2020 IEEE International Symposium on Circuits and Systems
Virtual, October 10-21, 2020

A Systolic Dataflow Based Accelerator for CNNs

Saptarsi Das, Arnab Roy, Kiran Kolar Chandrasekharan, Ankur Deshwal
SAIT, Samsung R&D Institute India-Bangalore, India
Email: saptarsi.das, arnab.roy, kiran.kc, a.deshwal@samsung.com
Sehwan Lee
SAIT, Samsung Electronics, South Korea
Email: sehwan.b.lee@samsung.com

2020 IEEE International Symposium on Circuits and Systems
Virtual, October 10-21, 2020

Content

- Motivation & Background
- Design & Implementation of the Proposed Accelerator
- Experimental Results
- Conclusions

Motivation & Background

Why Accelerate CONV?

- Convolution Neural Networks (CNN) are ubiquitous in modern AI systems.
- Power efficient execution of CNN models is crucial especially in mobile phones and other hand held devices.
- In CNN models, the most compute intensive parts are convolution operations (CONV).

Motivation & Background

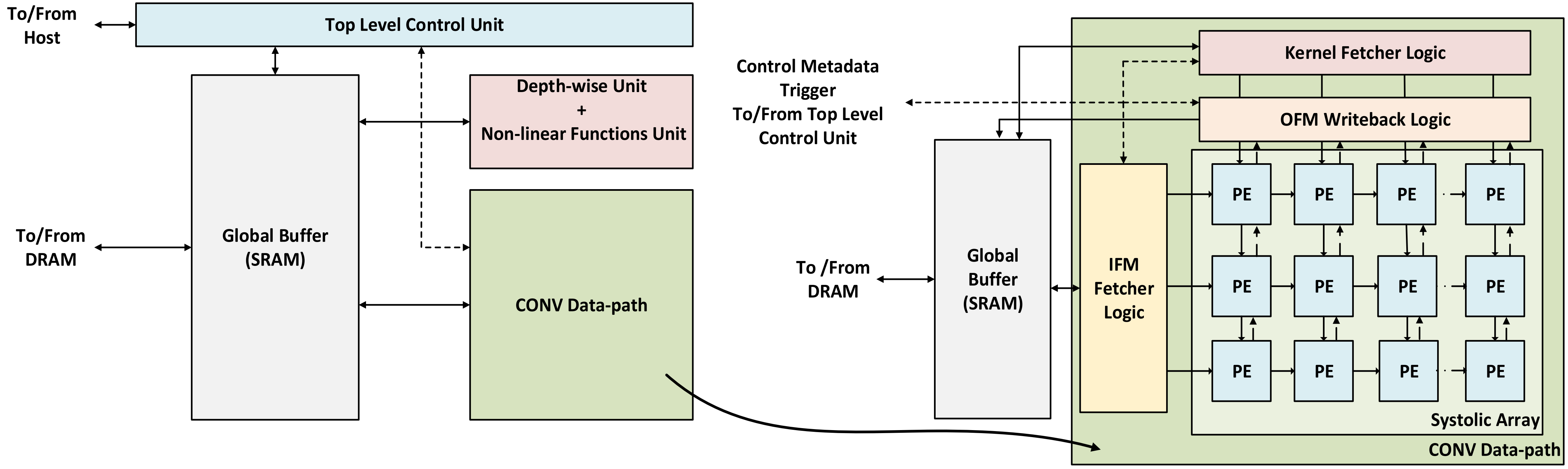
Why Use Systolic Array?

- Systolic arrays have a long history
- Have largely been off the mainstream
- Offers scalability, structural simplicity, ability to exploit parallelism and data reuse
- Renewed interest as candidate architecture for neural net accelerators

Why A New Systolic Array?

- Typically systolic arrays use scalar Processing Elements (PE)
- Cost of accumulation is high
- To mitigate this we propose a systolic array with inner-product units as the PEs

High Level View of the Accelerator

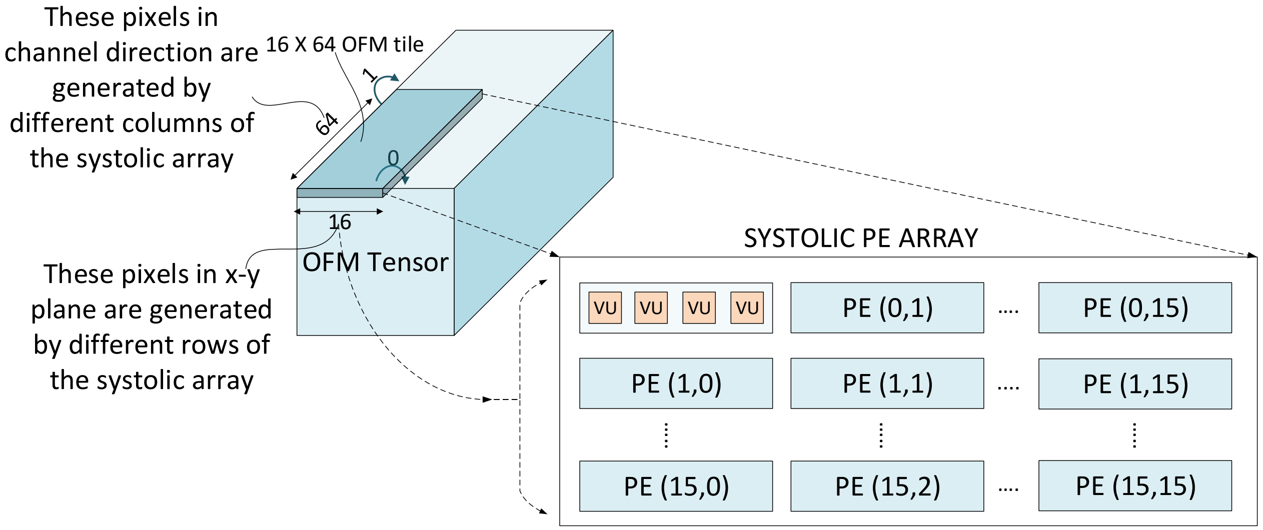


Primary Components of the Accelerator

- A Top Level Control Unit
- A Depth-wise Unit + Non-Linear Functions Unit
- A CONV Data-path
- A Global SRAM Buffer

Dataflow of CONV

- We follow an Output Stationary Traversal (OS)
- The Output Feature Map (OFM) 3D tensor is tiled as shown in Fig 1.
- Fig 2. shows delivery of Input Feature Map (IFM) & kernel into the systolic array



These pixels in channel direction are generated by different columns of the systolic array
16 X 64 OFM tile
1
64
0
16
OFM Tensor
These pixels in x-y plane are generated by different rows of the systolic array
SYSTOLIC PE ARRAY
VU
VU
VU
VU
PE (0,1)
....
PE (0,15)
PE (1,0)
PE (1,1)
....
PE (1,15)
PE (15,0)
PE (15,2)
....
PE (15,15)

```
for ofm_ch = 0 to C, stride = 64              //Loop level 1 – traversing OFM channels
    for ofm_px = 0 to HxW, stride = 16        //Loop level 0 – traversing OFM pixels
        systolic_execution()                  //Generation of 64x16 OFM tile
```

Fig. 1

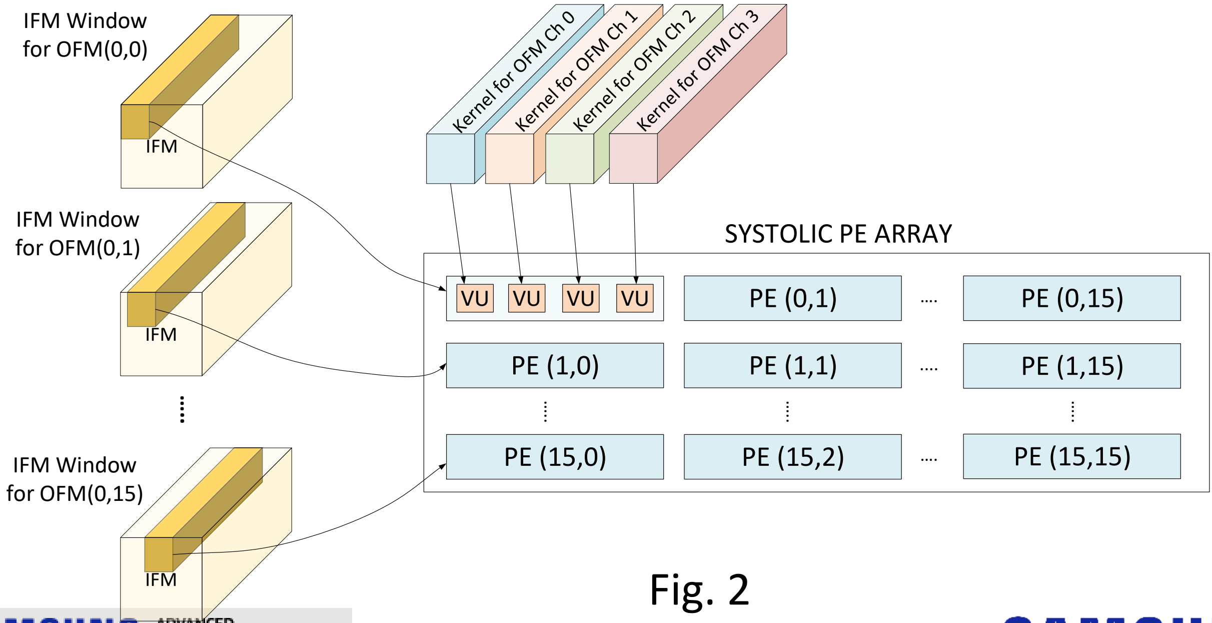


Fig. 2

- The IFM streams flow through a rows and get reused
- The kernel streams flow through columns and get reused

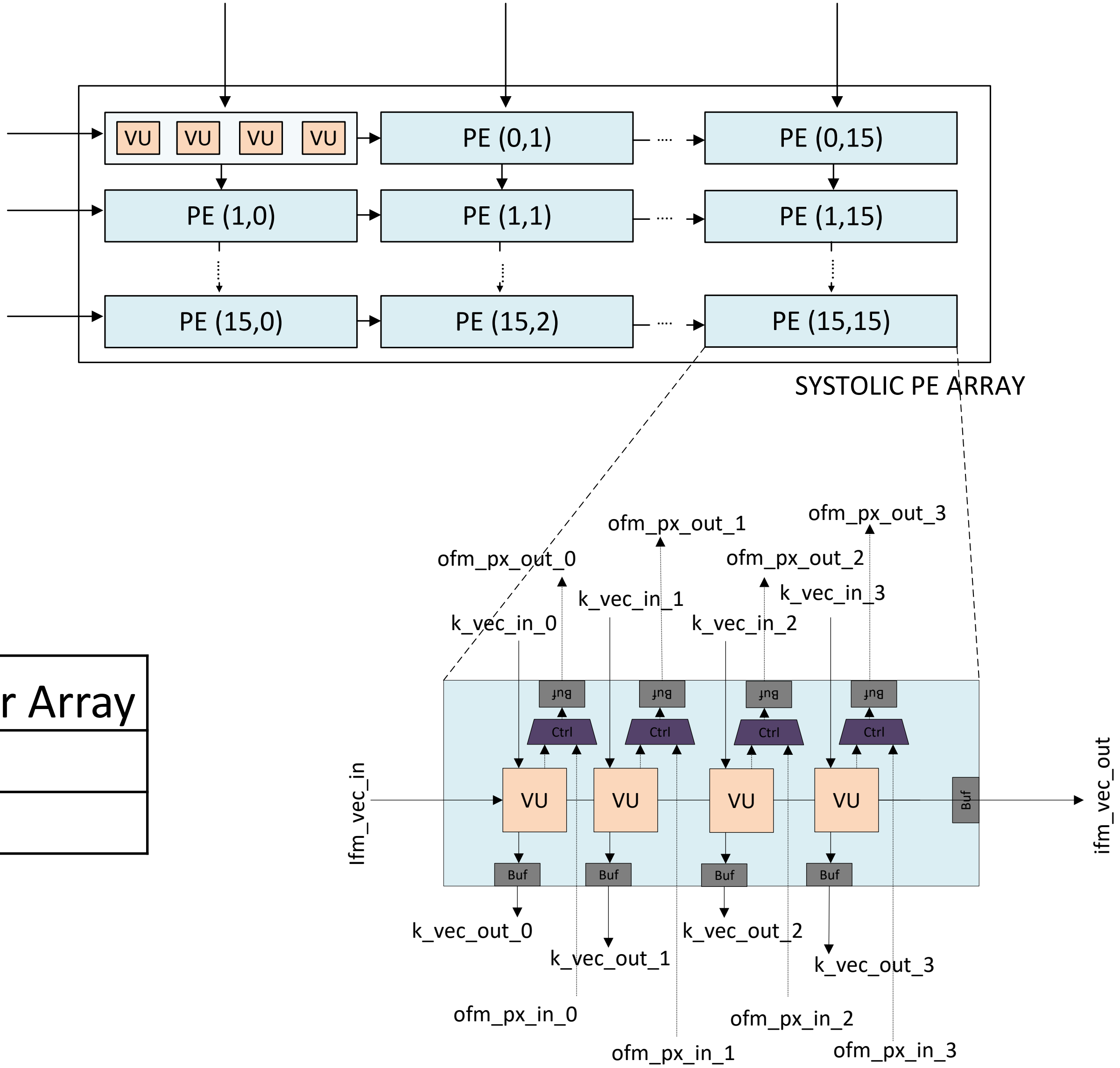
Features of the Accelerator

- Hierarchical vector units in PEs
- Channel-major data layout
- Overlapping of computation and communication
- Software-controlled scratch-pad SRAM with configurable logical buffers

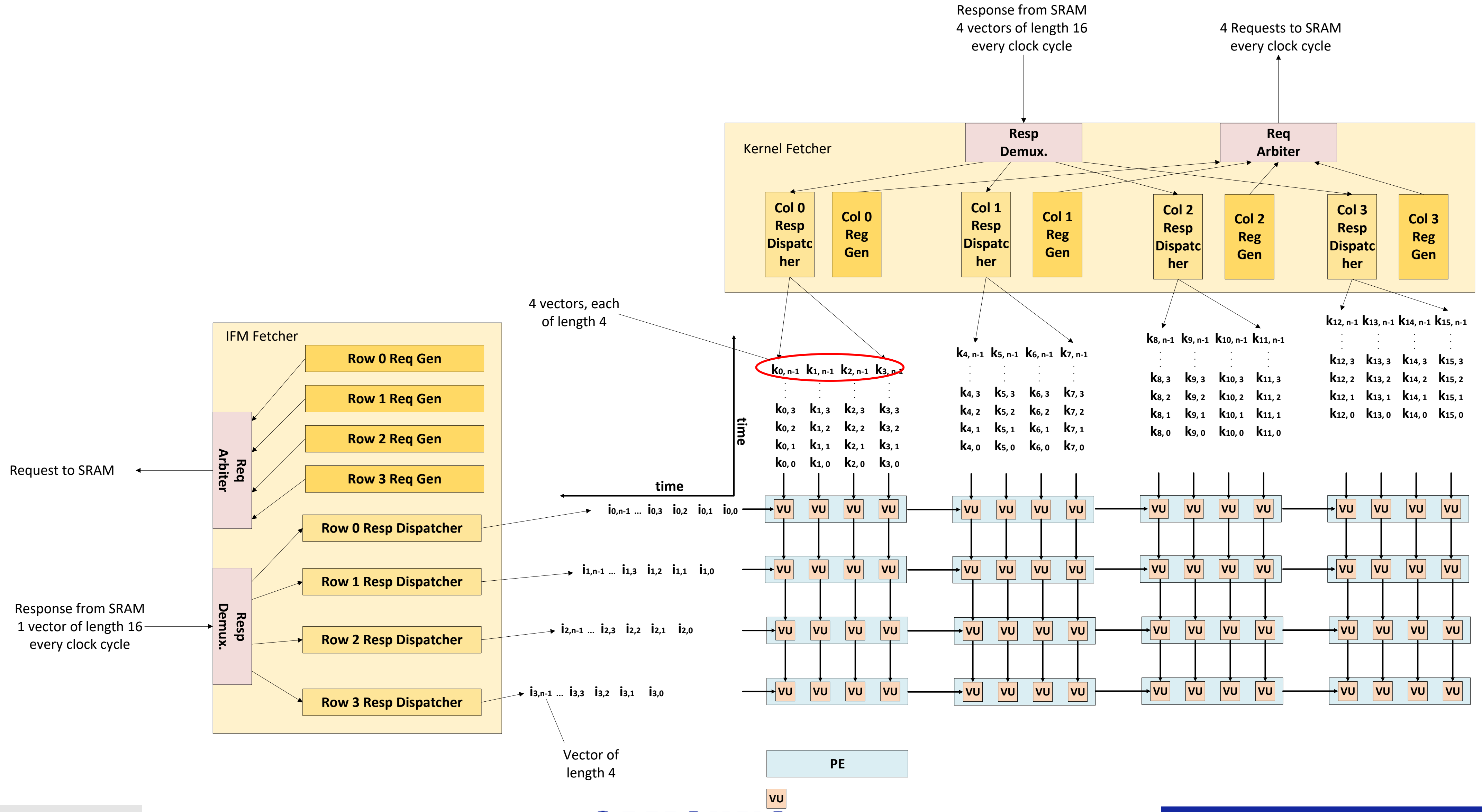
Hierarchical vector units in PEs

- Each PE houses a multiplicity of vector units (VU)
- We implement two systolic arrays using Samsung 10 nm library (at 800 MHz)
    - One with scalar PEs
    - One with PEs consisting of 4 VUs. Each VU performs inner-product on vectors of length 4
    - Peak throughput of both arrays = 4096 INT8 Multiply-Accumulate (MAC)/cycle

	Scalar Array	Hierarchical Vector Array
Power-Efficiency (TOPs/W)	7.1	14.2
Area-Efficiency (TOPs/mm^2)	10.7	21.8



IFM & Kernel Dataflow through a 4x4 Array


Response from SRAM
4 vectors of length 16
every clock cycle
4 Requests to SRAM
every clock cycle
Kernel Fetcher
Resp Demux.
Req Arbiter
Col 0 Resp Dispatcher
Col 0 Reg Gen
Col 1 Resp Dispatcher
Col 1 Reg Gen
Col 2 Resp Dispatcher
Col 2 Reg Gen
Col 3 Resp Dispatcher
Col 3 Reg Gen
4 vectors, each
of length 4
$k_{0,n-1}$ $k_{1,n-1}$ $k_{2,n-1}$ $k_{3,n-1}$
$k_{0,3}$ $k_{1,3}$ $k_{2,3}$ $k_{3,3}$
$k_{0,2}$ $k_{1,2}$ $k_{2,2}$ $k_{3,2}$
$k_{0,1}$ $k_{1,1}$ $k_{2,1}$ $k_{3,1}$
$k_{0,0}$ $k_{1,0}$ $k_{2,0}$ $k_{3,0}$
$k_{4,n-1}$ $k_{5,n-1}$ $k_{6,n-1}$ $k_{7,n-1}$
$k_{4,3}$ $k_{5,3}$ $k_{6,3}$ $k_{7,3}$
$k_{4,2}$ $k_{5,2}$ $k_{6,2}$ $k_{7,2}$
$k_{4,1}$ $k_{5,1}$ $k_{6,1}$ $k_{7,1}$
$k_{4,0}$ $k_{5,0}$ $k_{6,0}$ $k_{7,0}$
$k_{8,n-1}$ $k_{9,n-1}$ $k_{10,n-1}$ $k_{11,n-1}$
$k_{8,3}$ $k_{9,3}$ $k_{10,3}$ $k_{11,3}$
$k_{8,2}$ $k_{9,2}$ $k_{10,2}$ $k_{11,2}$
$k_{8,1}$ $k_{9,1}$ $k_{10,1}$ $k_{11,1}$
$k_{8,0}$ $k_{9,0}$ $k_{10,0}$ $k_{11,0}$
$k_{12,n-1}$ $k_{13,n-1}$ $k_{14,n-1}$ $k_{15,n-1}$
$k_{12,3}$ $k_{13,3}$ $k_{14,3}$ $k_{15,3}$
$k_{12,2}$ $k_{13,2}$ $k_{14,2}$ $k_{15,2}$
$k_{12,1}$ $k_{13,1}$ $k_{14,1}$ $k_{15,1}$
$k_{12,0}$ $k_{13,0}$ $k_{14,0}$ $k_{15,0}$
time
time
IFM Fetcher
Row 0 Req Gen
Row 1 Req Gen
Row 2 Req Gen
Row 3 Req Gen
Req Arbiter
Request to SRAM
Row 0 Resp Dispatcher
Row 1 Resp Dispatcher
Row 2 Resp Dispatcher
Row 3 Resp Dispatcher
Resp Demux.
Response from SRAM
1 vector of length 16
every clock cycle
$i_{0,n-1}$ … $i_{0,3}$ $i_{0,2}$ $i_{0,1}$ $i_{0,0}$
$i_{1,n-1}$ … $i_{1,3}$ $i_{1,2}$ $i_{1,1}$ $i_{1,0}$
$i_{2,n-1}$ … $i_{2,3}$ $i_{2,2}$ $i_{2,1}$ $i_{2,0}$
$i_{3,n-1}$ … $i_{3,3}$ $i_{3,2}$ $i_{3,1}$ $i_{3,0}$
Vector of
length 4
VU
PE
VU

Channel-major Data Layout

- We employ a channel-major data layout
- The IFM and kernel tensors are fetched as 1x1x64 vectors (in a 16x16 array)
- We measure memory read efficiency as #Bytes Used/#Bytes Read
- We set the length of channel vector in channel-major storage to 64 and dimension of 2D tile to 4x4 in the XY-major storage.

Network Models	Memory read efficiency	
	Channel-major	XY-major
InceptionV3	78.89%	17.79%
Resnet50	99.13%	10.88%
Inception-Resnet	90.63%	15.96%

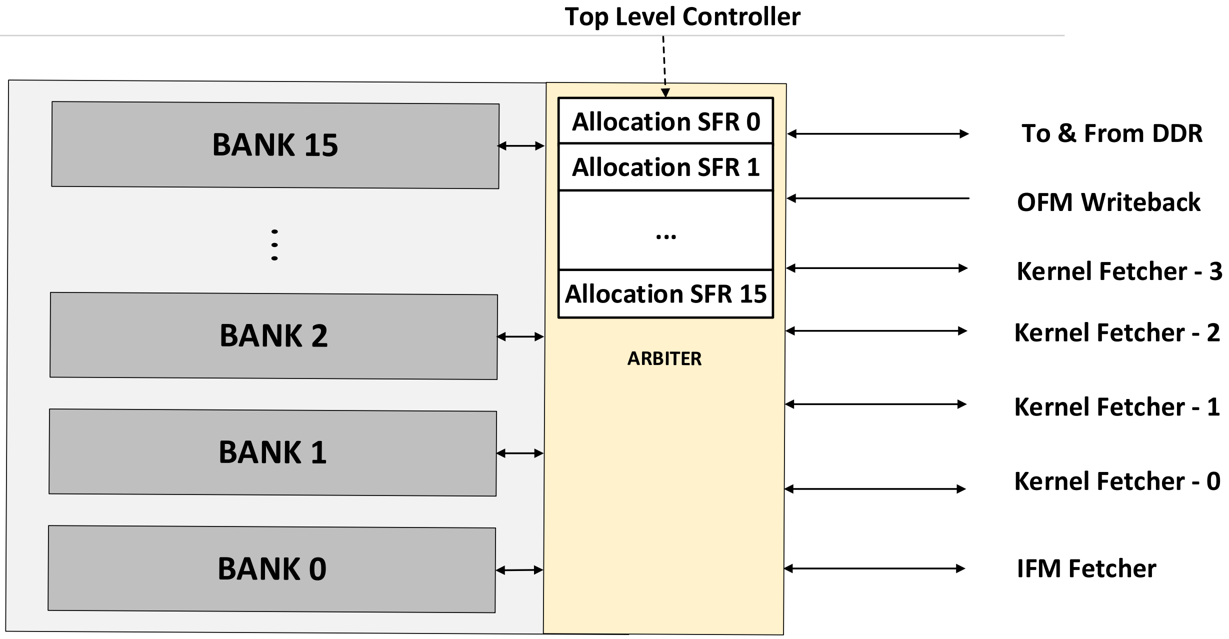
Overlapping of Computation & Communication

- Each OFM tile generation involves computation & write-back
- To improve performance we overlap next tile computation with current tile write-back
- Data hazard is prevented through software-controlled mode selection

Network Models	TOPs		
	Non-overlapped	Overlapped	% Increase
InceptionV3	3.2974	4.16	26.1
Inception-Resnet	3.8859	4.84	24.5
Resnet50	3.3467	3.77	12.6

Software-controlled Scratchpad SRAM

- Software-controlled scratchpad memory
- Configurable logical buffers
  - Minimal hardware overhead
  - Support for variable size IFM, kernel, OFM tensors
  - Logic synthesis performed using Samsung 10 nm library (at 800 MHz)



	Area (mm^2)		
Total Area	Combinational Area	Sequential Area	SRAM Bank Area
0.785442	0.01068	0.001243	0.774739
100.00%	1.35%	0.15%	98.50%

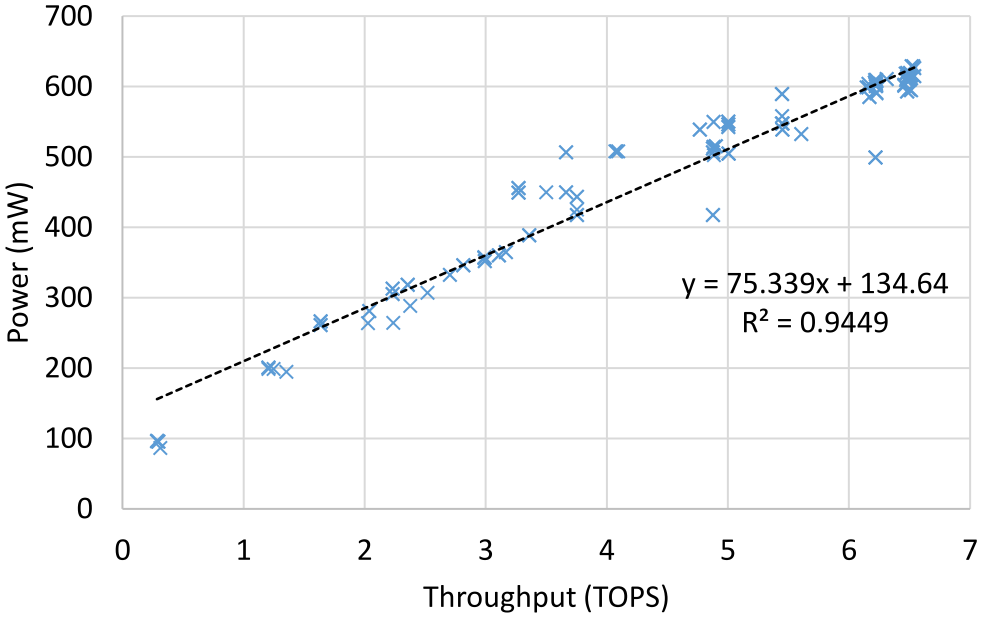
Area & Power Results

Module	Area		Peak Power	
	mm^2	Percentage	W	Percentage
SRAM	0.785442	67.72%	0.1147	19.56%
Systolic Array	0.3008	25.93%	0.416	70.97%
Support Logic	0.073505	6.33%	0.0555	9.47%
Total	1.159747		0.5864	

Table 1

Module	Module-wise Power Breakup		
	InceptionV3	Resnet50	Inception-Resnet
SRAM	19.60%	17.05%	18.34%
Systolic	71.00%	73.53%	72.78%
Support	9.40%	9.42%	8.88%

Table 2



Comparison with Other Accelerators

	Google TPU	Samsung NPU	Our Proposal
Technology	28 nm	8 nm	10 nm
Architecture	Weight-stationary Systolic Array	Wide-SIMD with zero-skipping	Output-stationary systolic array
Network	InceptionV1	InceptionV3	InceptionV3
TOPs/mm^2	0.28	1.25	3.6
TOPs/W	2.3 (peak)	3.4 (avg)	8.95 (avg)

Concluding Remarks

- In this presentation we presented a systolic dataflow based CNN accelerator
- Features like hierarchical vector units in PEs and channel major data layout are effective for improving energy efficiency
- Computation-communication overlap improves performance
- Software controlled scratch-pad memory makes SRAM more area efficient
- Effective clock gating further improves energy efficiency

Thank You!