
- The primary computation of CNN is in the convolutional (CONV) layers, which perform high-dimensional convolutions.
- A CONV layer applies filters on the input fmaps (ifmaps) to extract embedded visual characteristics and generate the output fmaps (ofmaps). The dimensions of both filters and fmaps are 4D.
- A small number of fully-connected (FC) layers are typically stacked behind the CONV layers for classification purposes.
- In between CONV and FC layers, additional layers can be added optionally, such as the pooling (POOL) and normalization (NORM) layers.

$$\mathbf{O}[z][u][x][y] = \mathbf{B}[u] + \sum_{k=0}^{C-1} \sum_{i=0}^{R-1} \sum_{j=0}^{R-1} \mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j],$$

$$0 \le z < N, 0 \le u < M, 0 \le x, y < E, E = (H - R + U)/U. \tag{1}$$

Challenges in CNN processing

Data Handling

Exploiting different types of input data reuse

- Convolutional reuse- Due to the weight sharing property in CONV layers, a small amount of unique input data can be shared across many operations.
- Filter data reuse
- ifmap reuse

Proper operation scheduling so that generated psums can be reduced as soon as possible

Adaptive Processing

The dataflow must be efficient for different shapes, and the hardware architecture must be programmable to dynamically map to an efficient dataflow.

Existing CNN Dataflows

- Weight Stationary (WS) Dataflow
- Output Stationary (OS) Dataflow
- No Local Reuse (NLR) Dataflow

Dataflow	**Data Handling**
WS	Maximize *convolutional reuse* and *filter reuse* of weights in the RF.
SOC-MOP OS	Maximize *psum accumulation* in RF. *Convolutional reuse* in array.
MOC-MOP OS	Maximize *psum accumulation* in RF. *Convolutional reuse* and *ifmap reuse* in array.
MOC-SOP OS	Maximize *psum accumulation* in RF. *Ifmap reuse* in array.
NLR	*Psum accumulation* and *ifmap reuse* in array.

Energy Efficient Dataflow : Row Stationary

- **1D Convolution Primitives** - It breaks the high-dimensional convolution down into 1D convolution primitives that can run in parallel; each primitive operates on one row of filter weights and one row of ifmap pixels, and generates one row of psums. Psums from different primitives are further accumulated together to generate the ofmap pixels.
- **Two-step Primitive Mapping** –

Logical mapping - deploys the primitives into a logical PE array, which has the same size as the number of 1D convolution primitives and is usually much larger than the physical PE array in hardware.

Physical mapping - *folds* the logical PE array so it fits into the physical PE array.

Folding - serializing the computation, and is determined by the amount of on-chip storage, including both the global buffer and local RF.

- **Energy-Efficient Data Handling**

RF - There are *convolutional reuse* within the computation of each primitive, *filter reuse* and *ifmap reuse* due to input data sharing between folded primitives, and *psum accumulation* within each primitive and across primitives.

Array(inter-PE communication) - *Filter reuse* and *ifmap reuse* can be achieved by having multiple sets mapped spatially across the physical PE array.

Psum accumulation is done within each set as well as across sets that are mapped spatially.

Global Buffer - Used to exploit the rest of *filter reuse*, *ifmap reuse* and *psum accumulation* that remain from the RF and array levels after the second phase folding.

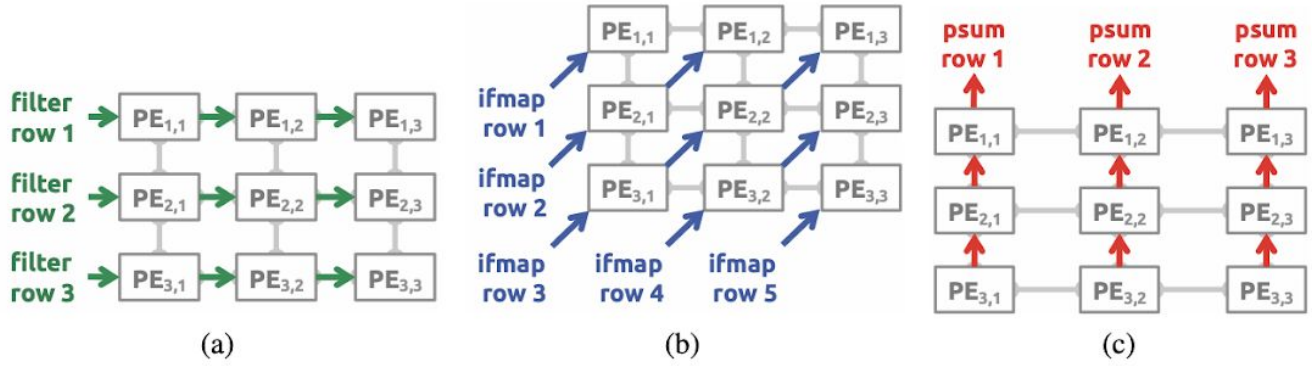


Figure 6. The dataflow in a logical PE set to process a 2D convolution. (a) rows of filter weight are reused across PEs horizontally. (b) rows of ifmap pixel are reused across PEs diagonally. (c) rows of psum are accumulated across PEs vertically. In this example, $R = 3$ and $H = 5$.

Experimental Methodology

- All dataflows are given the same number of PEs with the same storage area, which includes the global buffer and RF.
- The baseline storage area for a given number of PEs is calculated as

  #PE × *Area*(512B RF) + *Area*((#PE × 512B) global buffer).
- The accelerator throughput is assumed to be proportional to the number of active PEs for a dataflow.

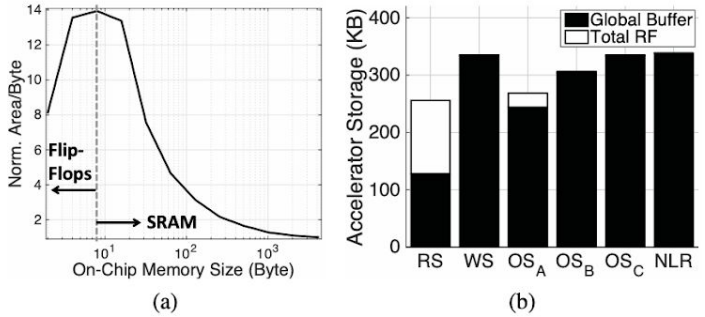


Figure 7. The trade-off between storage area allocation and the total storage size. (a) A smaller memory have a higher cost on area utilization. (b) Due to the area allocation between global buffer and RF, the total on-chip storage size varies between dataflows.

Energy Efficiency Analysis

The way each MAC operation in Eq. (1) fetches inputs (filter weight and ifmap pixel) and accumulates psums introduces different energy costs due to two factors:

- how the dataflow exploits input data reuse and psum accumulation scheduling
- fetching data from different storage elements in the architecture have different energy costs.

Data Movement Hierarchy

	DRAM	Global Buffer (>100kB)	Array (inter-PE) (1-2mm)	RF (0.5kB)
Norm. Energy	200×	6×	2×	1×

Table IV
NORMALIZED ENERGY COST RELATIVE TO A MAC OPERATION EXTRACTED FROM A COMMERCIAL 65NM PROCESS.

Analysis Methodology

- **Input Data Access Energy Cost**:

Reuse at each level is defined as the number of times each data value is read from this level to its lower-cost levels during its lifetime. Suppose the total number of reuses for a data value is a×b×c×d, it can be split into reuses at DRAM, global buffer, array and RF for a, b, c, and d times, respectively.

The energy cost estimation for this reuse pattern is

$a \times EC(DRAM) + ab \times EC(\text{global buffer}) + abc \times EC(\text{array}) + abcd \times EC(RF)$,

where $EC(\cdot)$ is the energy cost

- **Psum Accumulation Energy Cost:**

The number of accumulation at each level is defined as the number of times each data goes in and out of its lower-cost levels during its lifetime.

The energy cost can then be estimated as

$(2a-1) \times EC(DRAM) + 2a(b-1) \times EC(\text{global buffer}) + ab(c-1) \times EC(\text{array}) + 2abc(d-1) \times EC(RF)$.

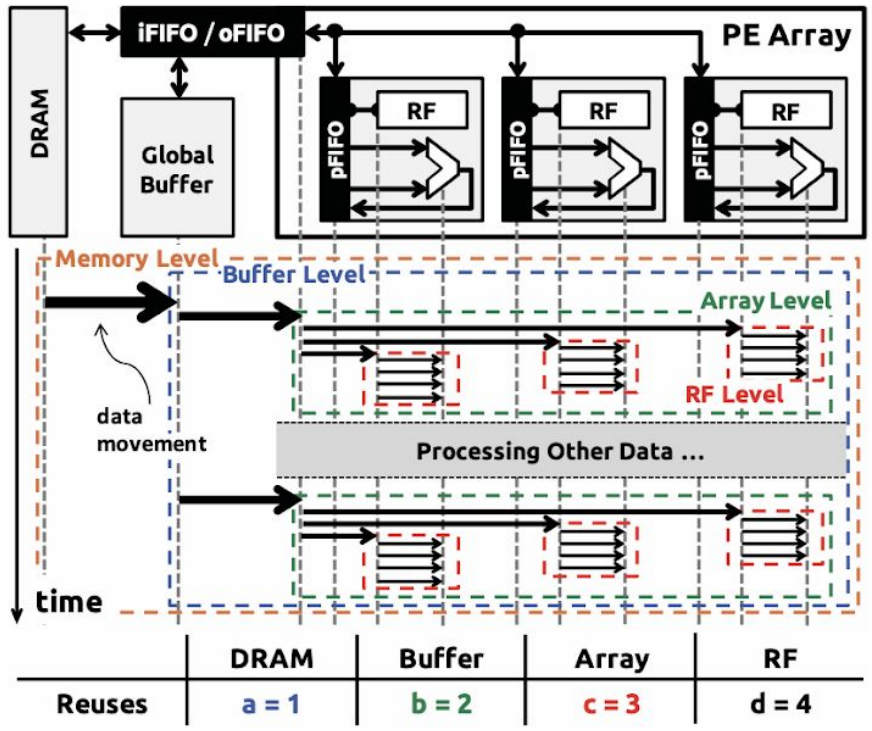


Figure 8. An example of the ifmap pixel or filter weight being reused across four levels of hierarchy.

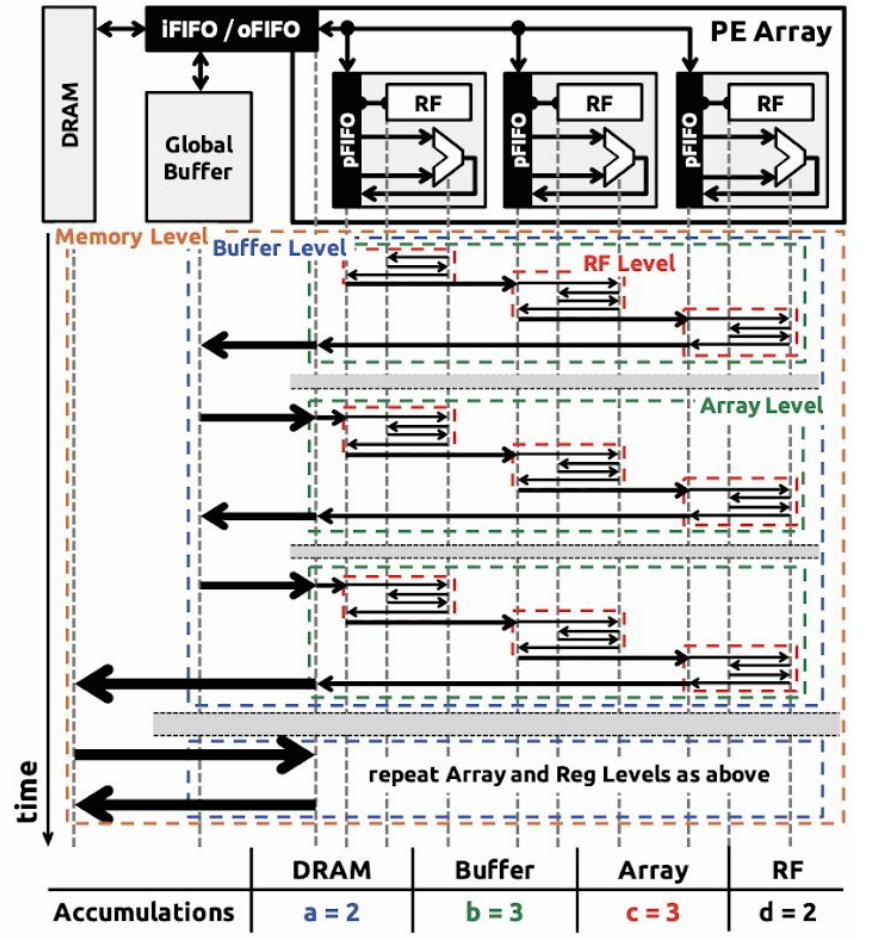


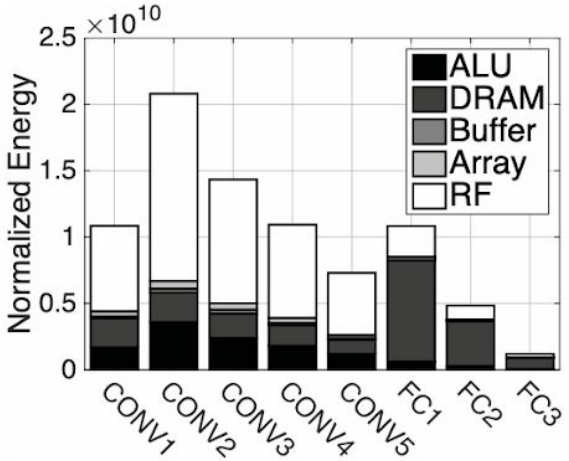
Figure 9. An example of the psum accumulation going through four levels of hierarchy.

Results

AlexNet is used as the CNN model for benchmarking.

Its 5 CONV and 3 FC layers provide a wide range of shapes that are suitable for testing the adaptability of different dataflows.

The figure shows the energy breakdown across the storage hierarchy in the 5 CONV and 3 FC layers of AlexNet. The energy is normalized to one ALU operation, i.e., a MAC.



Energy consumption breakdown of RS dataflow in AlexNet.

Dataflow Comparison in CONV layers

- DRAM Accesses
- Energy Consumption - Overall, RS is 1.4× to 2.5× more energy efficient than other dataflows.

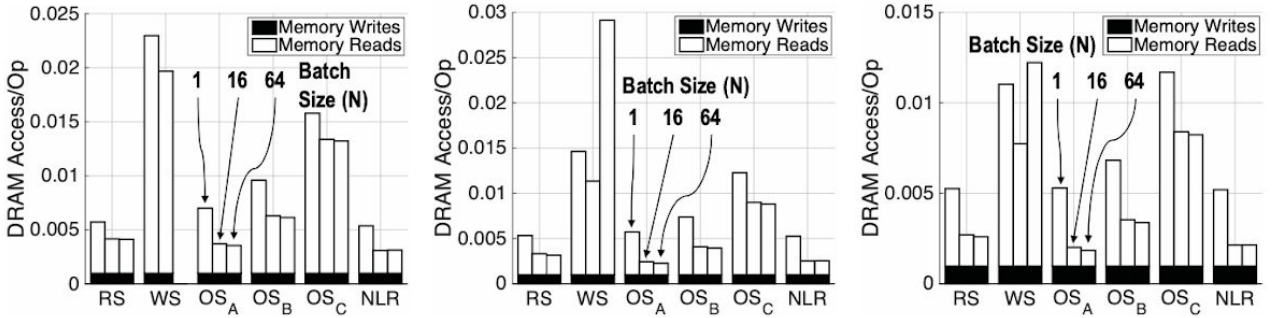


Figure 11. Average DRAM accesses per operation of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 and (c) 1024.

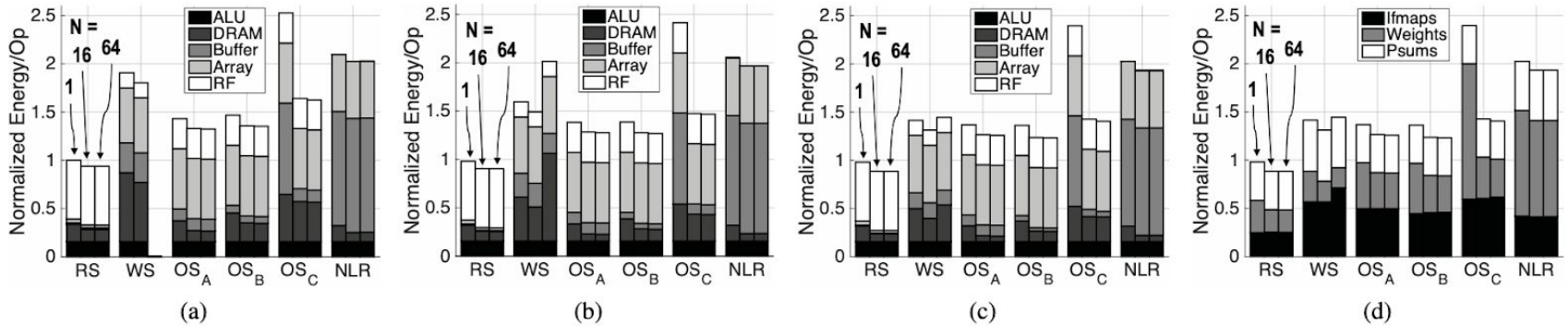


Figure 12. Energy consumption of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 and (c) 1024. (d) is the same as (c) but with energy breakdown in data types. The energy is normalized to that of RS at array size of 256 and batch size of 1. The RS dataflow is 1.4× to 2.5× more energy efficient than other dataflows.

- Energy Delay Product - Energy-delay product is used to verify that a dataflow does not achieve high energy efficiency by sacrificing processing parallelism, i.e., throughput. The delay is calculated as the reciprocal of number of active PEs.

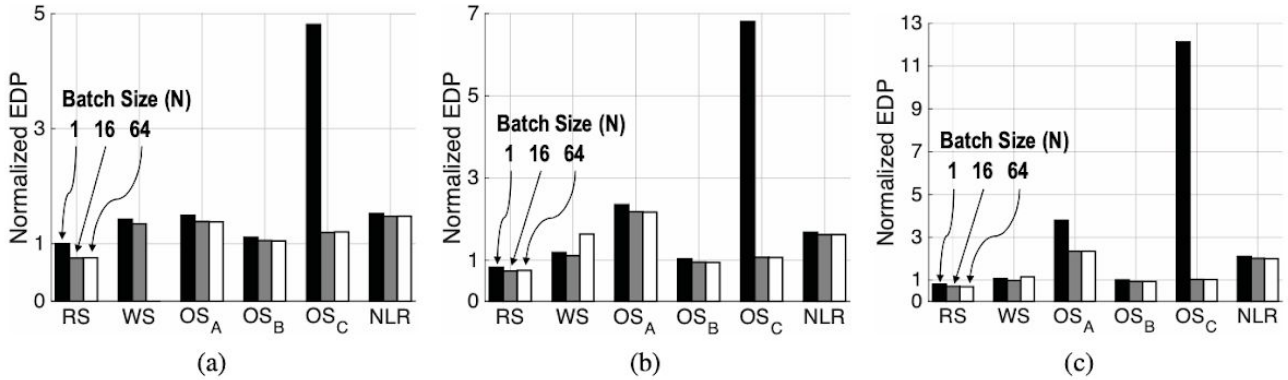


Figure 13. Energy-delay product (EDP) of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 and (c) 1024. It is normalized to the EDP of RS at PE array size of 256 and batch size of 1.

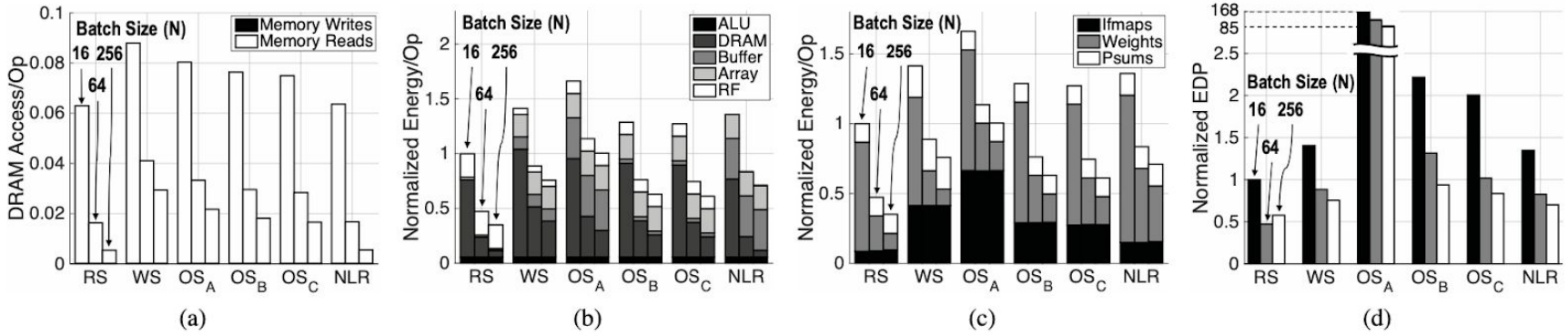


Figure 14. (a) average DRAM accesses per operation, energy consumption with breakdown in (b) storage hierarchy and (c) data types, and (d) EDP of the six dataflows in FC layers of AlexNet under PE array size of 1024. The energy consumption and EDP are normalized to that of RS at batch size of 1.

Conclusion

- Compared with existing dataflows such as the output stationary (OS), weight stationary (WS), and no local reuse (NLR) dataflows using AlexNet as a benchmark, the RS dataflow is 1.4× to 2.5× more energy efficient in convolutional layers, and at least 1.3× more energy efficient in fully-connected layers for batch sizes of at least 16.
- For all dataflows, increasing the size of the PE array helps to improve the processing throughput at similar or better energy efficiency.
- Larger batch sizes also result in better energy efficiency
- Finally, for the RS dataflow, the area allocation between processing and storage has a limited effect on energy-efficiency, since more PEs allow for better data reuse, which balances out the effect of less on-chip storage.

EIE: Efficient Inference Engine on Compressed Deep Neural Network

Motivation

- **Fully Connected Layers**
  - Convolution is easy to accelerate - data reuse, extensively studied
  - FC layers are bandwidth limited - no data reuse
  - FC layers account for 96% of connections
- **DRAM**
  - Power consumption dominated by fetching weights from DRAM
  - 2 orders of magnitude more expensive than ALU operations
- **Latency**
  - Targeting real-time inferencing
  - Batching (throughput) is undesirable

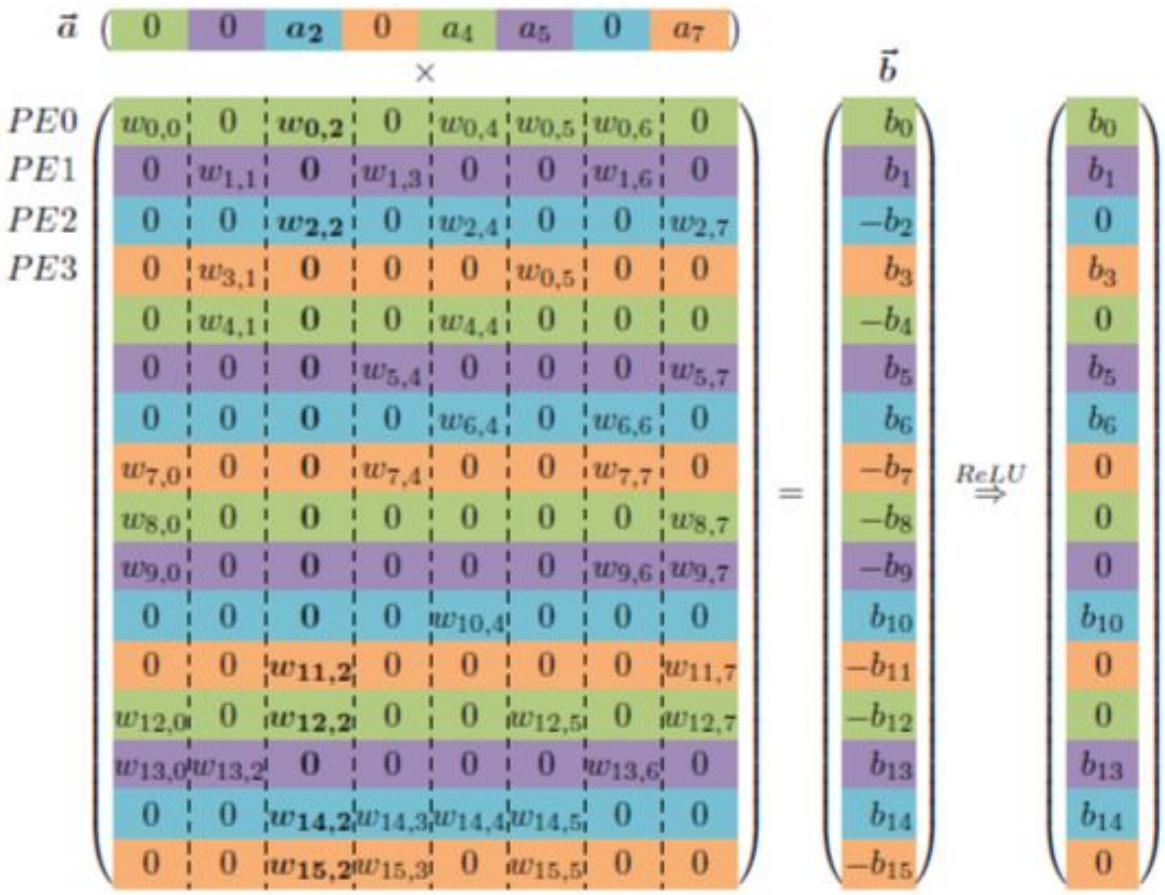
Compression

$$b_i = ReLU\left(\sum_{j=0}^{n-1} W_{ij} a_j\right)$$

$$b_i = ReLU\left(\sum_{j \in X_i \cap Y} S[I_{ij}] a_j\right)$$

- **Pruning**
  - Reduces the number of weights present
  - 4 - 25% of original matrix
  - Results in sparse matrices
- **Sparsity**
  - Only do computations on non-zero inputs
  - 70% zero activations => 65.16% energy savings
- **Weight-Sharing**
  - Weights are typically stored as 32-bit floating point numbers
  - Compressed and quantized into a 16-entry table of 16-bit fixed point numbers
  - Weights stored as a 4-bit index into table

Representation



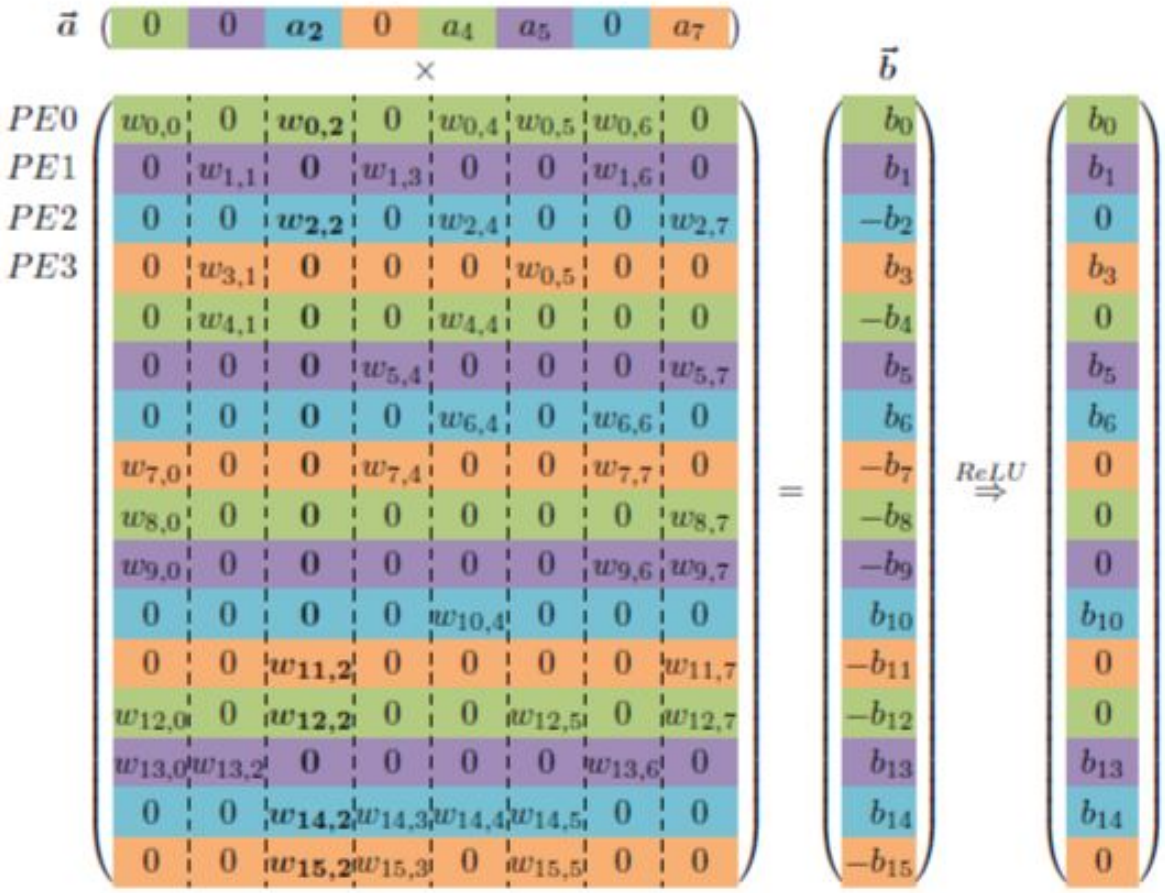
Virtual Weight	$W_{0,0}$	$W_{8,0}$	$W_{12,0}$	$W_{4,1}$	$W_{0,2}$	$W_{12,2}$	$W_{0,4}$	$W_{4,4}$	$W_{0,5}$	$W_{12,5}$	$W_{0,6}$	$W_{8,7}$	$W_{12,7}$
Relative Row Index	0	1	0	1	0	2	0	0	0	2	0	2	0
Column Pointer	0	3	4	6	6	8	10	11	13				

- Modified compressed sparse column (CSC) format
- v = virtual weight = table index of non-zero weights
- z = relative row index = number of zeros before non-zero weight
- p = column pointer = index into (v,x) to indicate the start of each column

Insights

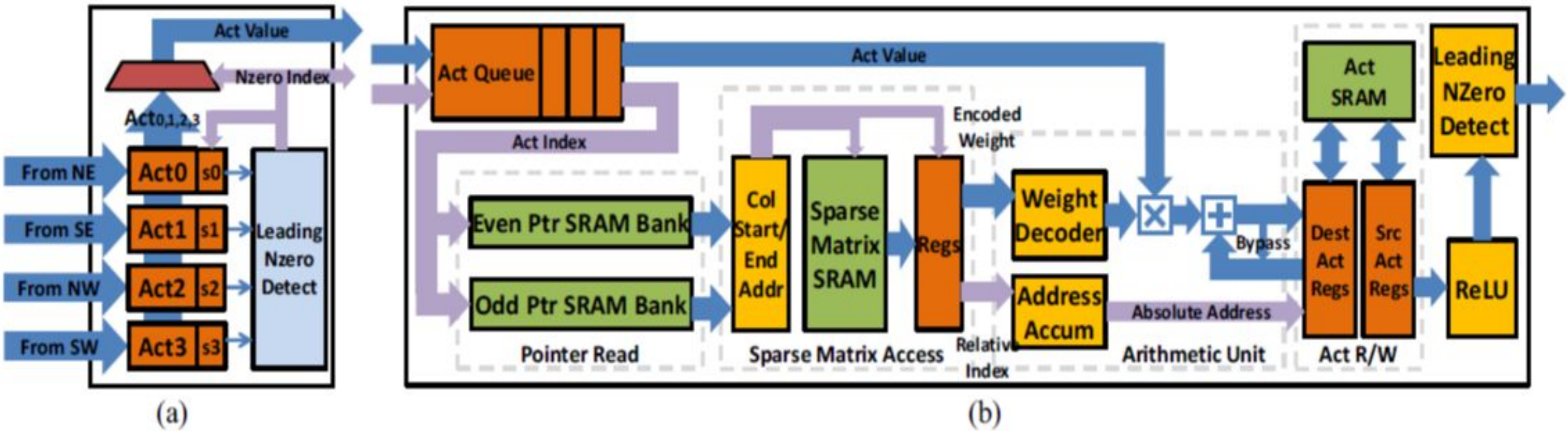
- **Utilize on-chip SRAM**
  - Compressed models fit in SRAM
  - 120x better energy savings over DRAM
- **Dynamic Sparsity**
  - Not addressed by previous SPMV or DNN accelerators
- **Compression needs custom hardware!!**
  - Compression is great, but it runs poorly on CPUs and GPUs
  - Irregular Patterns =>Extra levels of indirection => added complexity => inefficiency
  - Compression alone only yields 3x speedup

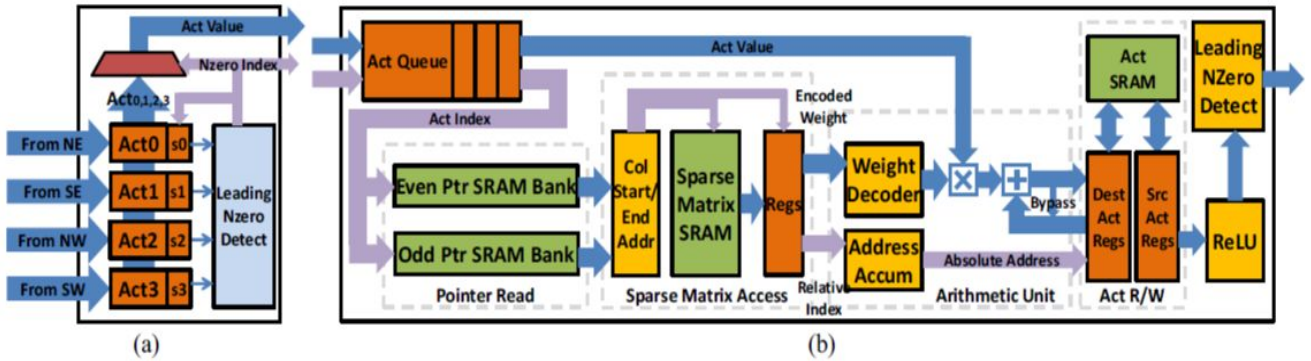
Parallelization



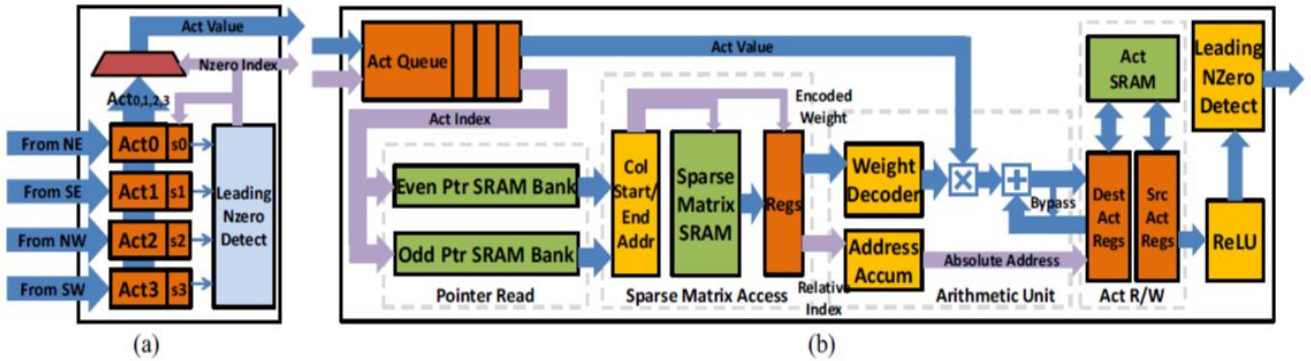
- Input activations and weights evenly distributed amongst an array of processing elements
- Partition by row to leverage sparsity
- Broadcast next non-zero element of input activation to all PEs
- All other computation kept local

Processing Element





- **Distributed Leading Non-Zero Detection**
  - Tree structure to detect and broadcast next non-zero input activation
- **Activation Queue**
  - Load balancing for uneven non-zero elements in weight columns
- **Pointer Read Unit**
  - Simultaneous read of column begin and column end
- **Sparse Matrix Read Unit**
  - Store 8-bit elements of (v, x)
  - 64-bit reads = 8 elements at a time



- **Arithmetic Unit**
  - Operate on a single (v, x) element & activation at head of queue
  - Table lookup to translate index v to real weight
  - Multiple-accumulate operation
- **Activation Read/Write**
  - Holds 64 16-bit activations (4K across 64 PEs)
  - Source/Destination exchange roles each layer -> No data movement
- **Central Control Unit (CCU)**
  - Communicates with Master
  - I/O Mode to load data & Computing Mode to produce results

Evaluation

- **Custom C++ Cycle Accurate Simulator**
  - Model RTL
  - Assist with design space exploration
- **Verilog RTL Implementation**
  - Area, power, critical path delay
- **Comparisons**
  - CPU - Intel Core i-7 5930k
  - GPU - NVIDIA GeForce GTX Titan X GPU
  - Mobile GPU - NVIDIA Tegra K1

Table III
BENCHMARK FROM STATE-OF-THE-ART DNN MODELS

Layer	Size	Weight%	Act%	FLOP%	Description
Alex-6	9216, 4096	9%	35.1%	3%	Compressed AlexNet [1] for large scale image classification
Alex-7	4096, 4096	9%	35.3%	3%	
Alex-8	4096, 1000	25%	37.5%	10%	
VGG-6	25088, 4096	4%	18.3%	1%	Compressed VGG-16 [3] for large scale image classification and object detection
VGG-7	4096, 4096	4%	37.5%	2%	
VGG-8	4096, 1000	23%	41.1%	9%	
NT-We	4096, 600	10%	100%	10%	Compressed NeuralTalk [7] with RNN and LSTM for automatic image captioning
NT-Wd	600, 8791	11%	100%	11%	
NTLSTM	1201, 2400	10%	100%	11%	

Area / Power / Energy

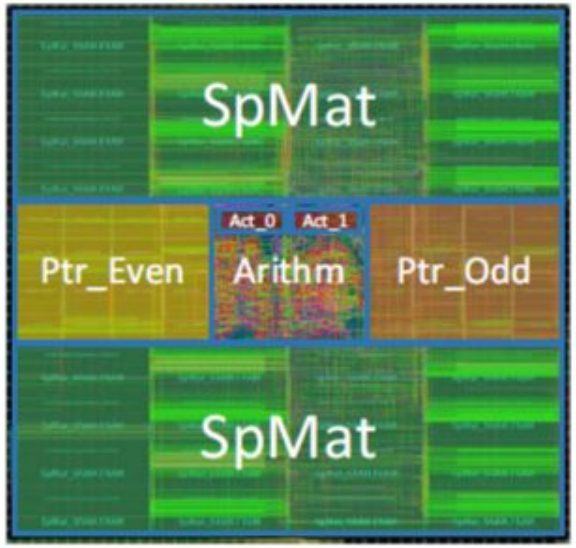


Figure 5. Layout of one PE in EIE under TSMC 45nm process.

Table II
THE IMPLEMENTATION RESULTS OF ONE PE IN EIE AND THE BREAKDOWN BY COMPONENT TYPE (LINE 3-7), BY MODULE (LINE 8-13). THE CRITICAL PATH OF EIE IS 1.15 NS

	Power (mW)	(%)	Area (μm^2)	(%)
Total	9.157		638,024	
memory	5.416	(59.15%)	594,786	(93.22%)
clock network	1.874	(20.46%)	866	(0.14%)
register	1.026	(11.20%)	9,465	(1.48%)
combinational	0.841	(9.18%)	8,946	(1.40%)
filler cell			23,961	(3.76%)
Act_queue	0.112	(1.23%)	758	(0.12%)
PtrRead	1.807	(19.73%)	121,849	(19.10%)
SpmatRead	4.955	(54.11%)	469,412	(73.57%)
ArithmUnit	1.162	(12.68%)	3,110	(0.49%)
ActRW	1.122	(12.25%)	18,934	(2.97%)
filler cell			23,961	(3.76%)

Energy: 24,000x less than CPU. 3,400x less than GPU. 2,700x less than Mobile GPU.

Performance

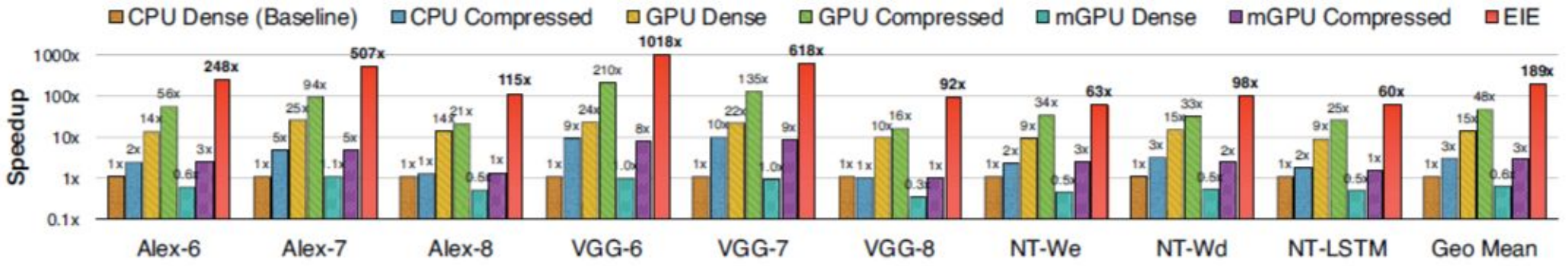


Figure 6. Speedups of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model. There is no batching in all cases.

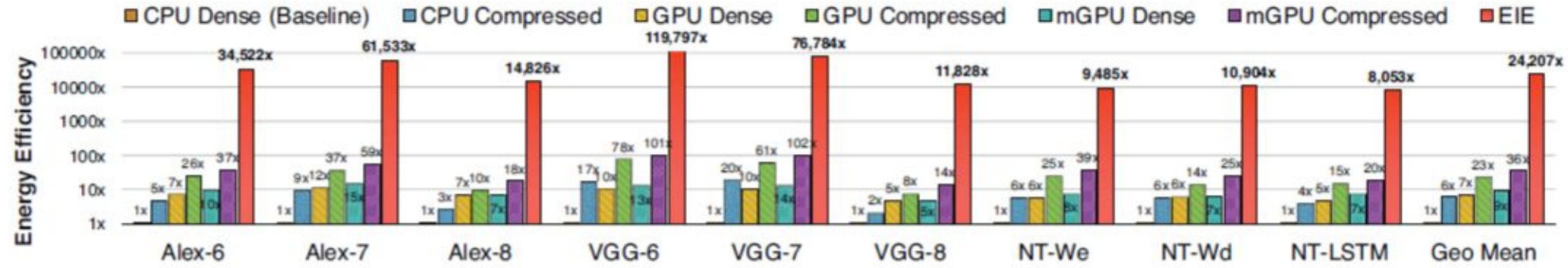
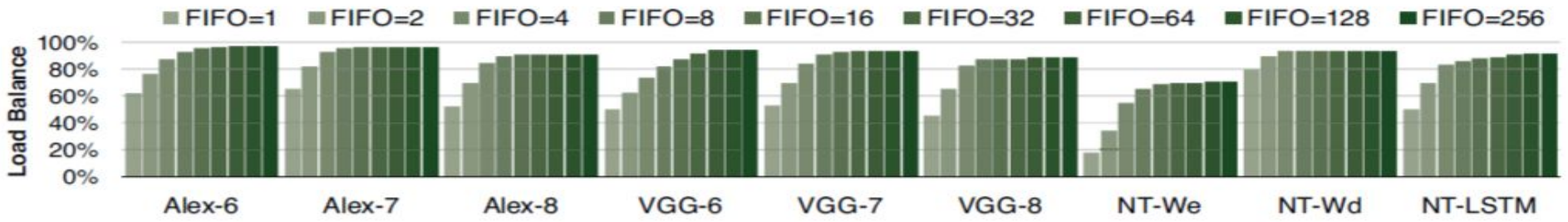


Figure 7. Energy efficiency of GPU, mobile GPU and EIE compared with CPU running uncompressed DNN model. There is no batching in all cases.

189x over CPU, 13x over GPU, 307x over Mobile GPU

Design Decisions



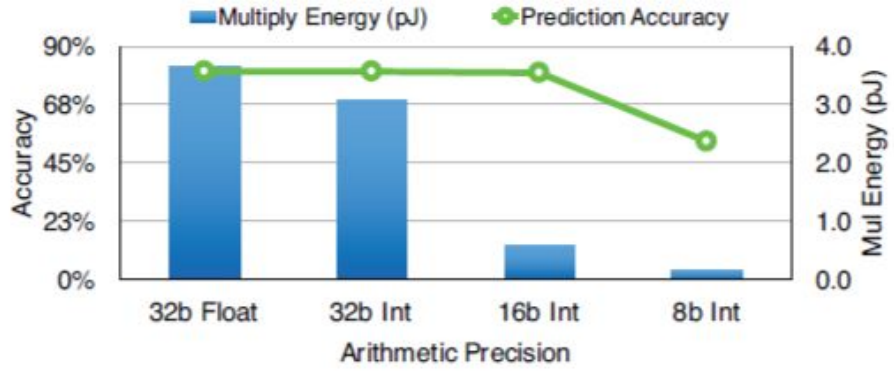
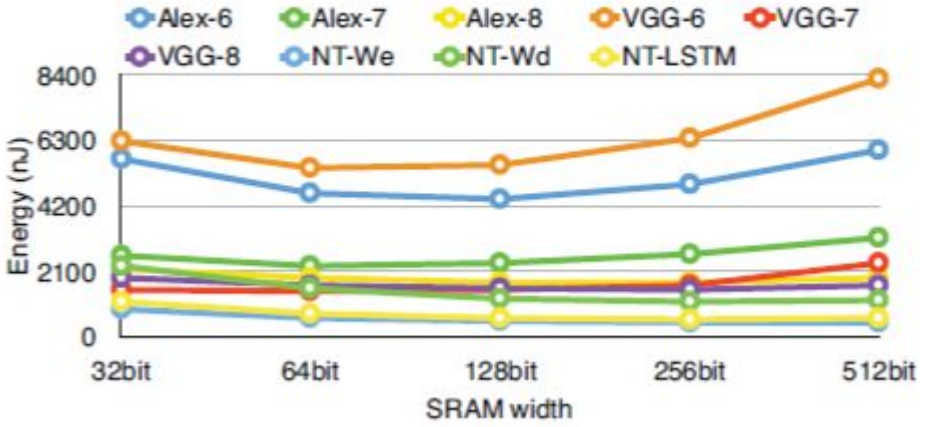


Figure 10. Prediction accuracy and multiplier energy with different arithmetic precision.



Scalability

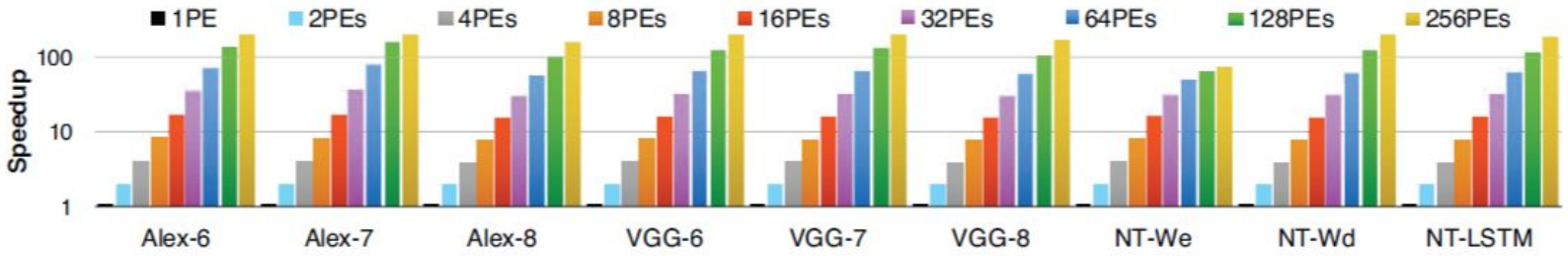


Figure 11. System scalability. It measures the speedups with different numbers of PEs. The speedup is near-linear.

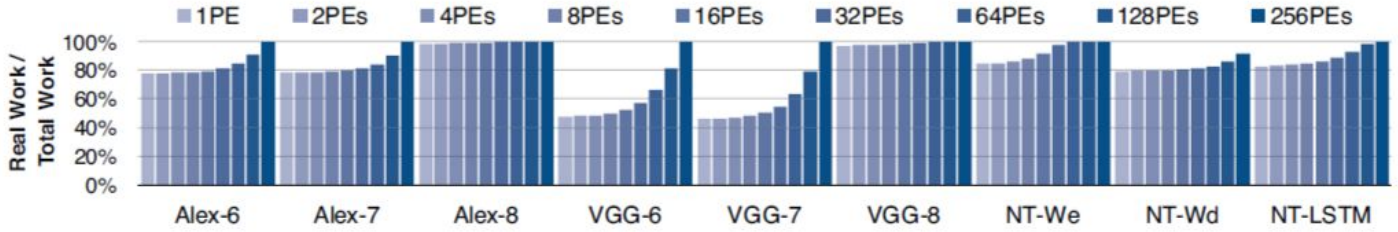


Figure 12. As the number of PEs goes up, the number of padding zeros decreases, leading to less padding zeros and less redundant work, thus better compute efficiency.

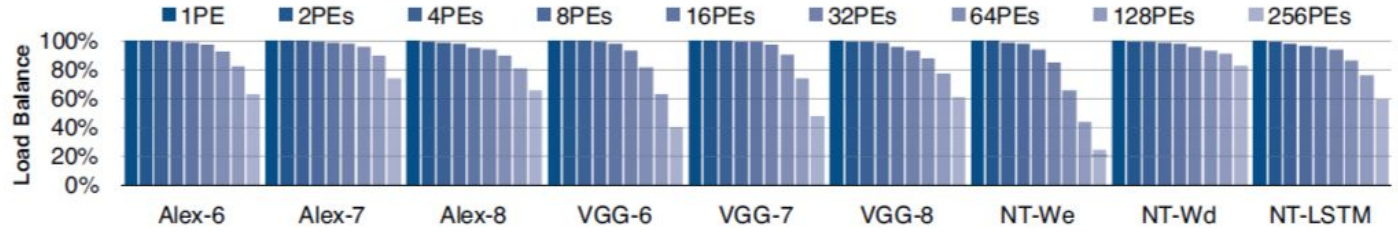


Figure 13. Load efficiency is measured by the ratio of stalled cycles over total cycles in ALU. More PEs lead to worse load balance, but less padding zeros and more useful computation.

Conclusion

- **Accelerate matrix-vector multiplication for fully connected layers**
  - Flexible to other operations too
- **Operate on Compressed Neural Networks**
  - Pruning, weight sharing, sparsity, quantization
- **Single PE: 1.6 GOPS in 0.64mm^2 with 9mW power dissipation**
- **64 PEs at 800MHz: 102 GOP/s equivalent to 3TOP/s for uncompressed network**
- **Scalable to over 256 PEs**