Harvard John A. Paulson
School of Engineering
and Applied Sciences

Don’t Use a Single Large Systolic Array, Use Many Small Ones Instead

H. T. Kung
Harvard University

Presentation at Workshop on ML for Systems at ISCA,

Phoenix, AZ, USA

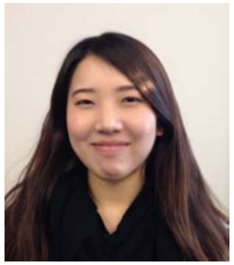
June 23, 2019

Outline

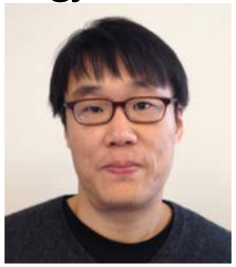
- Background: CNN, matmul, systolic arrays
- Issues of using a single large systolic array
- Solution approaches
  - Column combining
  - Maestro architecture for the use of many small systolic arrays
- Summary of next steps

Thanks to Great PhD Students in the Lab

Miriam Cha
(recently graduated; now a visiting scholar)
Marcus Comiter
Xin Dong
Youngjune Gwon
(graduated; now a visiting scholar)
Brad McDanel
(recently graduated; now a postdoc)
Philippe Tillet
Surat Teerapittayanon
(recently graduated)
James Yang
Sai Zhang

Miriam Cha

Marcus Comiter

Xin Dong

Youngjune Gwon

Brad McDanel

Philippe Tillet

Surat Teerapittayanon

James Yang

Sai Zhang

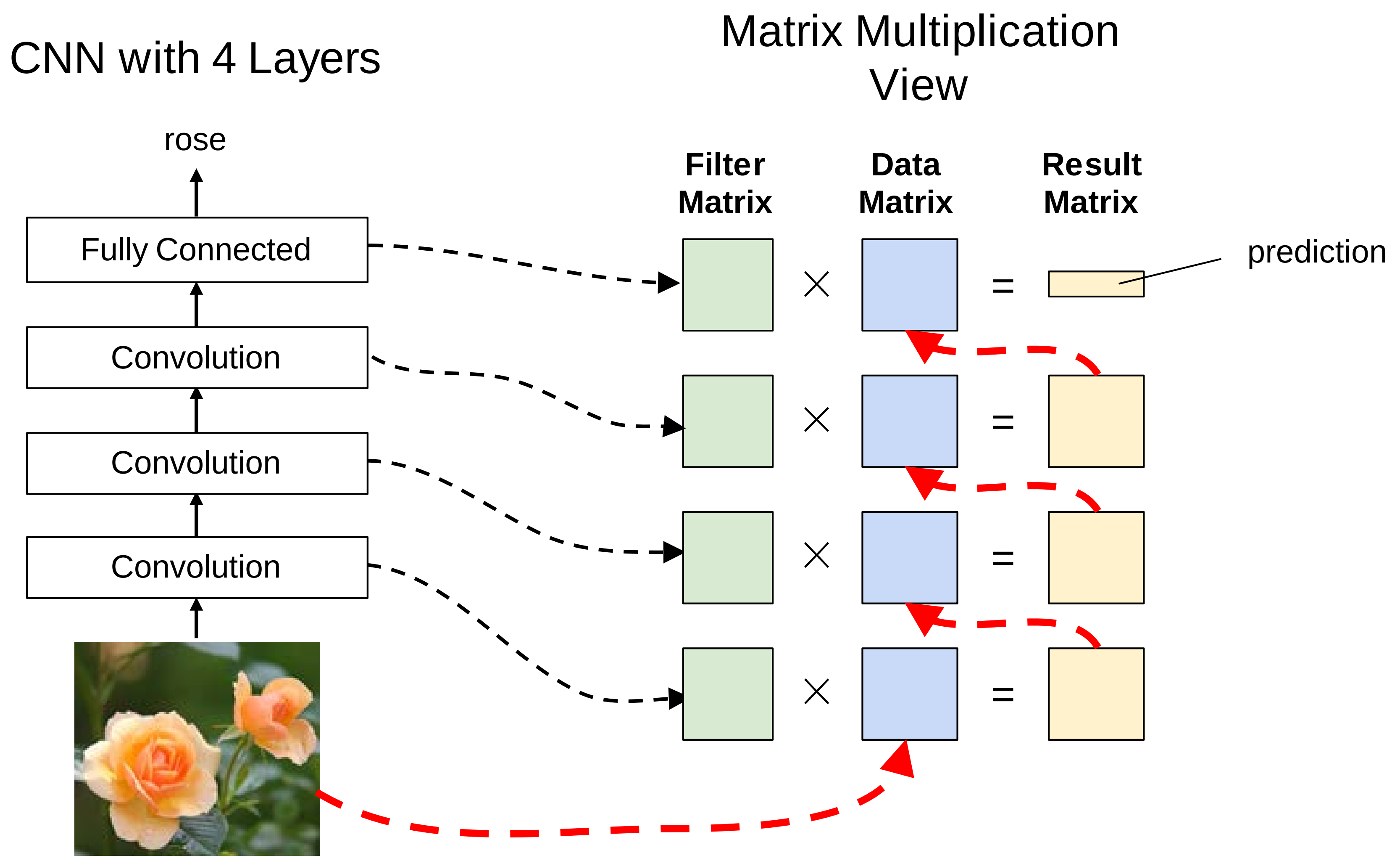
Red color: students who have contributed to work reported in this presentation

Two new PhD graduate students: Vikas Natesh and Andrew Sabot

Publications from Our Lab Related to this Presentation

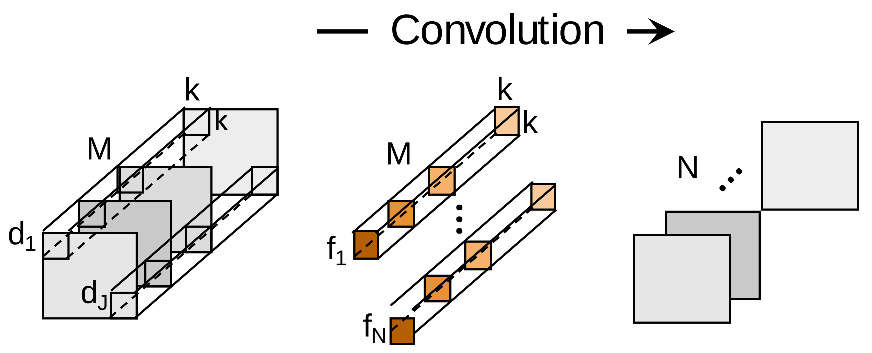
- [ASPLOS 2019] Packing **Sparse** Convolutional Neural Networks for Efficient **Systolic Array** Implementations Column Combining Under Joint Optimization
- [ICS 2019] Full-stack Optimization for Accelerating CNNs Using **Powers-of-Two** Weights with **FPGA** Validation
- [IEEE ASAP 2019] Maestro: A Memory-on-Logic Architecture for Coordinated Parallel Use of **Many Systolic Arrays**

Background: CNN Feedforward Pass as Series of Matrix Multiplications



More Precisely, Each Convolutional Layer as Matrix Multiplication

Computation of a convolutional layer



Data
(M input channels)

N Filters

Result
(N output feature maps)

Equivalent matrix multiplication

$$\begin{bmatrix} \text{—} \; f_1 \; \text{—} \\ \text{—} \; f_2 \; \text{—} \\ \vdots \\ \text{—} \; f_N \; \text{—} \end{bmatrix} \times \begin{bmatrix} | & | & & | \\ d_1 & d_2 & \cdots & d_J \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \text{—} \; r_1 \; \text{—} \\ \text{—} \; r_2 \; \text{—} \\ \vdots \\ \text{—} \; r_N \; \text{—} \end{bmatrix}$$

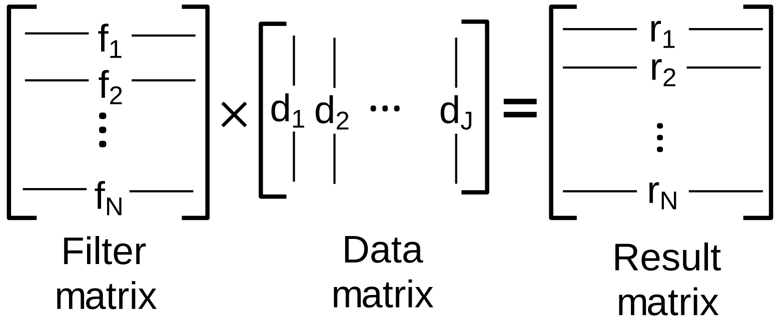
Filter matrix

Data matrix

Result matrix

Background: Using Systolic Array for Efficient Matrix Multiplication

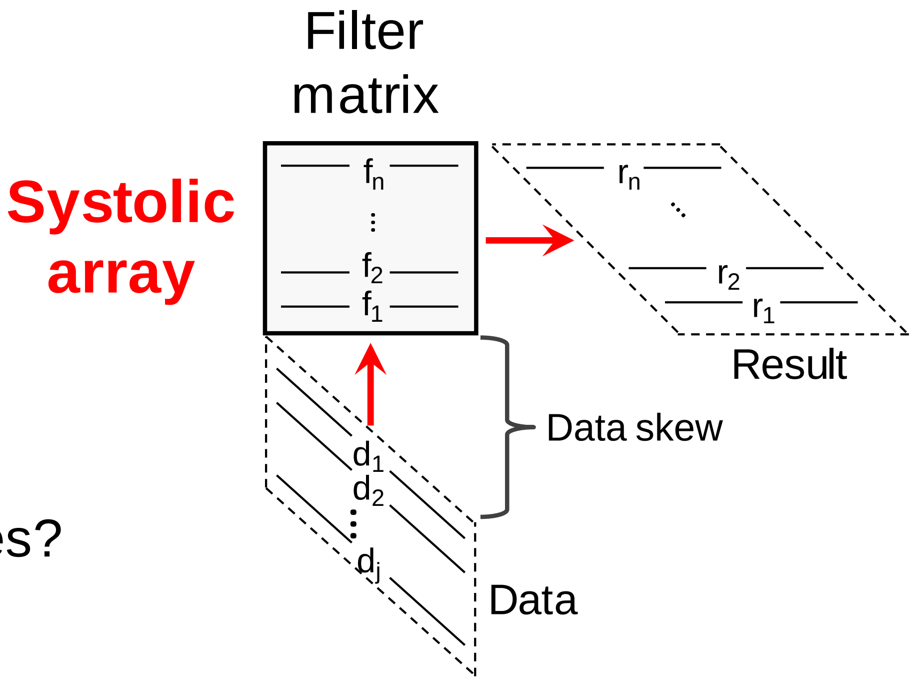
Matrix multiplication



[Kung and Leiserson 1979] VLSI Processor Arrays

[Kung 1982] Why Systolic Architectures?

Systolic array Implementation

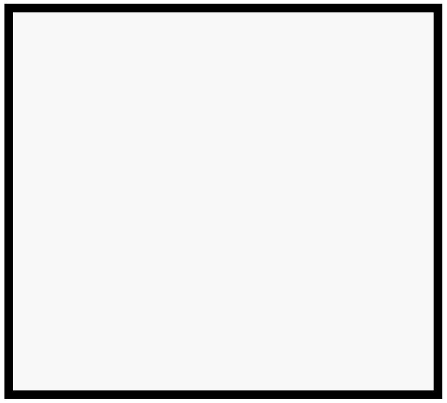


High efficiency due to: (1) regular design, (2) data flow architecture and (3) memory access reduction

Two Design Choices for Systolic Array Based Accelerators

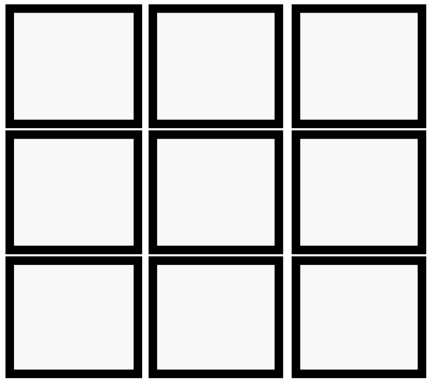
Option 1:

A single large systolic array



Option 2:

Many small systolic arrays



Problem of Using a Single Large Systolic Array: Under-utilization

- Issue 1: Large matrix may be sparse
- Issue 2: Application may have many matrix multiplications of various shapes and sizes to do

Expanding on Issue 1: Efficient CNNs Are Sparse

- We want to speed up a computation which is **already** efficient
- Efficient CNNs means fewer MAC operations in the computation, typically resulting from weight pruning
- This means filter matrices tend to be highly sparse
  - Moreover, weights can be **quantized**, even logarithmically (see powers-of-two weights in McDanel, Zhang, Kung and Dong [ICS 2019])

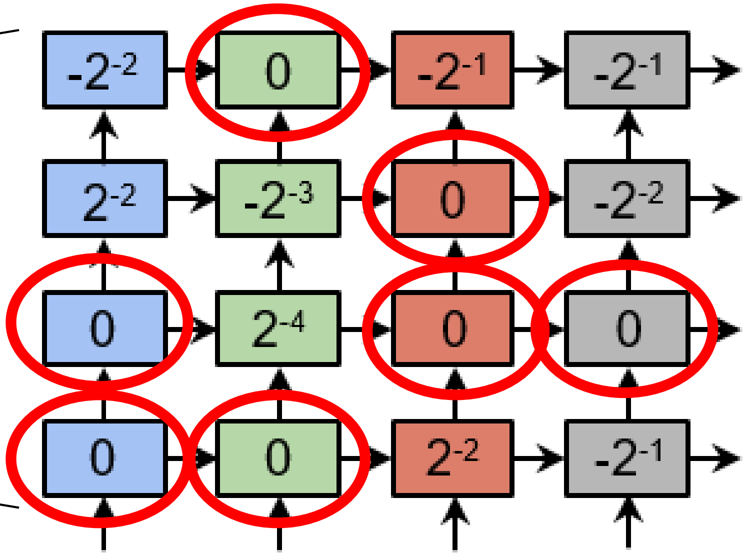
A Challenge: How not to Waste Systolic Cells for Zero-valued Weights

Sparse filter matrix

$$\begin{bmatrix} -2^{-2} & 0 & -2^{-1} & -2^{-1} \\ 2^{-2} & -2^{-3} & 0 & -2^{-2} \\ 0 & 2^{-4} & 0 & 0 \\ 0 & 0 & 2^{-2} & -2^{-1} \end{bmatrix}$$

(Streamlined CNNs, e.g., after pruning, tend to use many sparse filters)

Systolic array



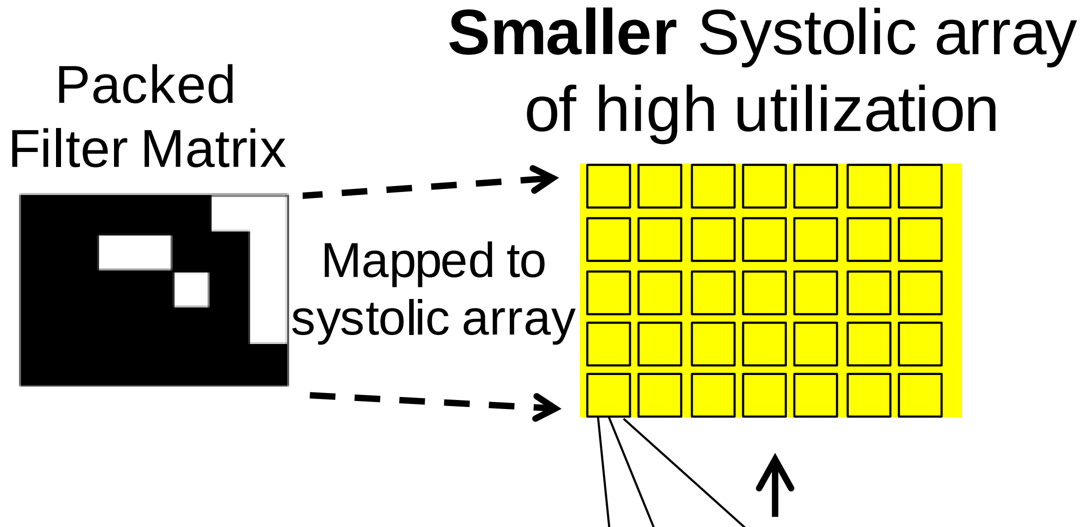
Systolic cells storing zero weights (circled) are wasteful

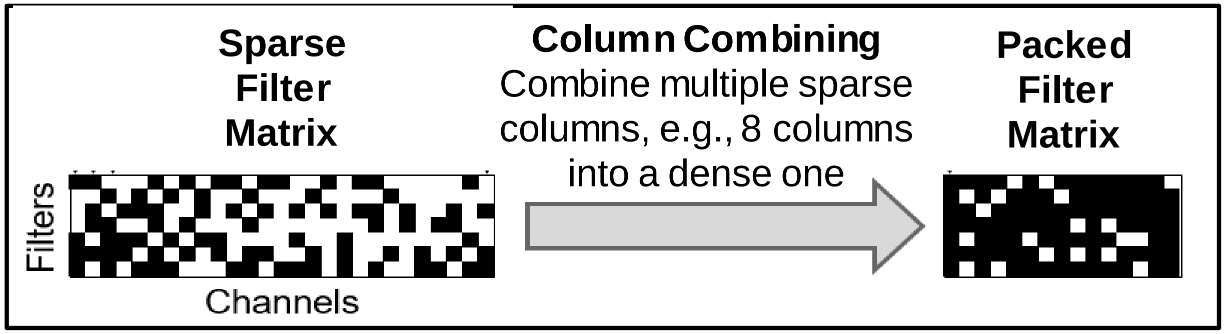
Goal: remove these wasteful cells **without** messing up data synchronization of the systolic array

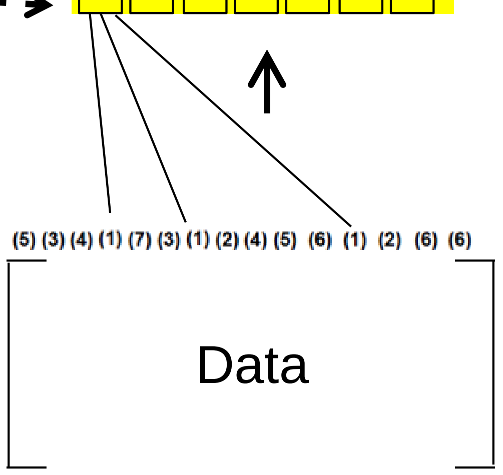
A Solution: Column Combining for Sparse Filter Matrix Kung, McDanel and Zhang [ASPLOS 2019]

Jointly optimize:

1. CNN **accuracy**
2. Systolic array **utilization**





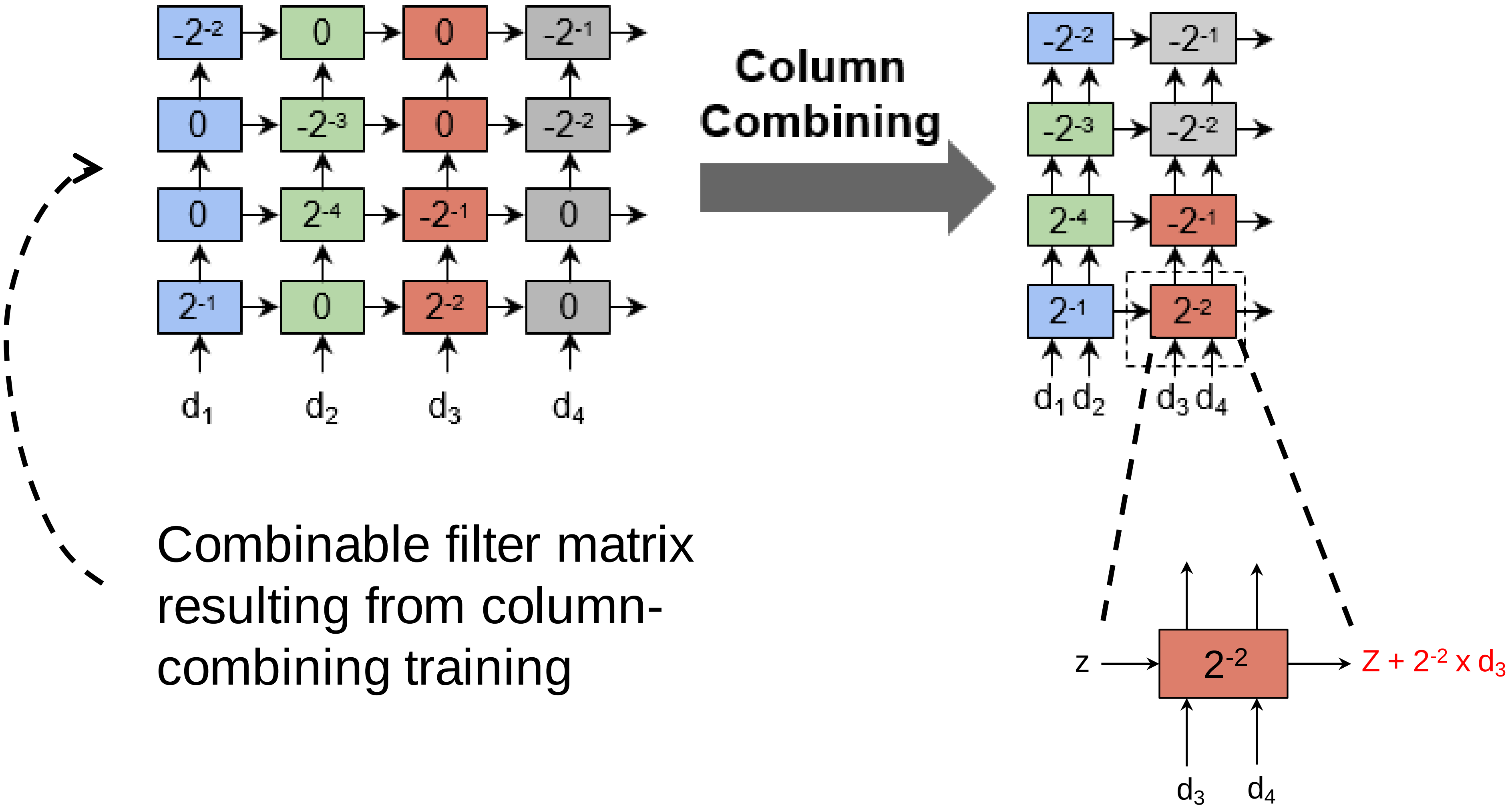


- For high packing density, in combining columns we allow **overlapping nonzero entries** for each row (e.g., up to 1.75 per row on average). We **prune** all of them except the one with the largest magnitude
- We **retrain** the remaining weights to bring up inference accuracy

Column Combining Illustration

(a) Conventional systolic array

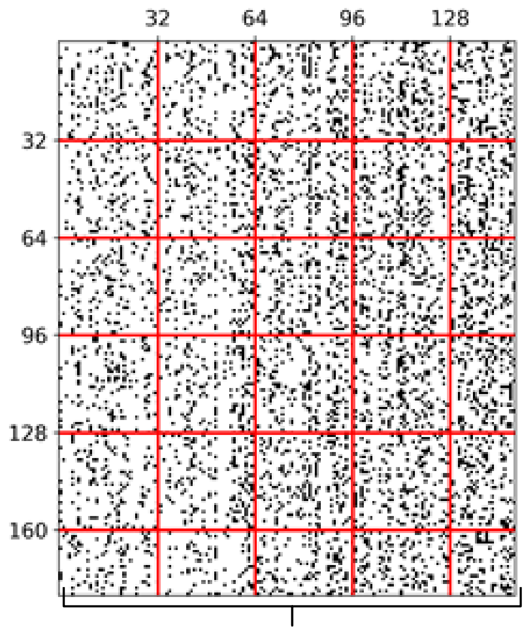
(b) Systolic array under column combining



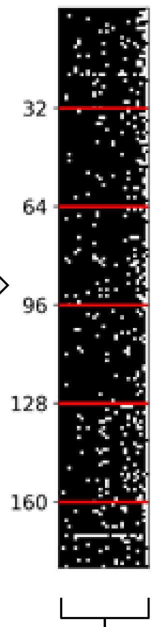
Combinable filter matrix resulting from column-combining training

By Packing Sparse CNNs, Column Combining Reduces # Required Tiles

Original sparse filter matrix

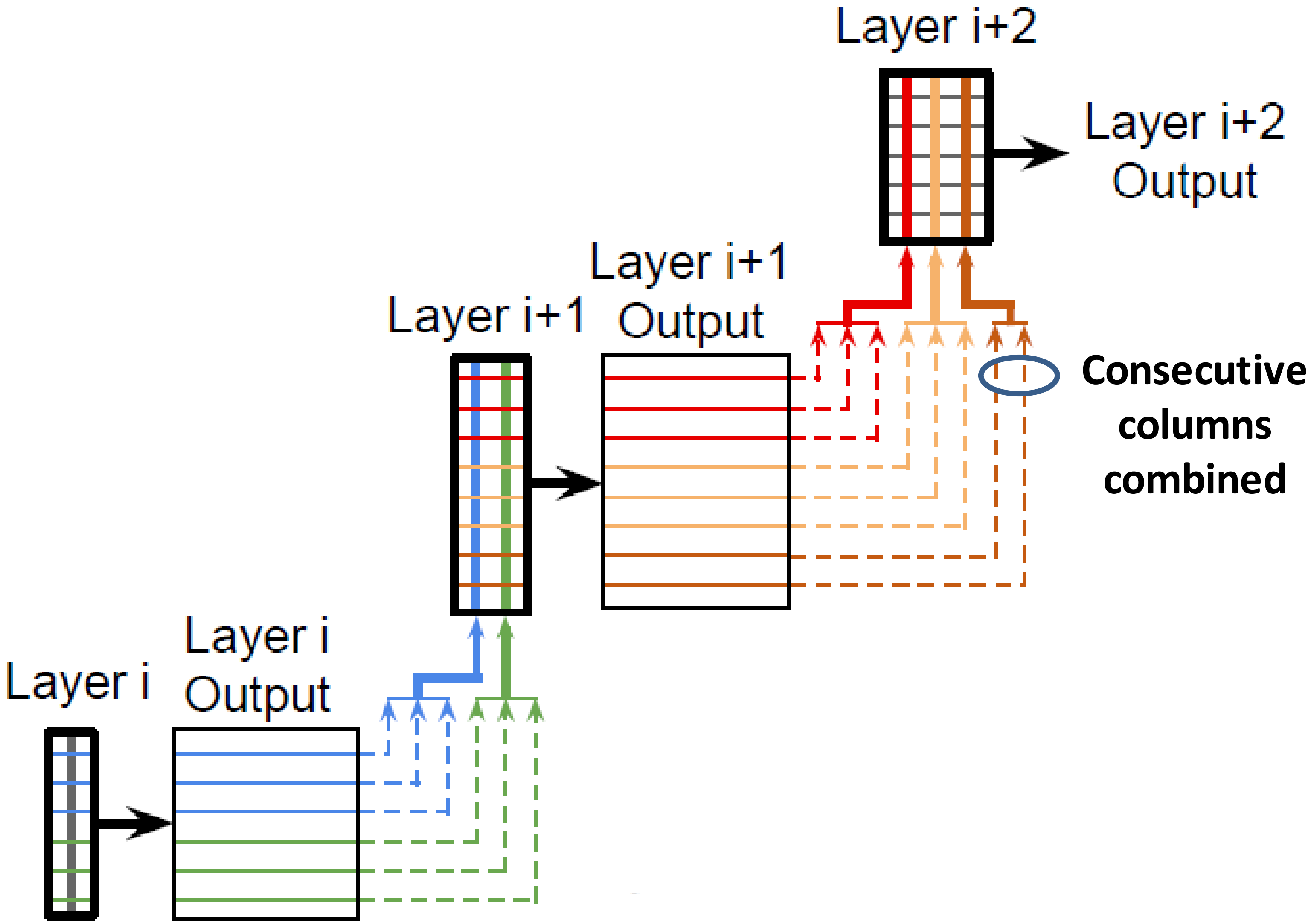


Column combining
(5x reduction in tiles)

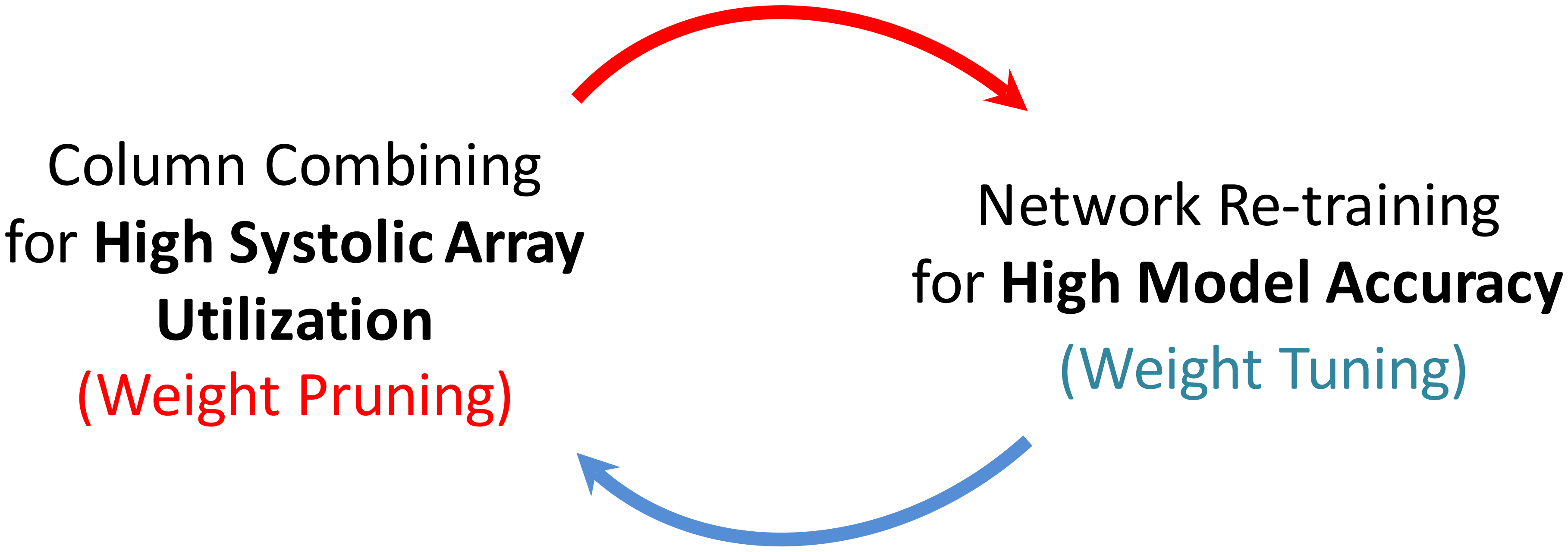


Packed filter matrix

Combining Columns Can Be Made Consecutive by Permuting Rows in the Filter Matrix of the Previous Layer



Column Combining: Co-design of Deep-learning Model and Parallel Processing Hardware to Make Them Fit Each Other

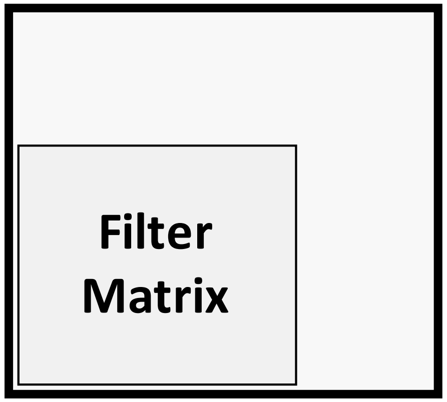

Column Combining
for **High Systolic Array Utilization**
(Weight Pruning)
Network Re-training
for **High Model Accuracy**
(Weight Tuning)

Problem of Using a Single Large Systolic Array: Under-utilization

- Issue 1: Large matrix may be sparse
- Issue 2: Application may have many matrix multiplications of various shapes and sizes to do

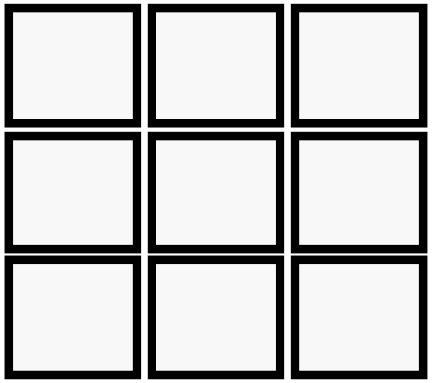
A Single Large Systolic Array vs. Many Small Ones

A single large systolic array



Problem:
under-utilization

Many small systolic arrays

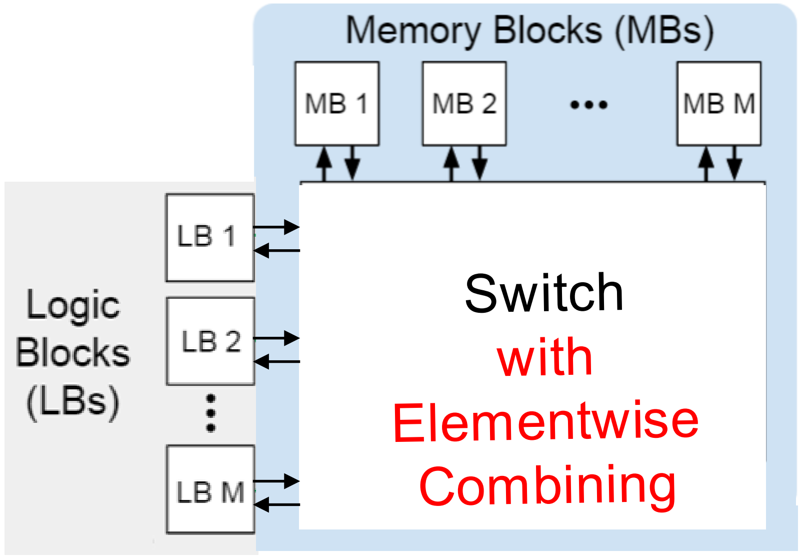


High-utilization possible

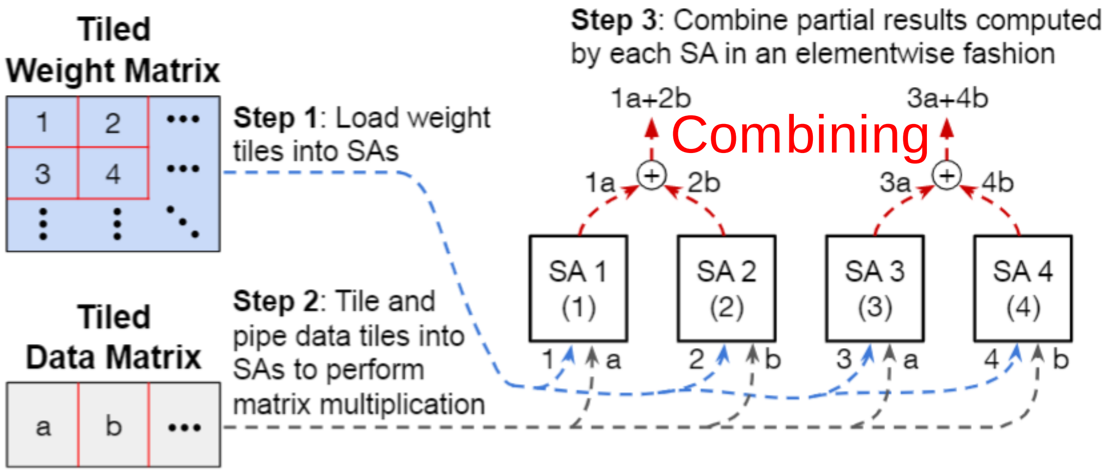
Challenges:
(1) **scheduling** these arrays for matrix computation of various shapes and sizes, and (2) **inter-array communication** via memory banks

Hardware Abstraction for Tiled Matrix Computations

Hardware abstraction



"Tile and pipe" computation model

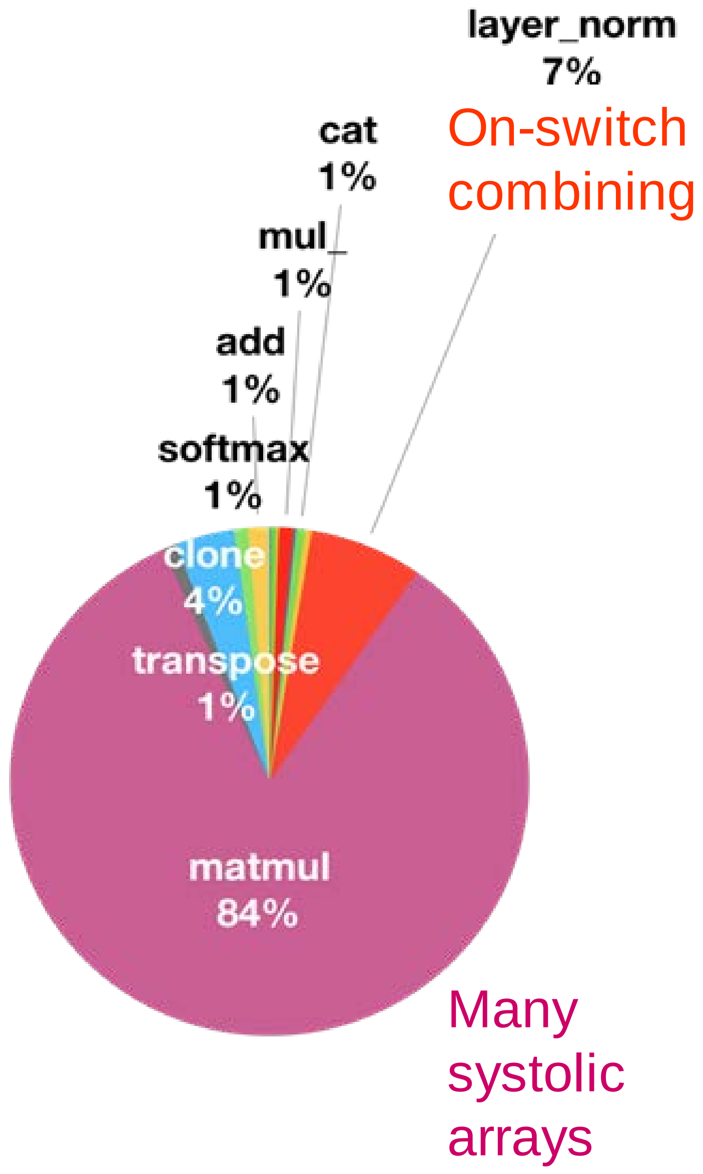

Tiled Weight Matrix
1
2
3
4
Step 1: Load weight tiles into SAs
Tiled Data Matrix
a
b
Step 2: Tile and pipe data tiles into SAs to perform matrix multiplication
Step 3: Combine partial results computed by each SA in an elementwise fashion
1a+2b
3a+4b
Combining
1a
2b
3a
4b
SA 1 (1)
SA 2 (2)
SA 3 (3)
SA 4 (4)

Reduced memory access

Latency Profiling of a Transformer Workload

Kung, McDanel, Zhang and Dong [ASAP 2019]

- We have profiled the inference performance on a GPU for a TensorFlow implementation of a 100M-parameter Transformer model
- The average translation time from English to German sentences is **0.945 seconds**, with a breakdown shown on the right
- We want to substantially **reduce this latency** with (1) many systolic arrays and (2) on-switch combining (see Maestro system on a later slide)
- Under a new DARPA-sponsored project, we begin to investigate **low-power** approaches based on optoelectronic approaches

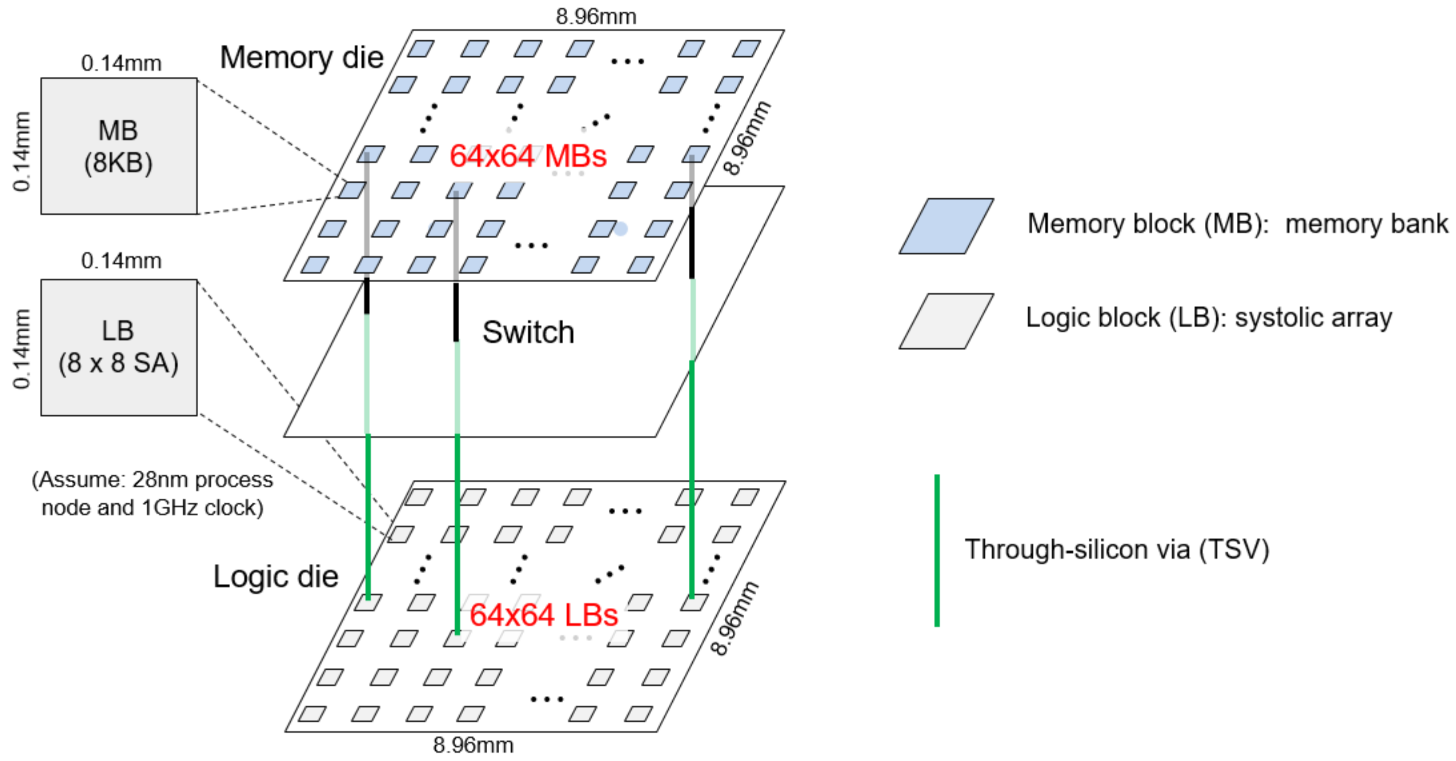


Matrices of Various Shapes and Sizes Used

- w is length of the input sentence. The average length of English sentences is 19. The length may vary a lot
- The chart on the right is for just one of the 8 Encoder Layers
- A Decoder Layer has a similar pattern. Note that Decoder Layer is only needed by some tasks such as translation
- Both BERT and GPT-1/2 only have Encoder Layers

Order	Mul	# times
1	(w,512) @ (512,512)	3
2	(w,64) @ (64,64)	8
3	(w,w) @ (w,64)	8
4	(w,512) @ (512,512)	1
5	(w,512) @ (512,2048)	1
6	(w,2048, 2048,512)	1

Harvard's Maestro Memory-on-Logic Architecture for Use of Many Systolic Arrays

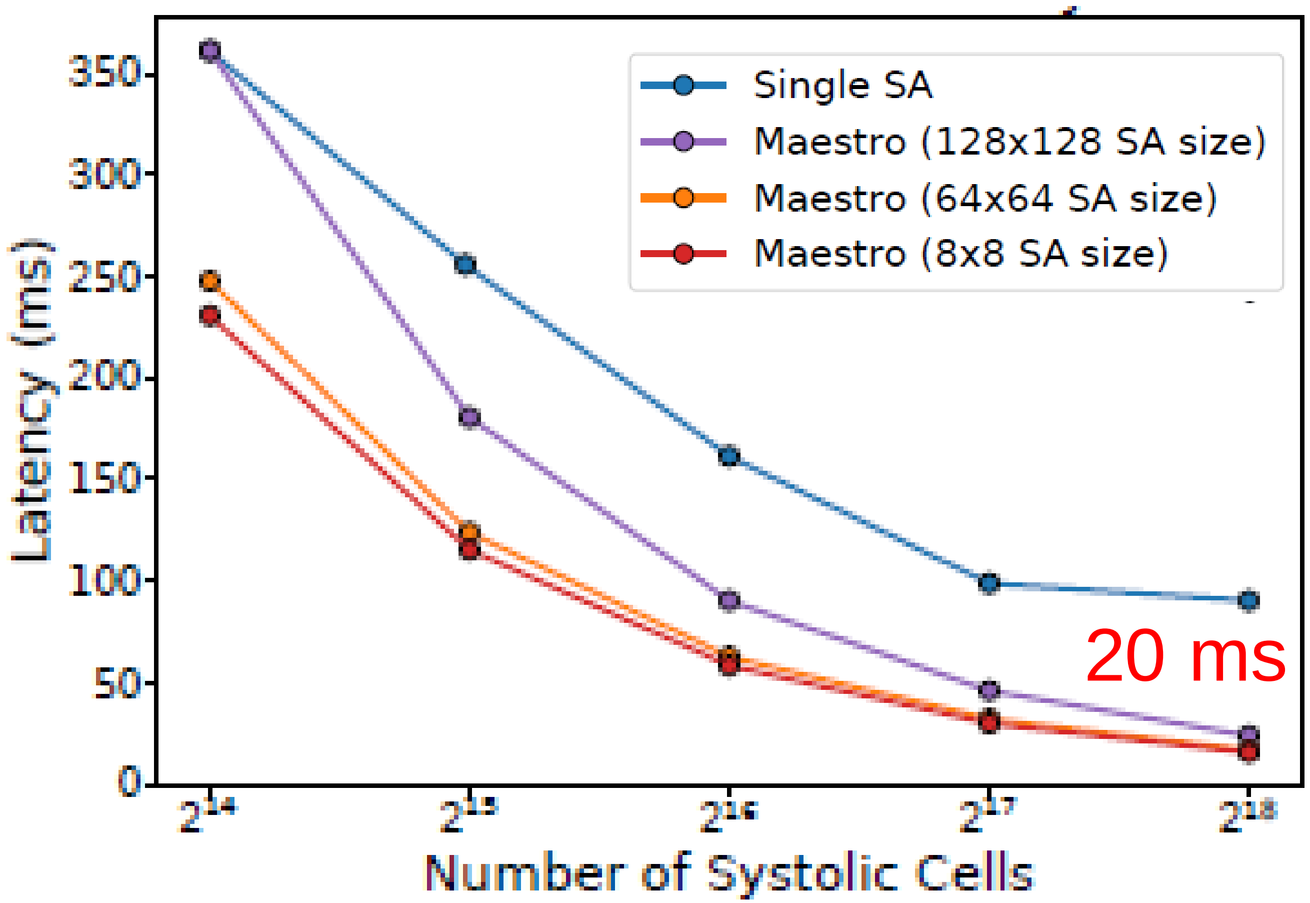

0.14mm
0.14mm
MB
(8KB)
Memory die
8.96mm
64x64 MBs
8.96mm
0.14mm
0.14mm
LB
(8 x 8 SA)
Switch
(Assume: 28nm process node and 1GHz clock)
Logic die
64x64 LBs
8.96mm
8.96mm
Memory block (MB): memory bank
Logic block (LB): systolic array
Through-silicon via (TSV)

Baseline Maestro

Preliminary study

- [IEEE ASAP 2019]: "Maestro: A Memory-on-Logic Architecture for Coordinated Parallel Use of Many Systolic Arrays"
- Initial FPGA prototyping underway for a 2D Maestro

Simulated Maestro's Performance on 100M-parameter Transformer



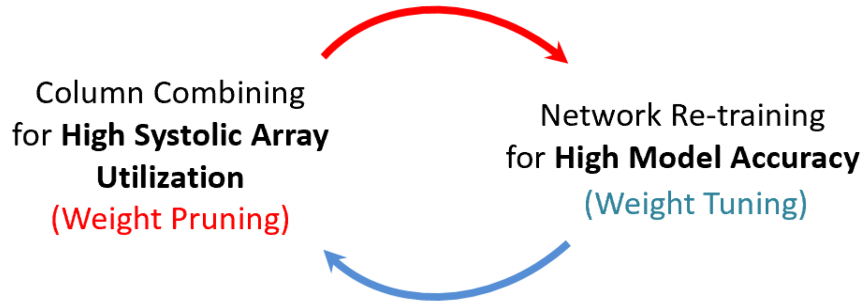
A large reduction in latency achieved with 64 small 64x64 systolic arrays

Optimization: Minimizing and Parallelizing Memory Access

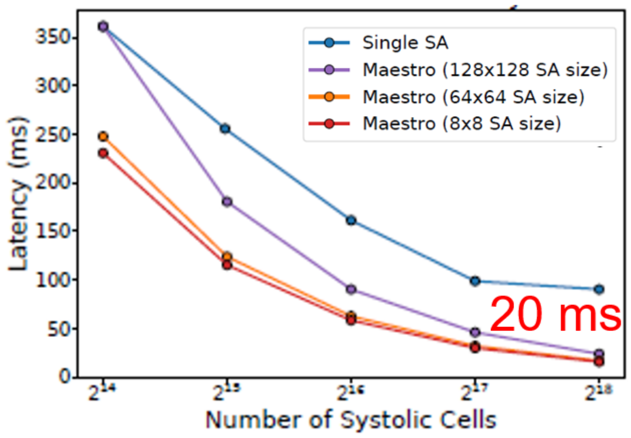
- Pre-loading of model parameters (weights) to allow a loaded data block to finish all its computations with model weights without having to be loaded again in the future
- Parallel reductions using multiple systolic arrays with on-switch combining circuitry and buffering
- Overlapping the computation time for the current data block with the loading time for the next data block
- Outputting computation results to memory banks where data for the next layer’s computation can be fetched in parallel

Summary and Next Steps

(1) Co-design to allow high-utilization systolic arrays for sparse CNN


Column Combining
for **High Systolic Array Utilization**
(Weight Pruning)
Network Re-training
for **High Model Accuracy**
(Weight Tuning)

(2) Use of many small systolic arrays wins



Next steps:

- FPGA implementation of Maestro as an experimental platform
- Addressing dynamic sparse data in training
- MLIR dialect for optimized scheduling of many systolic arrays