Reusing GEMM Hardware for Efficient Execution of Depthwise Separable Convolution on ASIC-based DNN Accelerators

Susmita Dey Manasi[1,2], Suvadeep Banerjee[3], Abhijit Davare[2], Anton A. Sorokin[2], Steven M. Burns[2], Desmond A. Kirkpatrick[2], and Sachin S. Sapatnekar[1]

[1] University of Minnesota Twin Cities
Minneapolis, MN, USA

[2] Intel Labs,
Hillsboro, OR, USA

[3] Intel Labs,
Santa Clara, CA, USA

manas018@umn.edu

Outline of the Talk

- Motivation and background
- ASIC hardware platform
- Methodology for GEMM-based DwC
- Implementation flow
- Results
- Conclusion

Lightweight CNNs

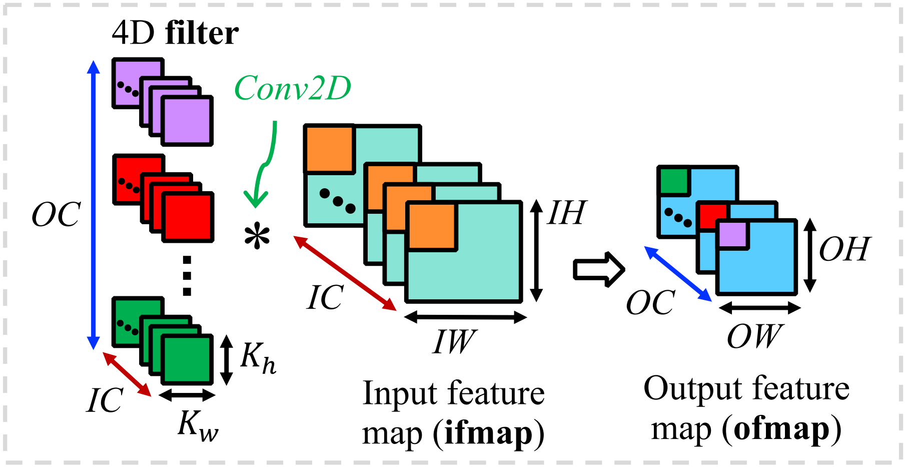
- **Lightweight** versions of CNN offer competitive accuracy
- **Wide range of applications:**
  - object detection
  - image classification
  - semantic segmentation
  - geo-localization

Very suitable for mobile and embedded platforms

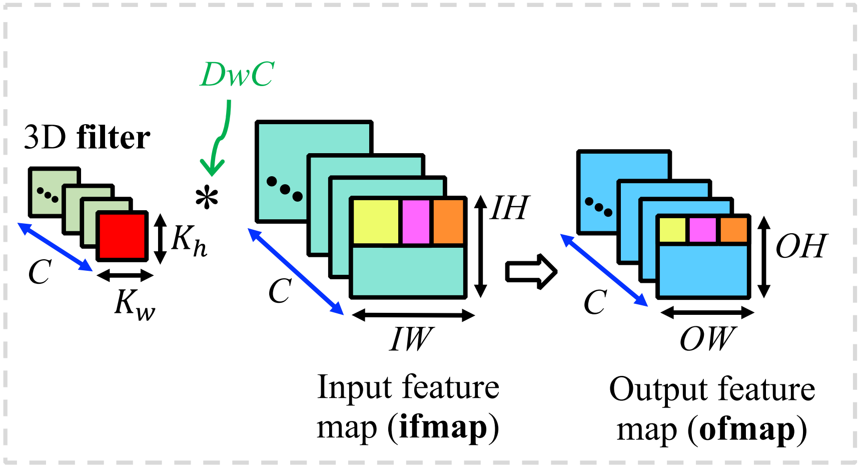
Depthwise convolution (DwC) layer is key to enable the lightweight feature

- Significantly **lower parameter counts and computational requirements** as compared to standard convolution (Conv2D)
- **Limited scope for** leveraging **data reuse and parallelism**
- **Maps poorly to general ASIC-based DNN accelerators** which are primarily optimized for standard convolution

Conv2D vs. DwC



Standard convolution

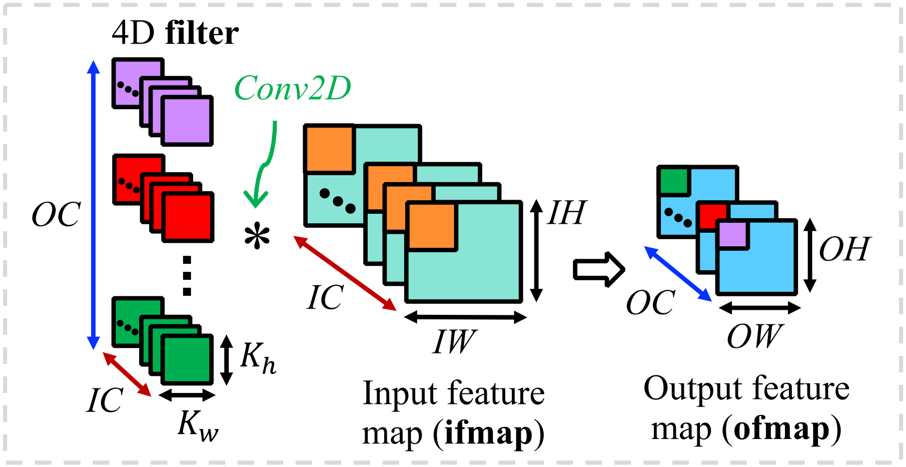


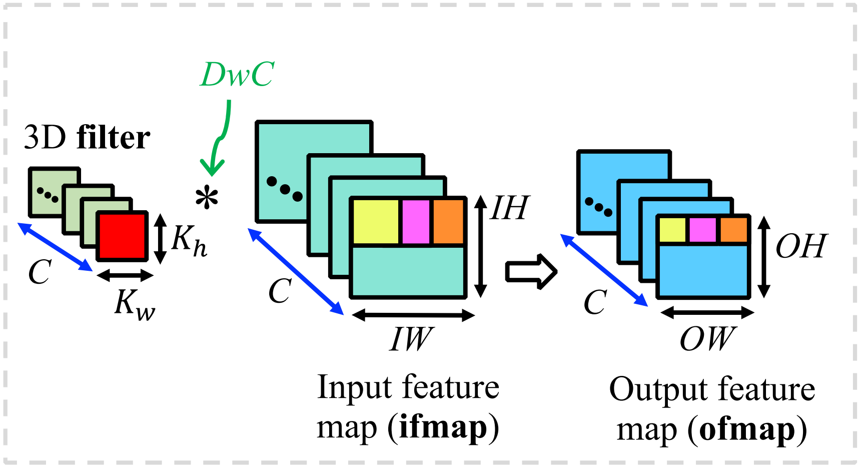
Depthwise convolution

- **Summation across all input channels**
- **4D filter**, ample data reuse opportunity

- **No summation across channel**, each channel operates individually
- **3D filter**, limited scope for data reuse

Conv2D vs. DwC





Standard convolution

Depthwise convolution

General ASIC-based DNN accelerators:

Multiplier rich 2D GEMM core

- Executes Conv2D and FC as general matrix-vector multiplication (GEMM) operation

Generic 1D ALU core

- Executes DwC, activation, pooling, etc.
- **DwC cannot be directly mapped as GEMM operation**

Value of Accelerating DwC

MobileNet-v1: Operations per layer type*

Layer type	#of MAC	Parameters
Conv2D	96.05%	74.61%
DwC	**3.06%**	**1.06%**
FC	0.18%	24.33%

Percentage of cycles for DwC and other layers wrt. total network cycles

ASIC accelerator	DwC cycles	Conv2D + Other cycles
Hardware1	**60.28%**	39.72%
Hardware2	**62.69%**	37.31%

DwC layers present significant performance bottleneck

Our proposed solution:

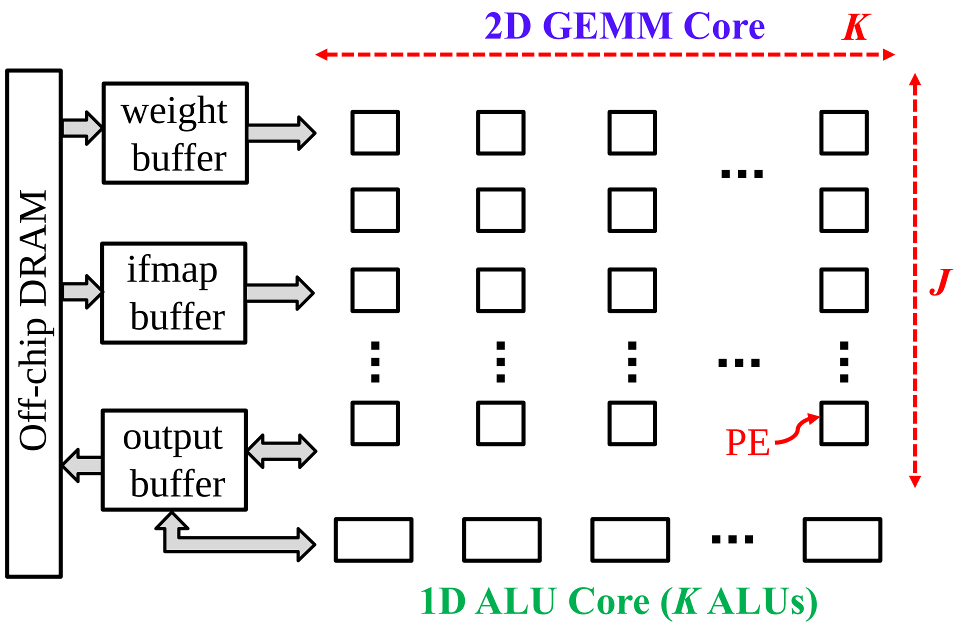
- Algorithmically map **DwC as channel-wise parallel matrix-vector multiplication**
- **Reuse the resourceful GEMM core** to execute DwC along with Conv2D
- **A simple and practical solution** to substantially accelerate DwC computation

* Howard et al., *arXiv*, 2017

Outline of the Talk

- Motivation and background
- ASIC hardware platform
- Methodology for GEMM-based DwC
- Implementation flow
- Results
- Conclusion

ASIC Hardware Platform*



A general ASIC-based platform with vector dot-product style hardware

Full-stack evaluation on TVM-VTA⁺

GEMM core:

- **In each clock cycle:**
  - MAC operations between
    - $1\times J$ **ifmap vector** and
    - $J\times K$ **filter matrix**
  - Outputs a $1\times K$ **vector of psums**
- ifmap vector: shared horizontally
- psums: reduced vertically

ALU core:

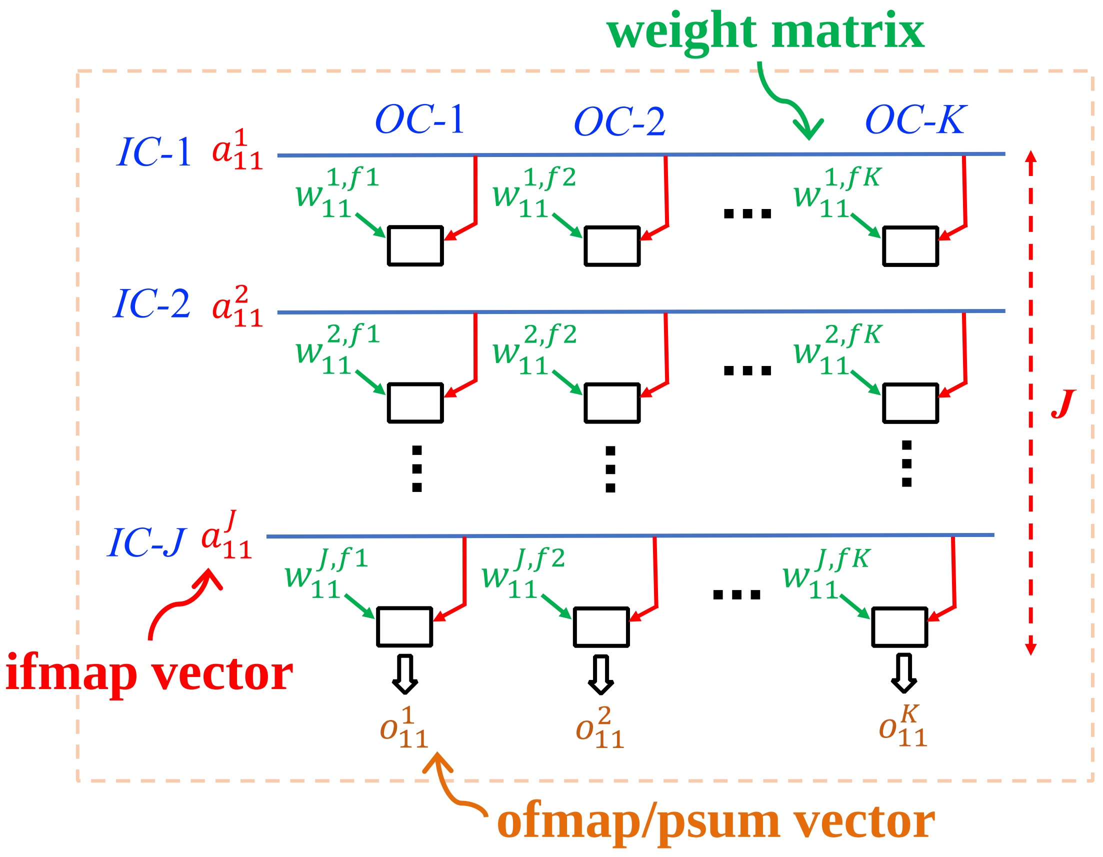
- ***K* ALUs parallelly** perform a single type of operation (i.e., mul, add, min, max, shift, etc.)
- Limited resources

* VTA[+]: Banerjee et al., *arXiv*, 2021; TVM: Chen et al., *OSDI*, 2018

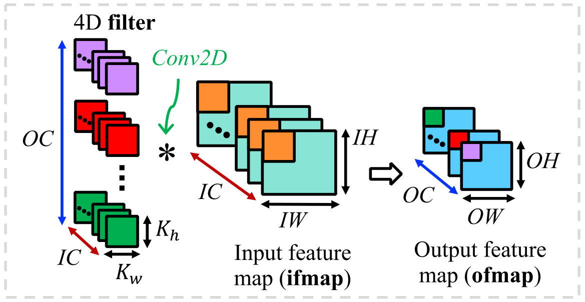
Outline of the Talk

- Motivation and background
- ASIC hardware platform
- Methodology for GEMM-based DwC
- Implementation flow
- Results
- Conclusion

Mapping Conv2D to GEMM

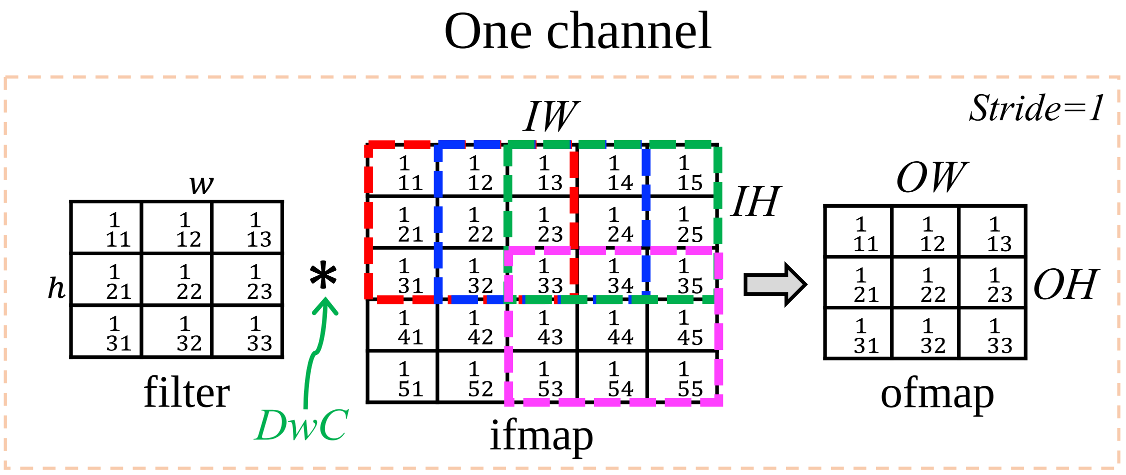


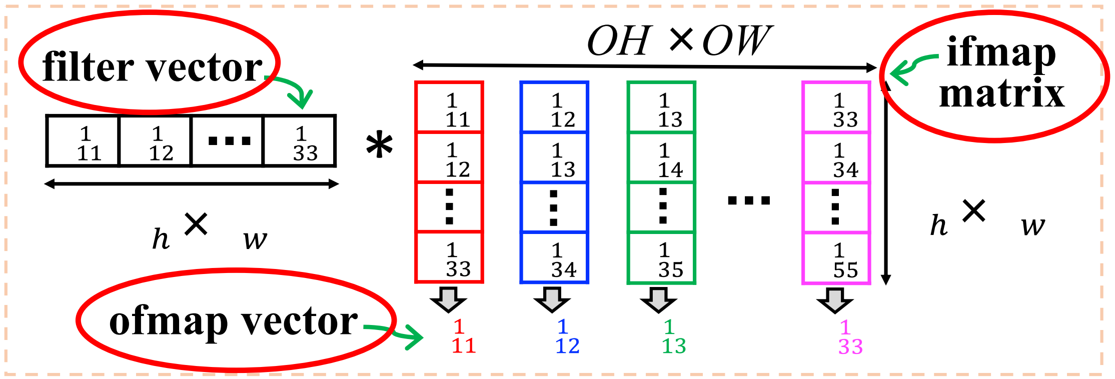
Computation in one cycle

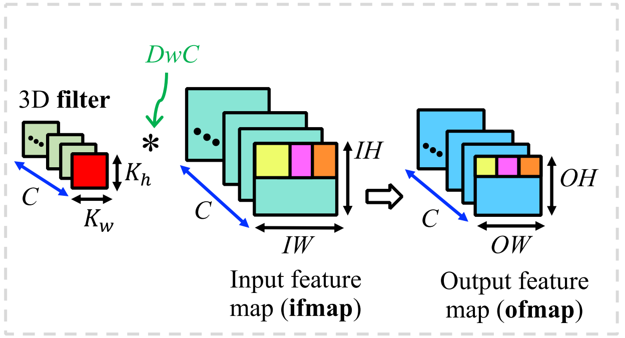


- **Vector from ifmap** using data from J ifmap channels (**shared across columns**)
- **Matrix from weight** using data from K 3D filters (**each column operates on one 3D filter**)
- Computation produces ofmap/ psum vector in K ofmap channels

DwC as Matrix-Vector Multiplication







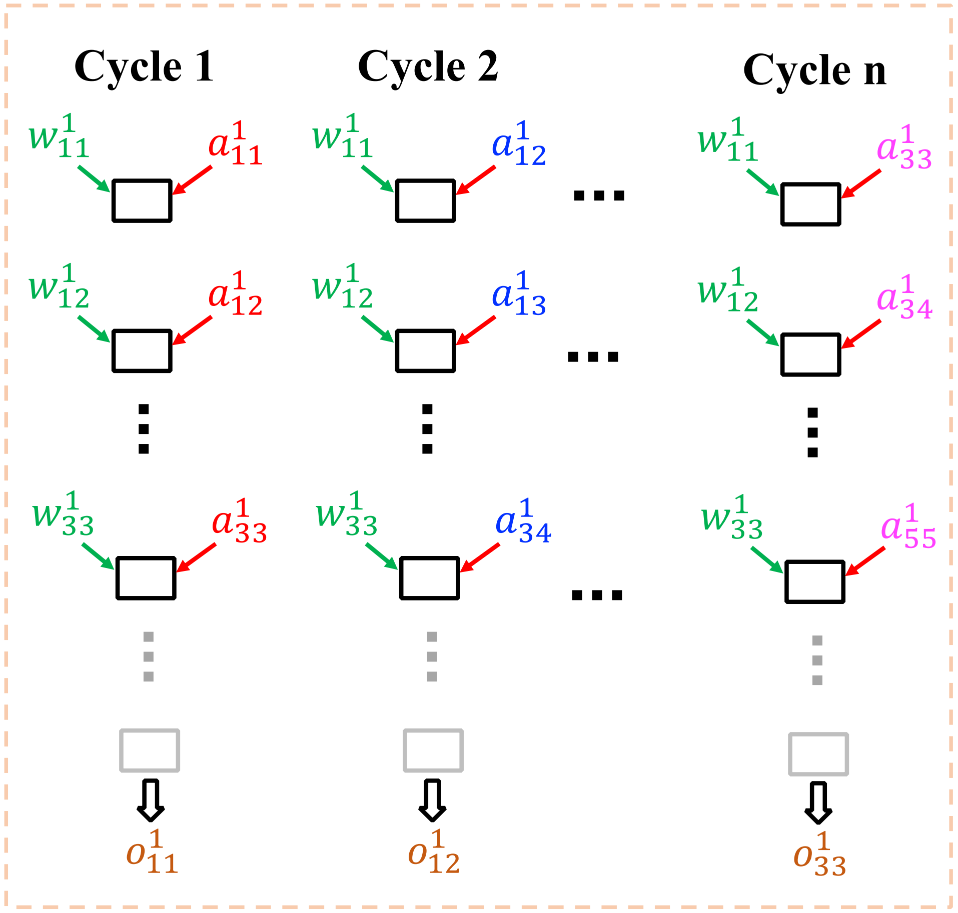
- **Vector** is formed **using filter data**
- **Matrix** is formed **using ifmap data**

Channel-wise mapping to matrix-vector multiplication

- Computation **produces vector of ofmap**
- Same mapping is applied to all channels

Mapping DwC to GEMM: One Column

Computation in one channel



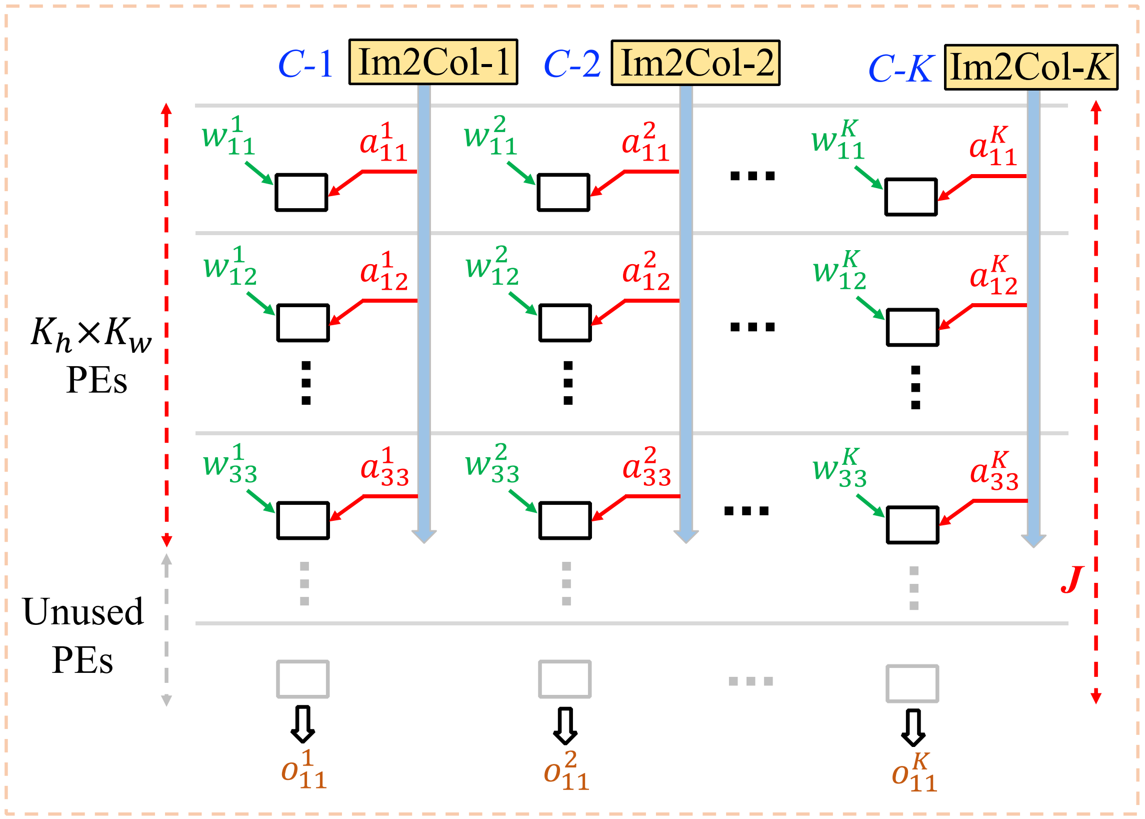
Mapping in one column over multiple cycles

- **Matrix-vector multiplication in one channel** is performed in one column of the PE array **over multiple cycles**
- **Weight vector is reused** over multiple cycles
- One column of the **ifmap matrix is supplied by an Im2Col module** every cycle

Im2Col modules: supply data from ifmap buffer in the right sequence

Mapping DwC to GEMM: 2D PE Array

Computation in one cycle



C-1
Im2Col-1
C-2
Im2Col-2
C-K
Im2Col-K
w_{11}^{1} a_{11}^{1} w_{11}^{2} a_{11}^{2} w_{11}^{K} a_{11}^{K}
w_{12}^{1} a_{12}^{1} w_{12}^{2} a_{12}^{2} w_{12}^{K} a_{12}^{K}
w_{33}^{1} a_{33}^{1} w_{33}^{2} a_{33}^{2} w_{33}^{K} a_{33}^{K}
$K_h \times K_w$ PEs
Unused PEs
J
o_{11}^{1} o_{11}^{2} o_{11}^{K}

Channel-wise parallel matrix-vector multiplication in the GEMM core

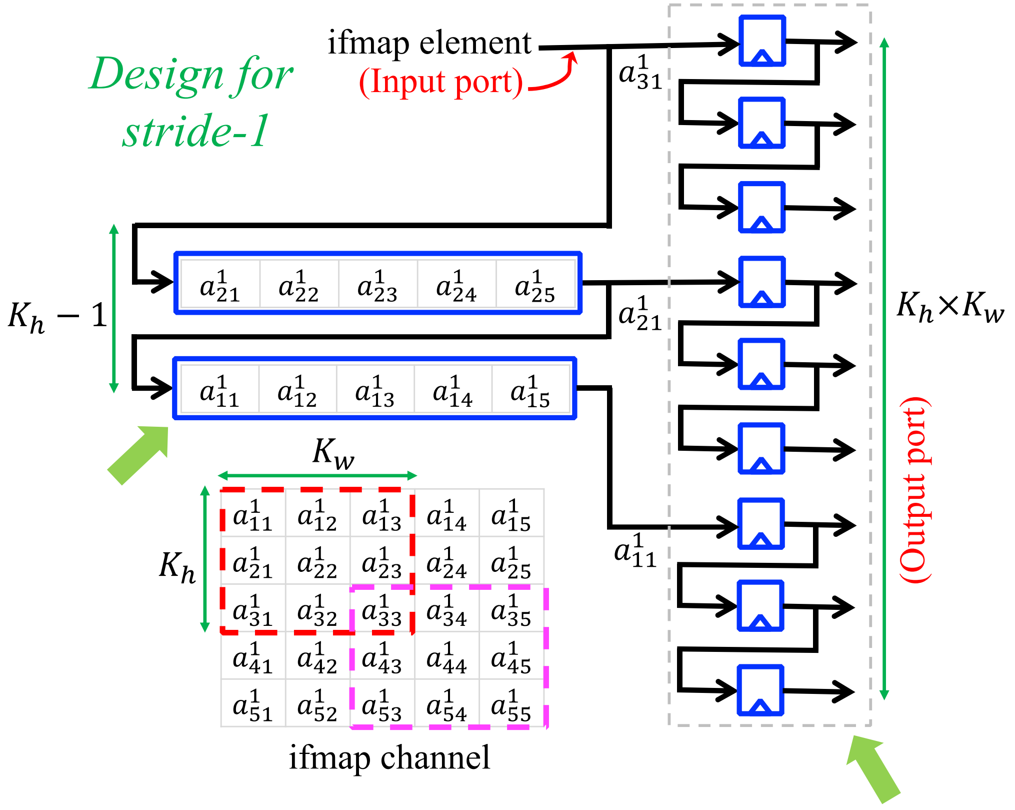
ifmap

- **Each column is equipped with an Im2Col** module
- **ifmap buffer feeds Im2Cols** instead of the PE array directly
- Multiplexer logics to **switch between Conv2D and DwC**

weight and ofmap

- **Same datapaths** for Conv2D and DwC

Im2Col Hardware



Design for stride-1
ifmap element
(Input port)
a^1_{31}
a^1_{21} a^1_{22} a^1_{23} a^1_{24} a^1_{25}
a^1_{11} a^1_{12} a^1_{13} a^1_{14} a^1_{15}
$K_h - 1$
a^1_{21}
a^1_{11}
$K_h \times K_w$
(Output port)
K_w
K_h
a^1_{11} a^1_{12} a^1_{13} a^1_{14} a^1_{15}
a^1_{21} a^1_{22} a^1_{23} a^1_{24} a^1_{25}
a^1_{31} a^1_{32} a^1_{33} a^1_{34} a^1_{35}
a^1_{41} a^1_{42} a^1_{43} a^1_{44} a^1_{45}
a^1_{51} a^1_{52} a^1_{53} a^1_{54} a^1_{55}
ifmap channel

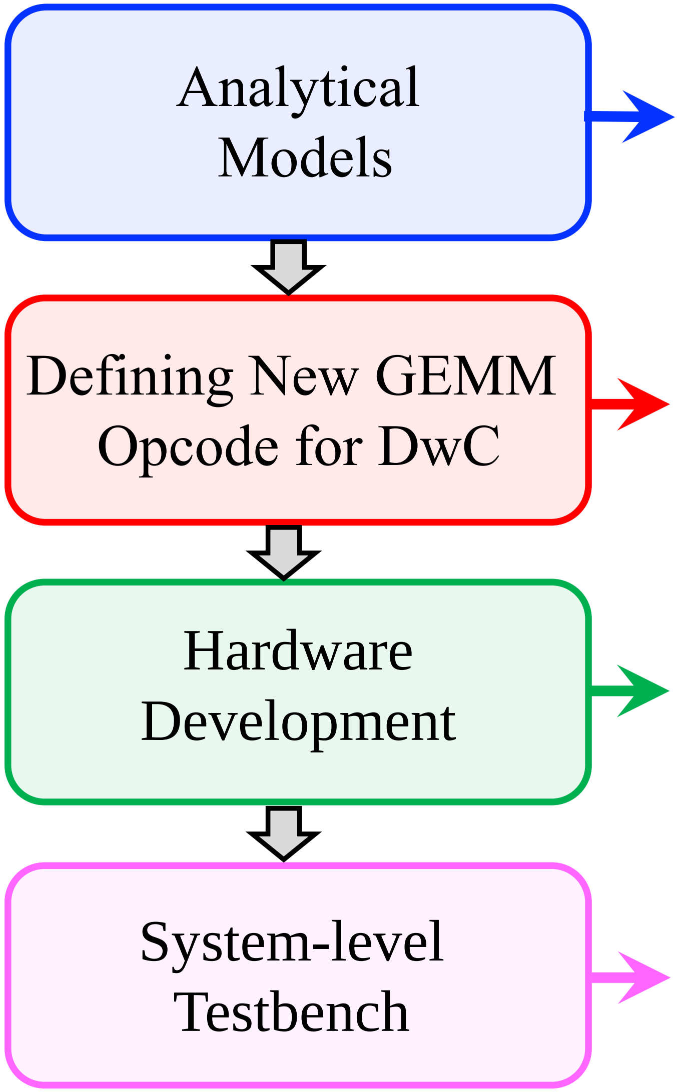
- **Line buffers:**
  - hold $(K_h - 1)$ ifmap rows at a time
  - **dual port SRAMs** to read and write data every cycle
  - FIFO like functionality
  - **Initial stall cycles** to fill the buffer lines
- **Window buffer:**
  - Produces vectorized output

- Produces **one column of the ifmap matrix every cycle**
- **A key hardware module** to enable GEMM-based DwC mapping

Outline of the Talk

- Motivation and background
- ASIC hardware platform
- Methodology for GEMM-based DwC
- Implementation flow
- Results
- Conclusion

End-to-End Implementation Flow



- **Guide** hardware design choices at the **pre-RTL phase**
- **Cost/benefit analysis for optimizations** that sound superficially promising

- Defining new opcode to **support the functionality of Im2Col-based DwC** using existing GEMM instruction

- **RTL implementations** of the required hardware modules
- **System-level integration** with VTA$^+$ stack

- **End-to-end functional verification** of the hardware

Hardware Design Trade-offs

Analytical models

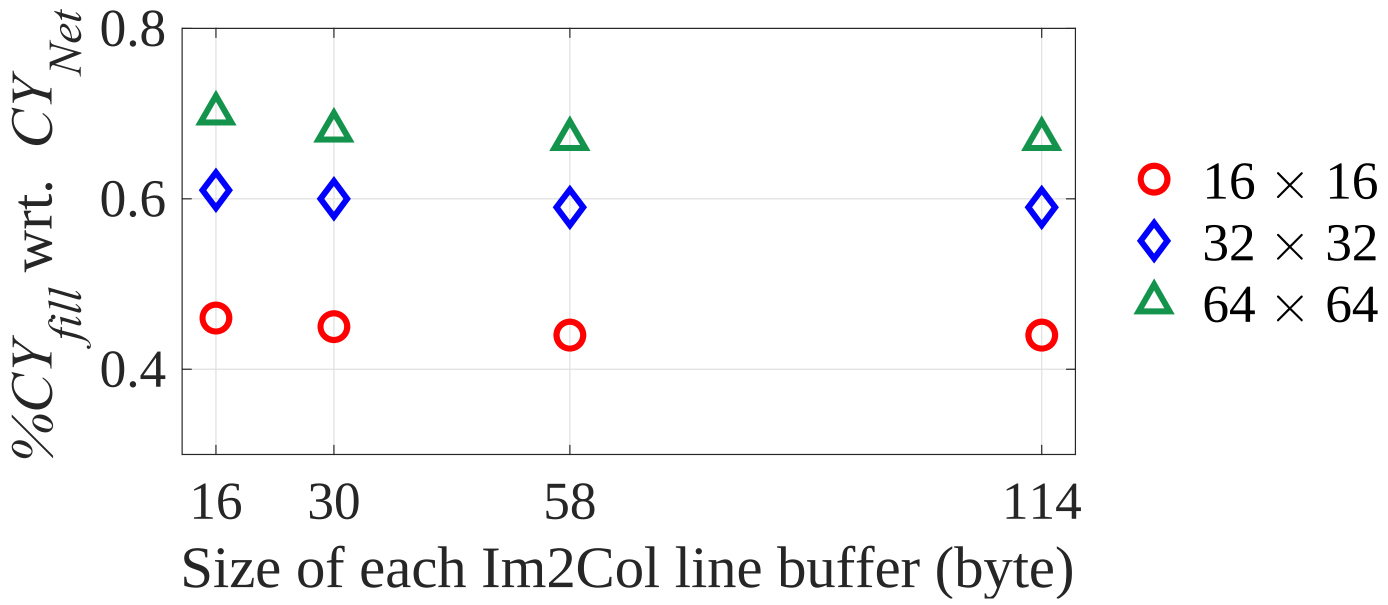
- **Conv2D and FC** in the GEMM core
- Our **GEMM-based or** traditional **ALU-based execution of DwC**
- ReLU, bias addition, shift, min, and pooling **in the ALU core**
- **Stalls due to off-chip data communication is ignored for this first order estimation.**

Key design decisions for the Im2Col-augmented GEMM core

- ***Q1: Should Im2Col induced stalls be hidden?***
- ***Q2: Should dedicated stride-2 Im2Col hardware be built?***
  - **Modified connectivity** among the SRAM lines and shift registers in the window buffer
  - **4× higher data rate** between the ifmap buffer and Im2Col module to generate one column of the ifmap matrix every cycle

Im2Col-Induced Stall Cycles

MobileNet-v1: Execution on various hardware configurations



The number of stall cycles to fill the Im2Col buffer lines (CY_{fill}) is a small fraction ($< 0.8\%$) of the total network cycles (CY_{Net})

Stride-2 Im2Col Hardware

Network performance on three different engines for MobileNet-v1

Hardware configurations*			Gain of Engine-1 w.r.t. Engine-2	Gain of Engine-1 w.r.t. Engine-3
$J \times K$	ifmap buffer	output buffer		
16×16	32 kB	64 kB	2.09%	7.24%
32×32	64 kB	128 kB	2.80%	9.54%
64×64	128 kB	256 kB	3.05%	11.10%

Dedicated stride-2 Im2Col

- **Engine-1: (extra hardware cost)**
  - Stride-1: **8 bits/cycle**
  - Stride-2: **32 bits/cycle**

Use stride-1 Im2Col for stride-2 DwC

- **Engine-2:**
  - Stride-1 and 2: **8 bits/cycle**
  - Stride-2 computation is ~4× slower
- **Engine-3:**
  - Stride-1 and 2: **4 bits/cycle**
  - Stride-1 computation is ~2× slower
  - Stride-2 computation is ~8× slower

Faster execution of stride-2 DwC at the expense of more hardware cost provides small (2-11%) performance gain

* Bitwidth of ifmap = 8; Size of each Im2Col line buffer = 58 byte

Defining New DwC-GEMM Opcode

- ➢ GEMM mapping of DwC is quite different than the GEMM mapping of Conv2D
- ➢ **A Conv2D-GEMM instruction cannot be used directly for DwC-GEMM**

VTA^{+} ISA: Affine function to express deep learning operator*

GEMM Instruction Fields

Opcode	Dept flag	Reset	μ_{begin}	μ_{end}	L_{out}	L_{in}	Unused

0th bit … *63rd bit*

$f_{o,out}$	$f_{o,in}$	$f_{i,out}$	$f_{i,in}$	$f_{w,out}$	$f_{w,in}$	Unused

64th bit … *127th bit*

Arguments for DwC-GEMM Opcode

Opcode	Dept flag	Reset	0	1	IH_{tile}	IW_{tile}	Unused

0th bit … *63rd bit*

OW_{tile}	1	IW_{tile}	1	0	0	Unused

64th bit … *127th bit*

A new DwC-GEMM opcode:

- Expresses functionality of the Im2Col-augmented DwC-GEMM operation
- **Ensures data layout compatibility** between consecutive Conv2D and DwC layers
- **Computes appropriate indices to access weight, ifmap, and ofmap data**
- **Reuses** the fields from an **existing generic GEMM instruction**

* VTA: Moreau et al., *Micro*, 2021; VTA^{+}: Banerjee et al., *arXiv*, 2021

Hardware Design and Verification

Array of Im2Col Modules

- SRAM-based implementation
- **Parallelly operate on multiple 2D channels** of a 3D ifmap

Index Generator

- Decodes the new DwC-GEMM opcode
- **Generates data indices for stride-1 and 2 DwC, and Conv2D**
- Generates appropriate valid signals

Integration with VTA$^+$ Stack

- **Integration** of Im2Col and index generator **with GEMM core**
- **System-level integration:**
  - additional control logic
  - pipeline stages

- **All implementation are done in Chisel HDL**
- Module level **parameterizable testbenches to verify functionality**

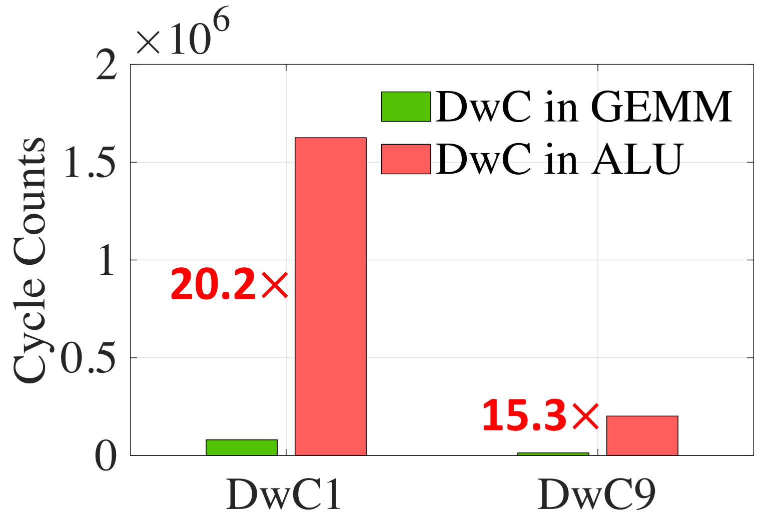
System-level Python-based testbench:

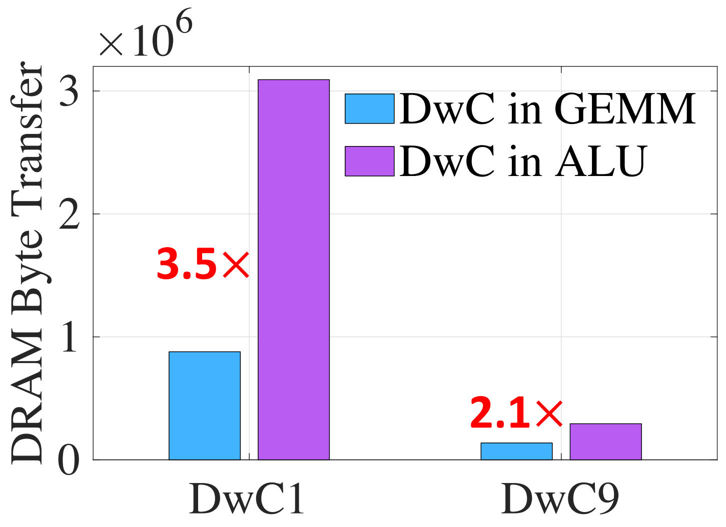
- **Tiles and schedules** GEMM-based DwC computation
- **Performs functional verification of the end-to-end hardware stack**

Outline of the Talk

Results: Performance Comparison

GEMM-executed DwC (**GED**) vs. ALU-executed DwC (**AED**)
for two DwC layers of MobileNet-v1



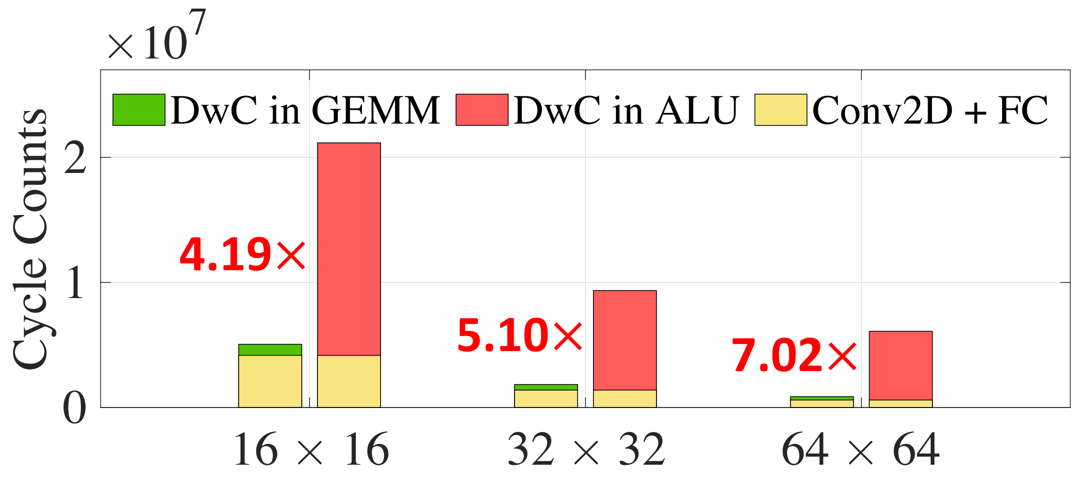


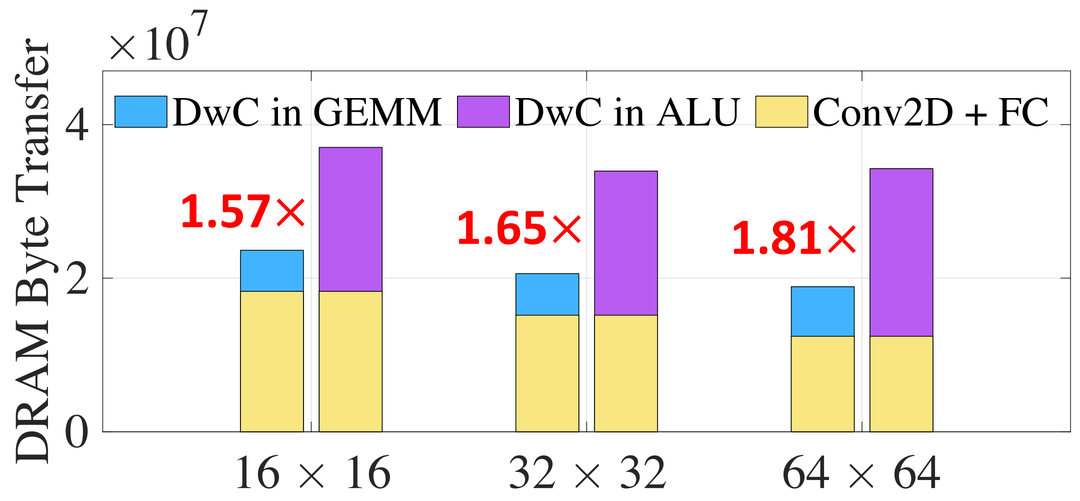
- Evaluation*[1] on VTA$^+$ accelerator platform
- Data obtained **by a cycle-accurate simulator:** extracts end-to-end performance metric from signal traces of the RTL hardware*[2]

GED offers **substantial speed-up and lower off-chip communication** for DwC layers

*[1] Hardware specifications: $J \times K = 32 \times 32$, 8-bit ifmap/weight, 32-bit psum/ofmap, 32kB, 32kB, 128kB of SRAMs, 256 bits/cycle off-chip bandwidth
*[2] https://github.com/pasqoc/incubator-tvm-vta.

Results: Performance Comparison





GED vs. AED* for MobileNet-v1 on three hardware configurations

End-to-end network runtime

Total DRAM accesses

GED offers substantial gain in overall network performance

* GED: GEMM-executed DwC; AED: ALU-executed DwC

Results: Area Comparison

Post SP&R results for GED and AED using Intel 22FFL

Evaluation on two hardware configurations

DwC in GEMM vs ALU	$J \times K$	Total SRAM size	Normalized wrt. AED	
			Total area	**Hardware overhead for GED**
AED	16 × 16	24 kB	1	**6%**
GED			1.06	
AED	32 × 32	96 kB	1	**4%**
GED			1.04	

Area cost of the supplementary hardware modules of GED is a small fraction of the accelerator area

Results: Comparison with CPU and GPU

GED vs. {CPU, GPU} performance for MobileNet-v1

Hardware platform	Runtime per inference (ms)	Speed-up of GED
GED: 64×64 PE array 396kB SRAMs, @1GHz, Off-chip bandwidth: 512 bits/cycle	0.87	1.00×
CPU: Intel(R) Xeon(R) Gold 6132 @2.60GHz, Memory: 768GB DDR4	64.26	**73.86**×
GPU: NVIDIA Tesla V100S-PCI, Memory: 32GB HBM2	1.23	**1.41**×

Significant speed-up over a modern CPU

Outperforms even a power-hungry GPU

- built for ML acceleration
- **~16× higher on chip memory** than GED

Outline of the Talk

- Motivation and background
- ASIC hardware platform
- Methodology for GEMM-based DwC
- Implementation flow
- Results
- Conclusion

Conclusion

- A new methodology is proposed to execute DwC on general ASIC-based DNN accelerator:
  - Reuses the fast GEMM core
  - Pre-RTL hardware choices are guided by **careful analytical study**
  - Incurs very small supplementary hardware cost
  - Developed **instruction-level support and system-level testbenches** to perform end-to-end evaluation on a full hardware stack

- **Substantial performance gain:**
  - **Up to 7× speed-up** and **1.8× lower off-chip communication** over a conventional DL accelerator for MobileNet-v1
  - **74× speed-up** over a CPU and **1.4× speed-up** over a GPU

THANK YOU

Questions?