Efficient Winograd or Cook-Toom Convolution Kernel Implementation on Widely Used Mobile CPUs

Partha Maji, Andrew Mundy, Ganesh Dasika, **Jesse Beu**, Matthew Mattina, Robert Mullins

Arm Research, University of Cambridge

ML and the Rise of the Edge

VR/AR/MR

Robotics

Drones

Shipping & logistics

IoT

Home, surveillance & analytics

Automotive

Mobile

Contributions of this work

- We discuss what Winograd convolution can offer in terms of performance
- Breakdown the instruction-level implications and memory layout tradeoffs for different flavors of a Winograd kernel in order to realize its full potential
- Demonstrate how general matrix multiply (GEMM) can further optimize Winograd
- Present performance results for Winograd vs conventional im2row + GEMM solution
  - More than a 2x performance boost on real hardware today!

Ultimately enable more efficient ML compute at the edge through Winograd in the Arm Compute Library (ArmCL).

Convolution and Winograd

What is Winograd and why should I care?

- Convolutional Neural Networks (CNNs)
  - Common type of deep learning model employed in a variety of domains
  - Convolve filter bank (weights) over a field (input activations) to produce a response (output)
  - Push response through an activation function (typically ReLu) and feed to the next layer
- Winograd Convolution
  - Based in the Chinese Remainder Theorem and modulo arithmetic
  - Produces mathematically equivalent results to naïve convolution*
  - Similar to using Fourier: transform into ‘Winograd domain’, do simpler math, transform result back

*Assuming infinite precision

arm

What is Winograd and why should I care?

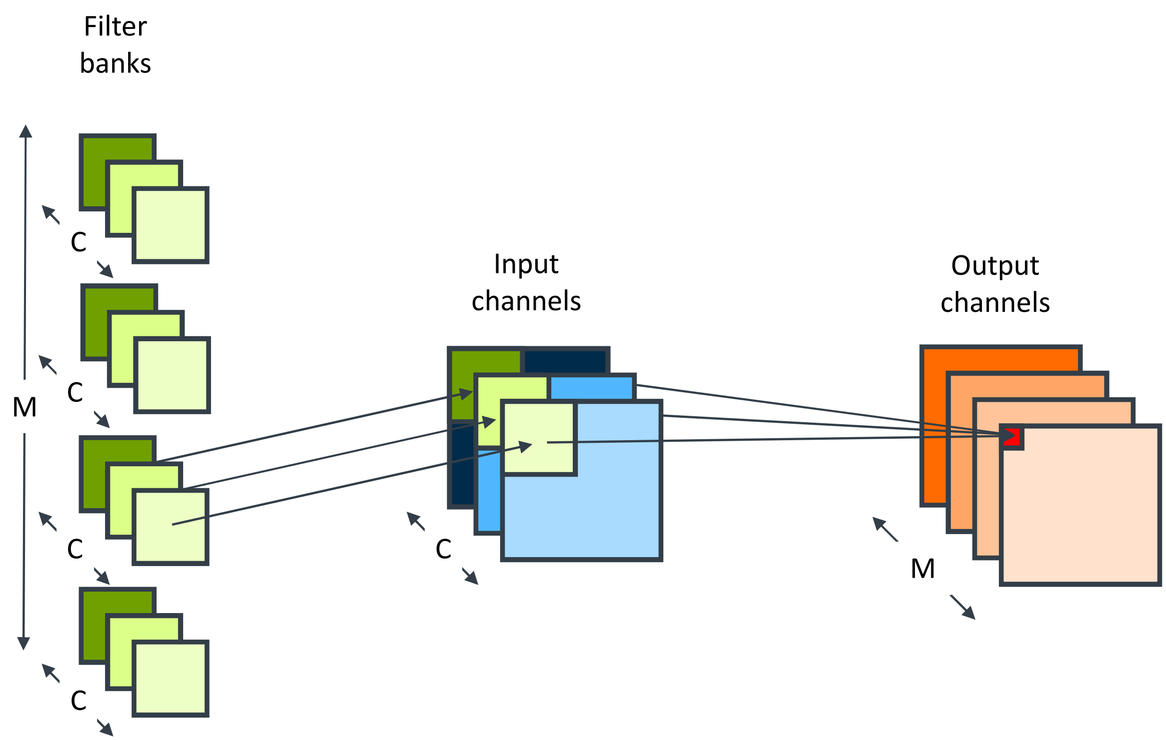
- Convolutional Neural Networks (CNNs)
  - Common type of deep learning model employed in a variety of domains
  - Convolve filter bank (weights) over a field (input activations) to produce a response (output)
  - Push response through an activation function (typically ReLu) and feed to the next layer
- Winograd Convolution
  - Based in the Chinese Remainder Theorem and modulo arithmetic
  - Produces mathematically equivalent results to naïve convolution*
  - Similar to using Fourier: transform into 'Winograd domain', do simpler math, transform result back

Objective: To (quickly) explain for a CPU context:

$$\boldsymbol{f} = \boldsymbol{Z}^{\boldsymbol{T}} \left[\left(\boldsymbol{W}\boldsymbol{w}\boldsymbol{W}^{\boldsymbol{T}}\right) \odot \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{x}\boldsymbol{X}\right) \right] \boldsymbol{Z}$$

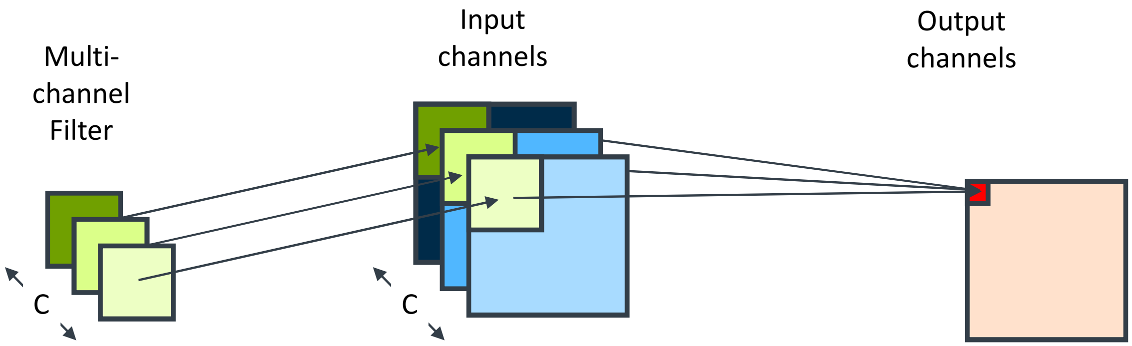
*Assuming infinite precision

Winograd Convolution

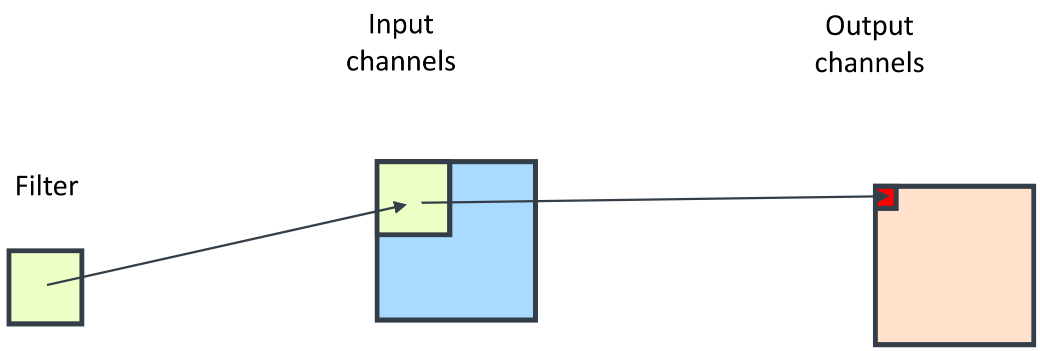


Standard CNN Configuration

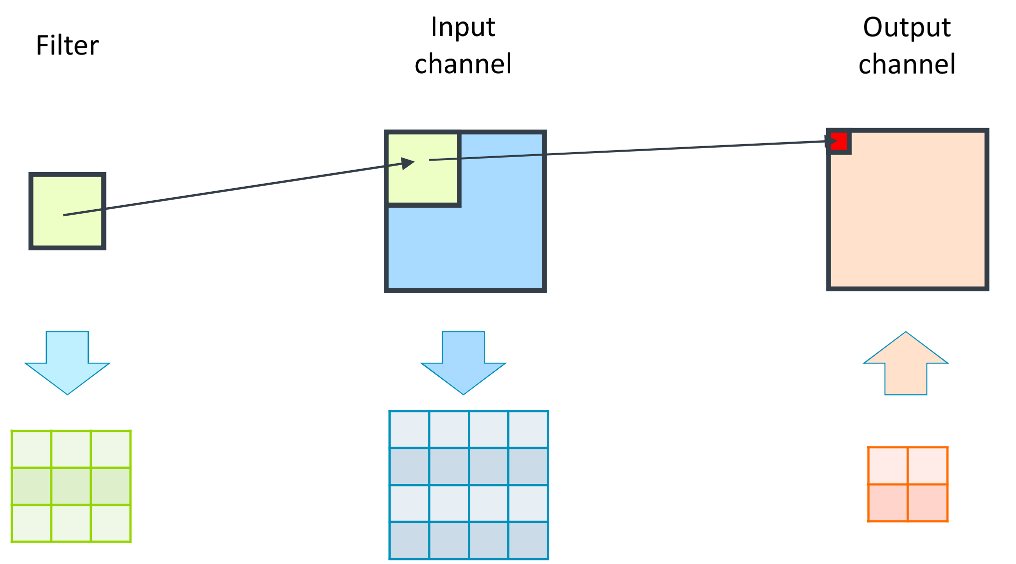
Winograd Convolution



Winograd Convolution



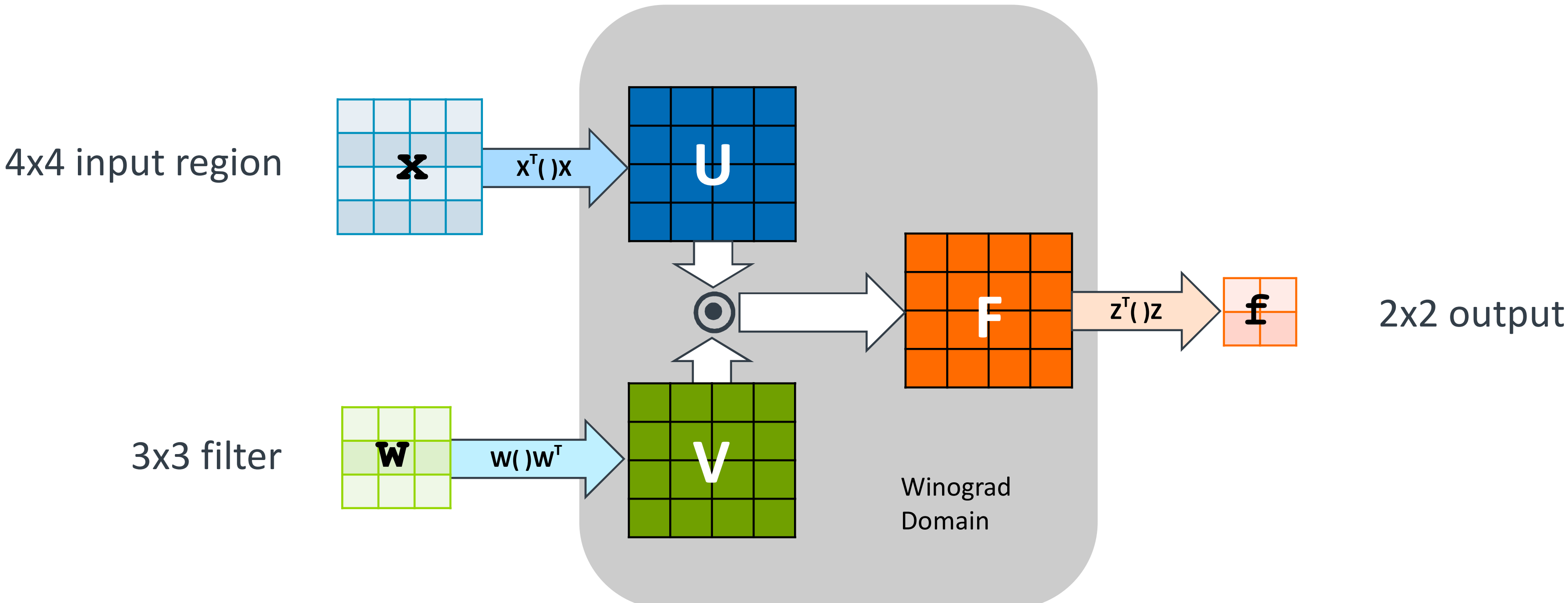
Winograd Convolution



w | x | f

Assume: 3x3 filter | 4x4 chunk of input activations (aka, a 'region') | 2x2 output

Winograd Convolution



$$f = Z^T\left[\left(WwW^T\right)\odot\left(X^TxX\right)\right]Z$$

Input Region Transform

$$(2 \times 2) = (2 \times 4)\,[\,(4 \times 3)(3 \times 3)(3 \times 4) \odot \boxed{(4 \times 4)(4 \times 4)(4 \times 4)}\,](2 \times 4)$$

$$\left[\underbrace{\begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}}_{X^T} \bullet \left[\, \underbrace{x}_{\mathbf{x}} \, \right] \right] \bullet \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}}_{X} = U$$

$$f = Z^T \left[\left(W w W^T \right) \odot \left(X^T x X \right) \right] Z$$

Filter Transform

$$(2\times 2) = (2\times 4)\,\left[\boxed{(4\times 3)(3\times 3)(3\times 4)} \odot (4\times 4)(4\times 4)(4\times 4)\right](2\times 4)$$

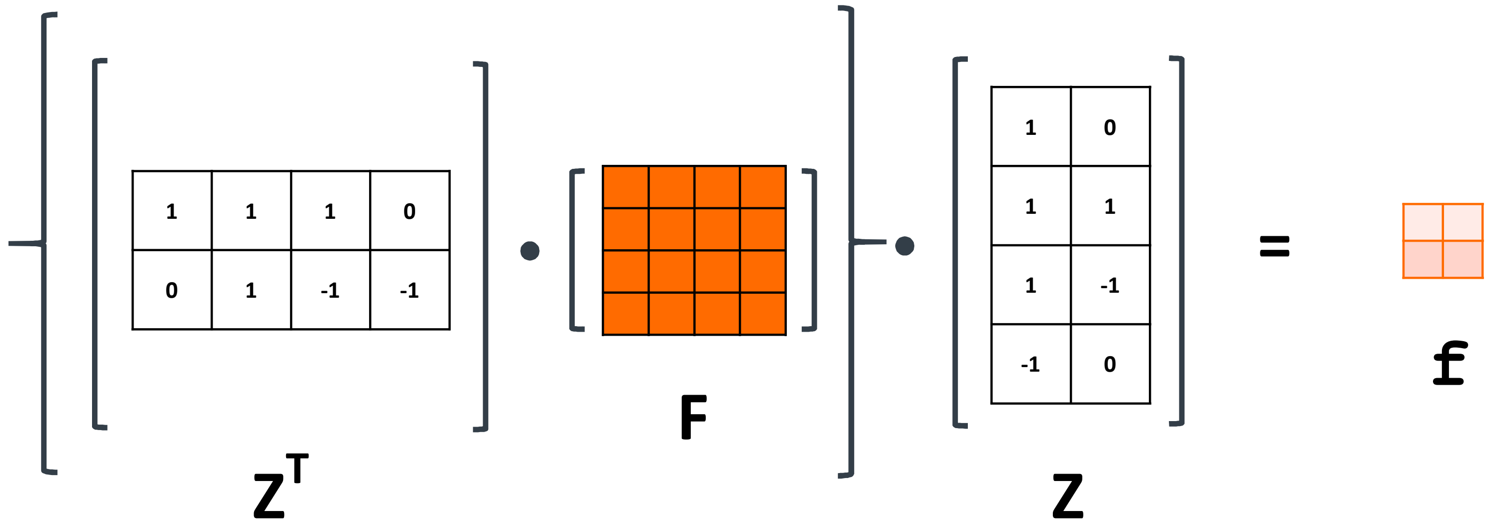
$$\left[\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1/2 & 1/2 \\ 1/2 & -1/2 & 1/2 \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{W}} \bullet \underbrace{\begin{bmatrix} \; \end{bmatrix}}_{\mathbf{w}}\right] \bullet \underbrace{\begin{bmatrix} 1 & 1/2 & 1/2 & 0 \\ 0 & 1/2 & -1/2 & 0 \\ 0 & 1/2 & 1/2 & 1 \end{bmatrix}}_{\mathbf{W}^T} = \mathbf{V}$$

$$f = Z^T\left[\left(WwW^T\right)\odot\left(X^TxX\right)\right]Z$$

Output Channel Transform

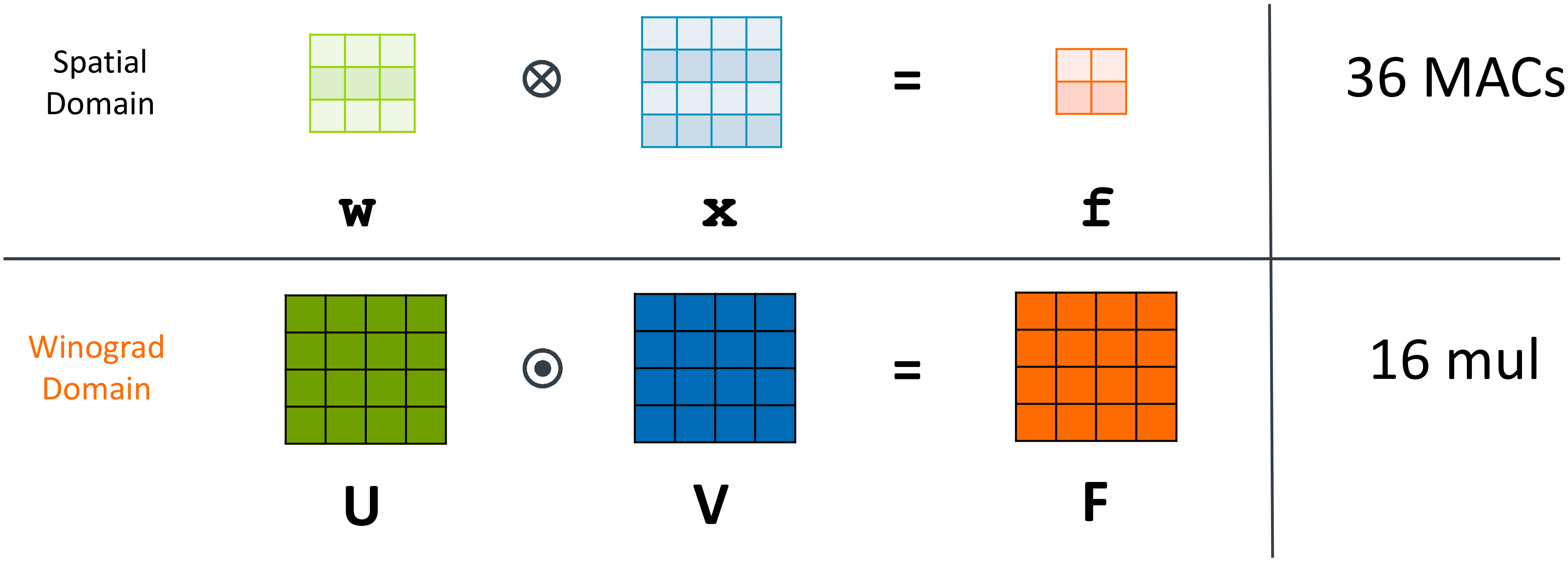
$$(2\times 2) = (2\times 4)\left[\boxed{(4\times 3)(3\times 3)(3\times 4)\odot(4\times 4)(4\times 4)(4\times 4)}\right](4\times 2)$$

(this reduces down to a 4x4)



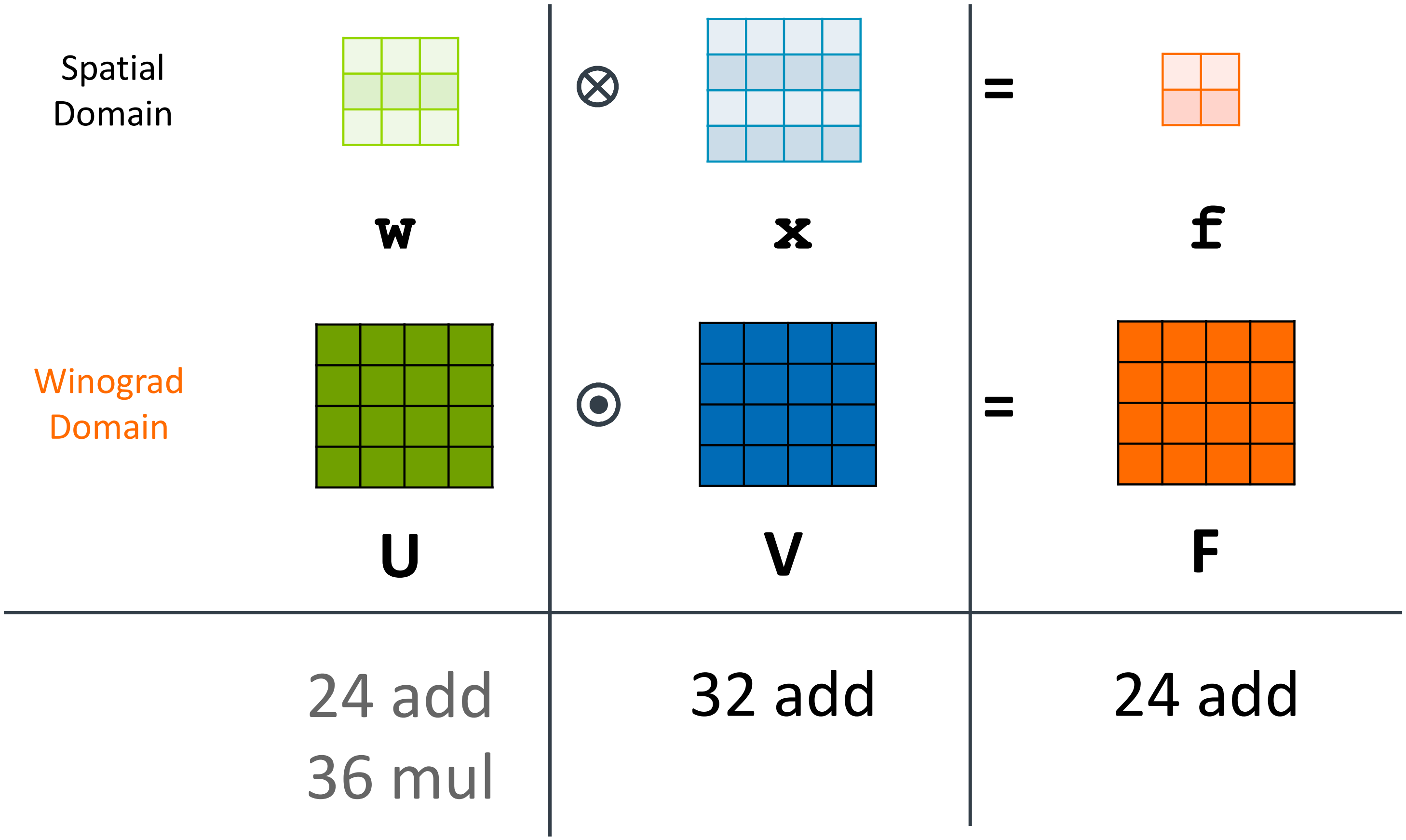
$$f = Z^T\left[\left(WwW^T\right)\odot\left(X^TxX\right)\right]Z$$

Elementwise Multiplication



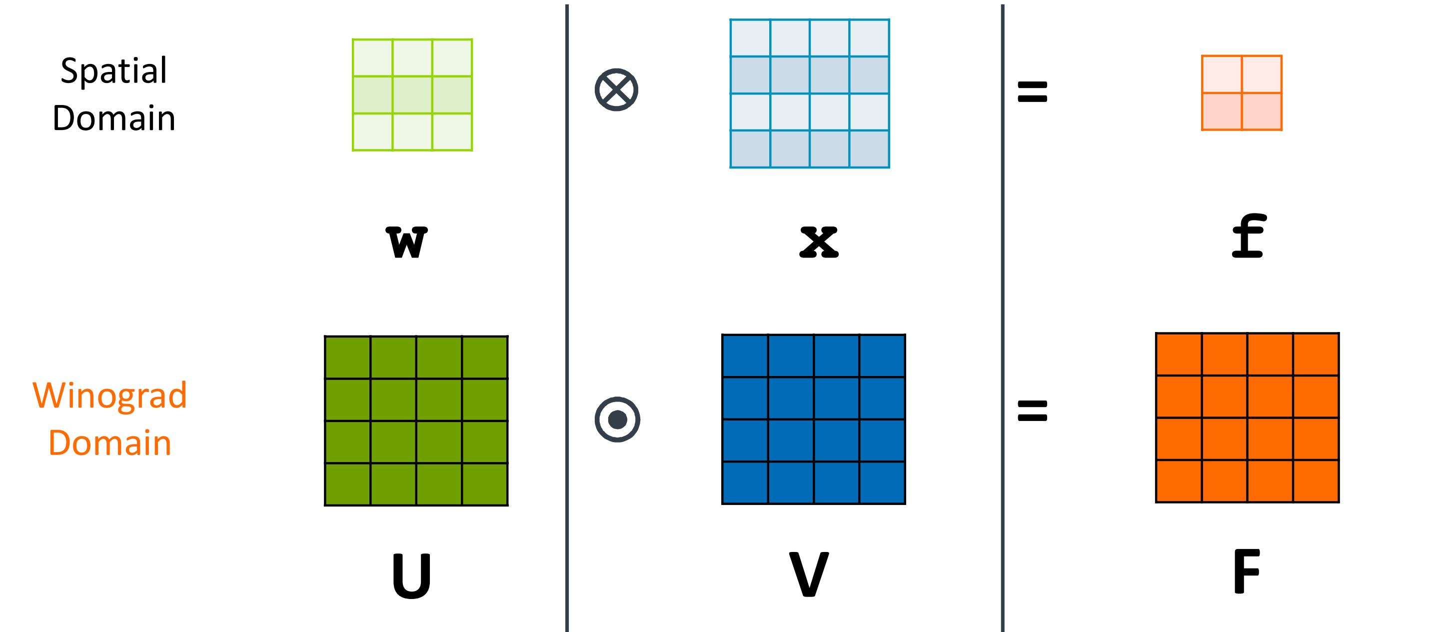
36 / 16 = 2.25x reduction in ops

Transform Cost



arm

Transform Cost



Offline Cost

24 add
36 mul

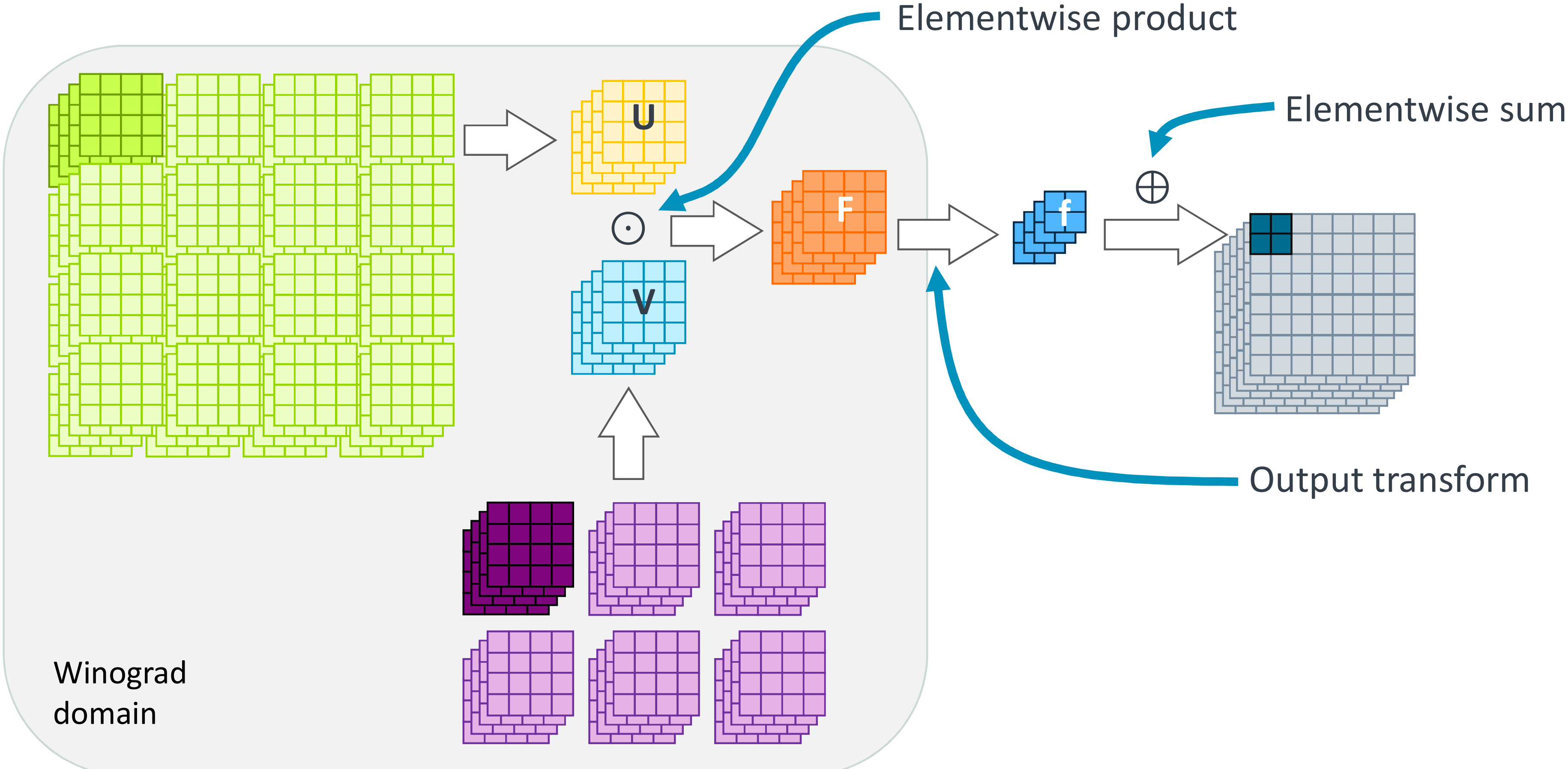
32 add

One-time cost, high reuse

24 add

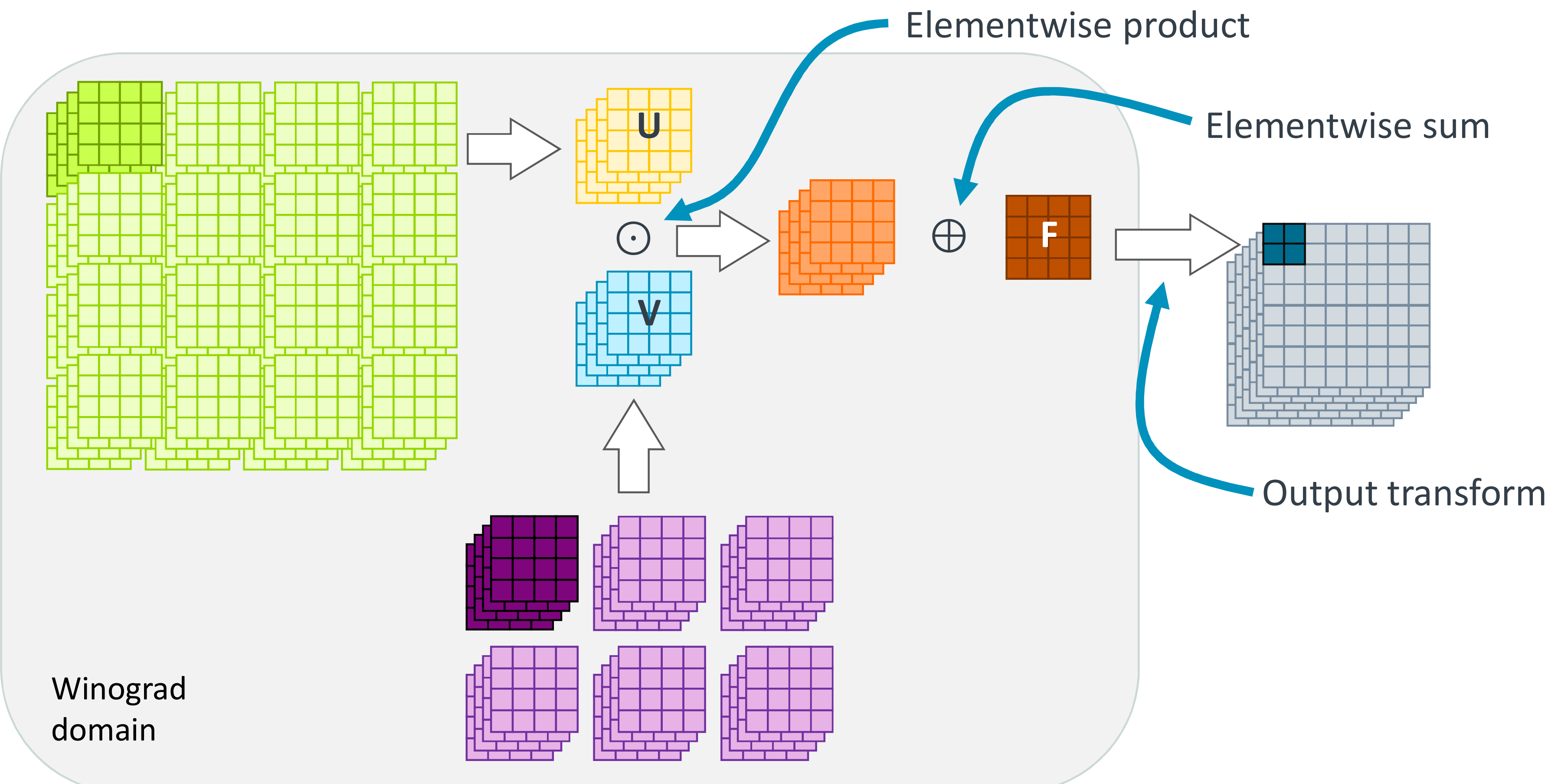
What about this??
Go Multi-channel!

Winograd for multichannel convolution



arm

Winograd for multichannel convolution – rearranged

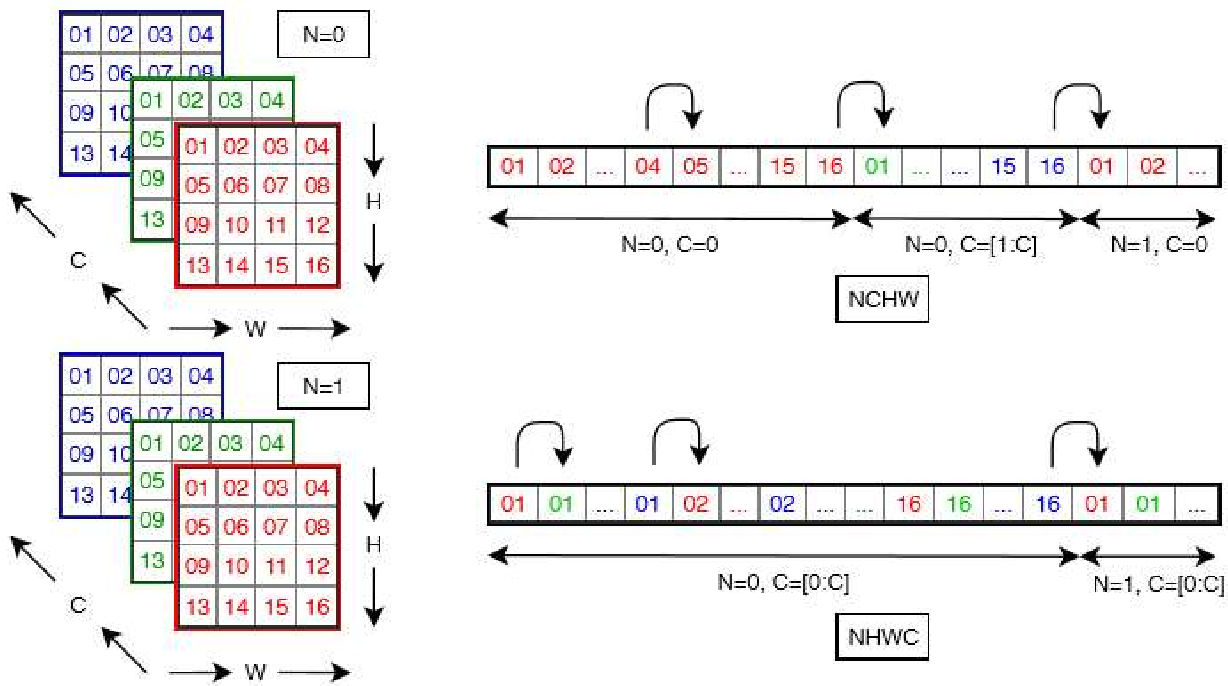


Multi-Channel Filters, Memory Layout, Vectorization, and GEMM

NCHW vs NHWC, data layout

Tensor Ordering

- N = batch
- C = channel
- H = height
- W = width

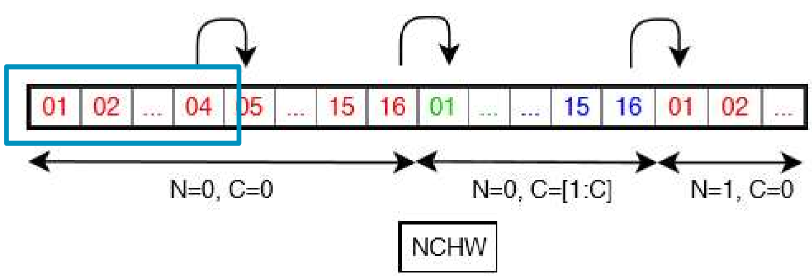


arm

NCHW vs NHWC, data layout

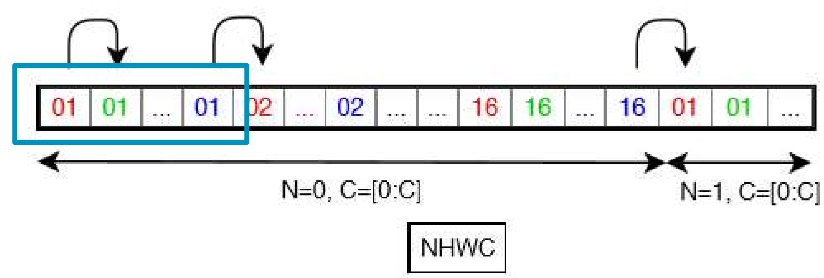
- Layout ultimately dictates how contiguous vector-load operations will populate registers
  - Under NCHW, registers will be filled entirely from a single channel
  - Under NHWC, registers will hold multiple channels for a single coordinate
- In the Arm-V8 architecture (with 128-bit SIMD registers), this means either:

An entire row of a filter per register



or

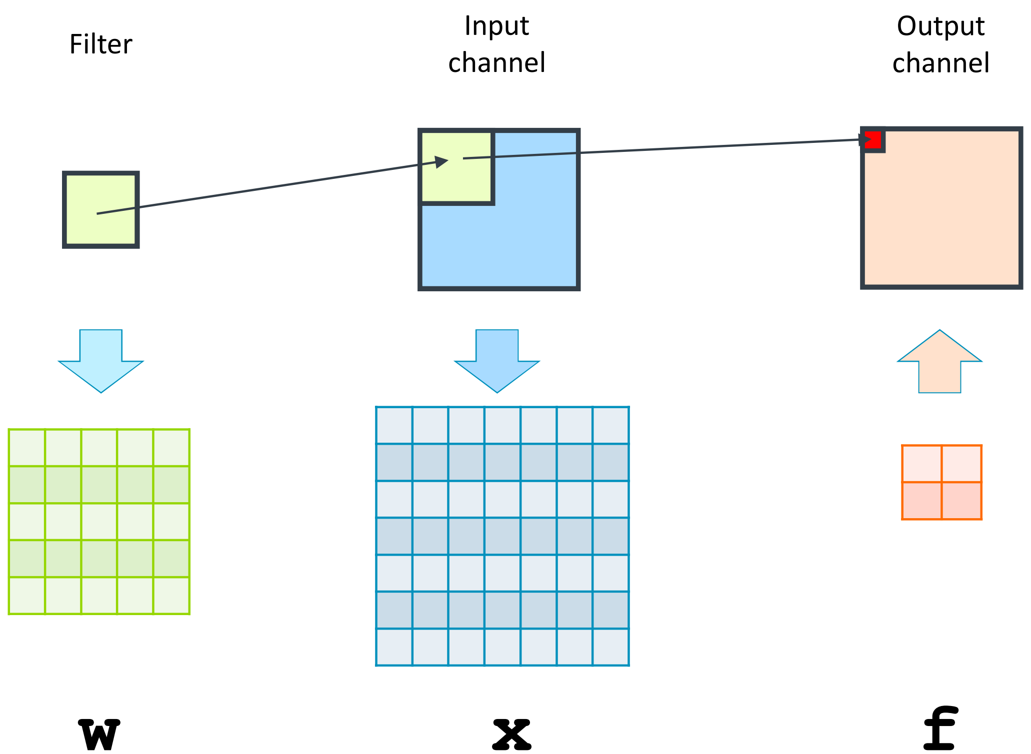
4-channels per register



Advantages to NHWC layout for CPUs

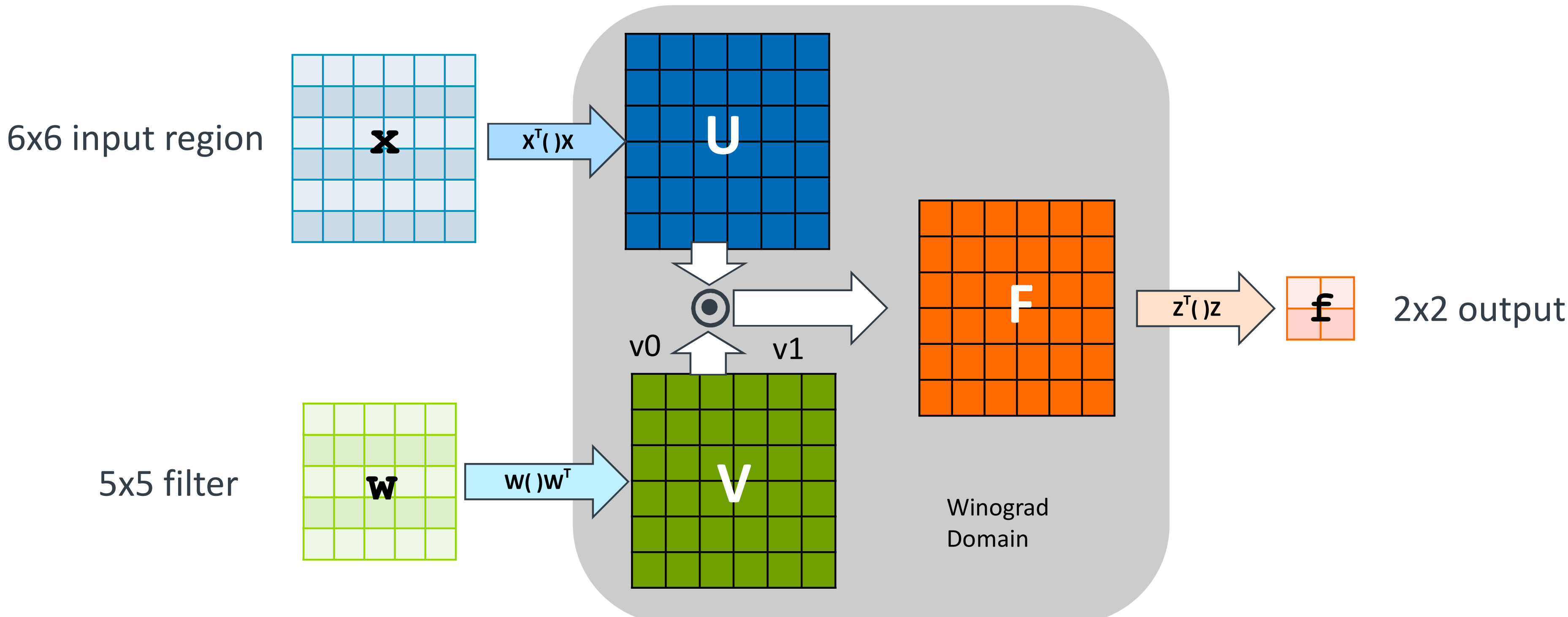
- Reasonably optimized transforms exist for both NCHW and NHWC at F(2x2, 3x3, 4x4)
- Convolution filters and Winograd are not restricted to F(2x2, 3x3, 4x4)
  - Larger regions yields can drive higher performance – e.g., F(3x3, 3x3, 5x5)
  - 5x5 and 7x7 filters found in inception networks – e.g., F(2x2, 5x5, 6x6)
  - Dimension-to-register capacity mismatch results in wasted register utilization and/or alignment complexity under NCHW
  - NHWC only experiences increased register pressure

F(2x2, 5x5, 6x6) Example

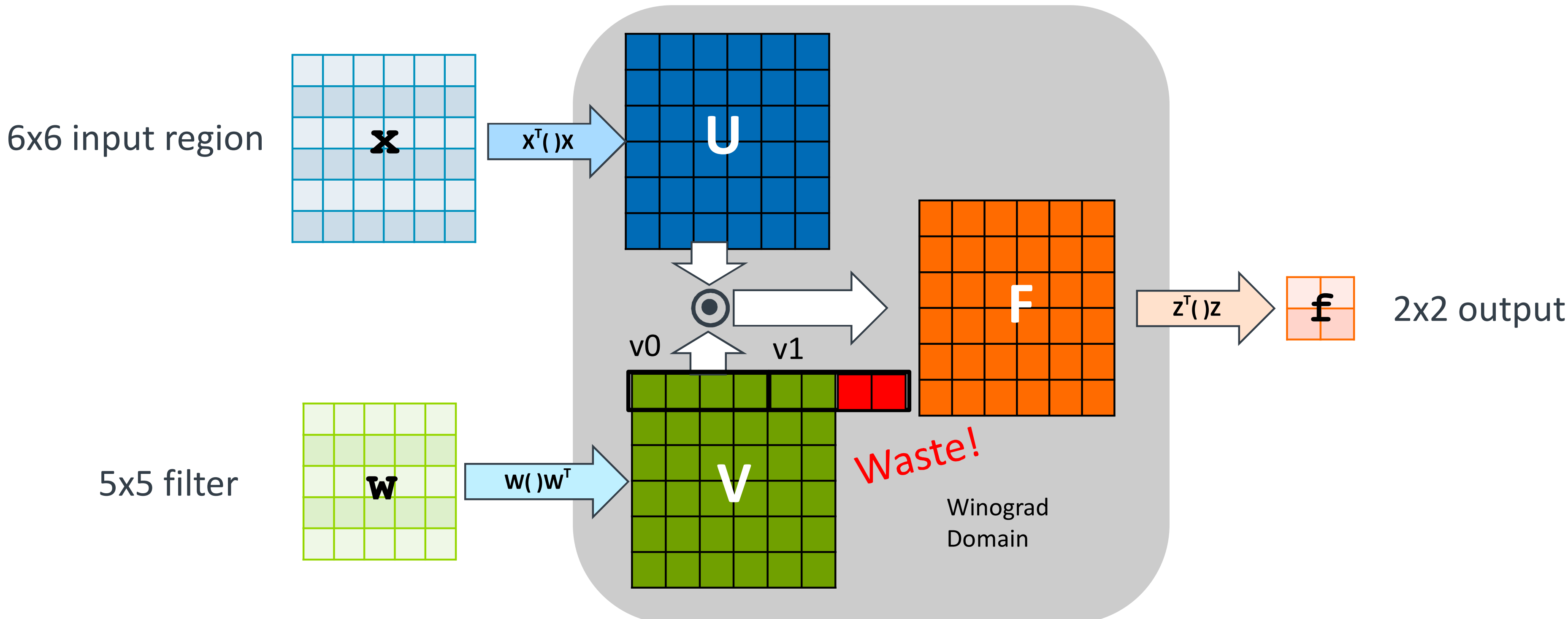


Assume: 5x5 filter | 6x6 chunk of input activations | 2x2 output

F(2x2, 5x5, 6x6) Example



F(2x2, 5x5, 6x6) Example



Advantages to NHWC layout for CPUs

- Reasonably optimized transforms exist for both NCHW and NHWC at F(2x2, 3x3, 4x4)
- Convolution filters and Winograd are not restricted to F(2x2, 3x3, 4x4)
  - Larger regions yields can drive higher performance – e.g., F(4x4, 3x3, 6x6)
  - 5x5 and 7x7 filters found in inception networks – e.g., F(2x2, 5x5, 7x7)
  - Dimension-to-register capacity mismatch results in wasted register utilization and alignment complexity under NCHW
  - NHWC only experiences increased register pressure
- Wider registers or lower precision also adds challenges for NCHW
  - 256-bit or FP16 means 8 values per register, or 2 rows per register under NCHW
  - Loss of 1:1 register-row mapping complicates assembly sequence for efficient NCHW transpose
  - NHWC simply doubles the # of channels stored per register

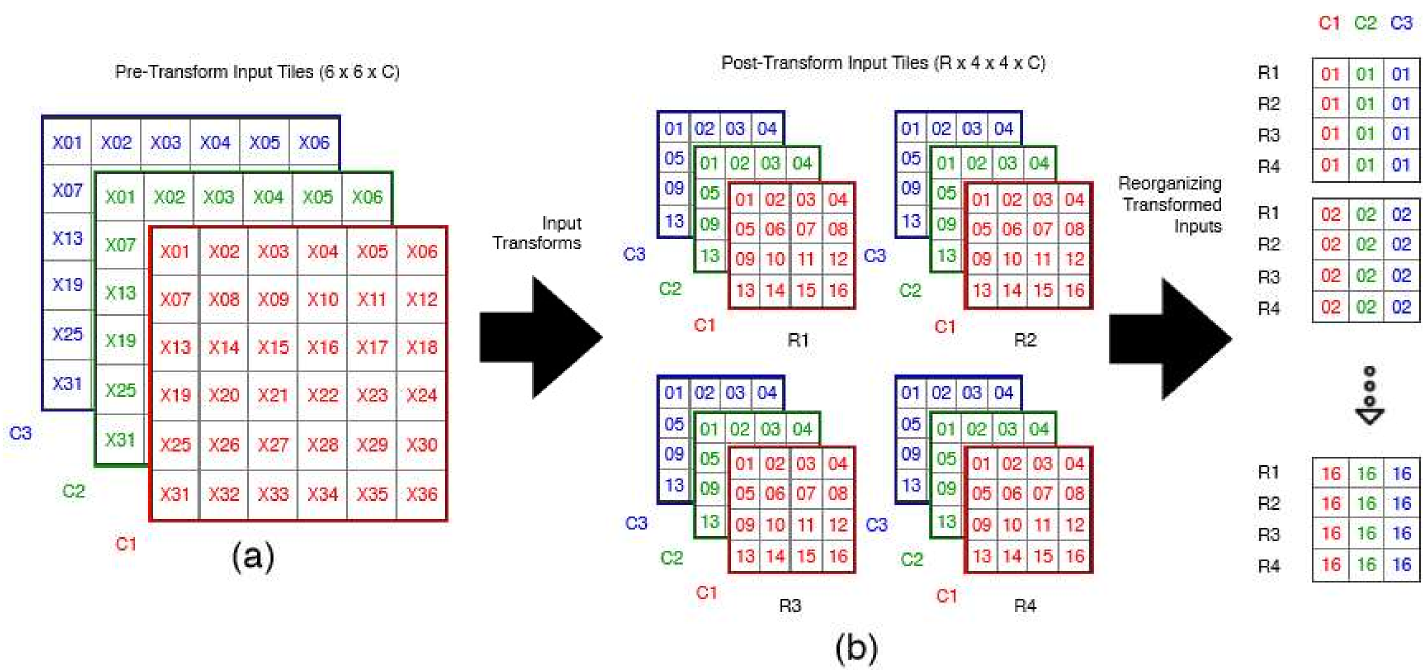
Vectorization over channels is more portable and performant!

Use of GEMM to further optimize

- General Matrix-Matrix Multiply is a common, highly optimized operation for most architectures, including Arm
- Inspection of the full Winograd convolution algorithm (Listing 1 in paper) shows:
  - The fundamental operation is a multiply-accumulate
  - There are 2 axis of data re-use:
    - weight tile reuse over all input regions and
    - input region reuse over all output channels
  - Opportunity to leverage GEMM to do the computation in a highly parallel manner

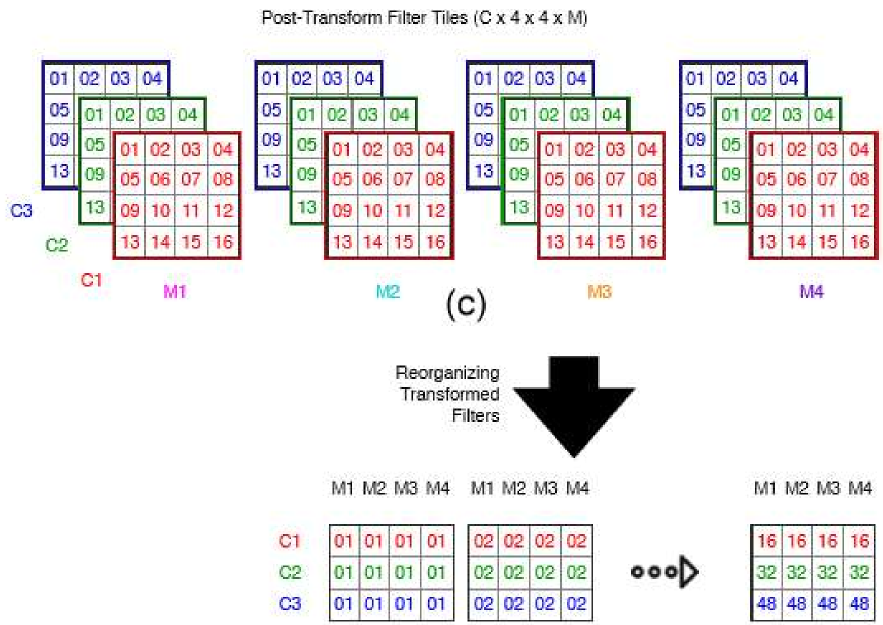
arm

Winograd execution using Matrix of GEMMs

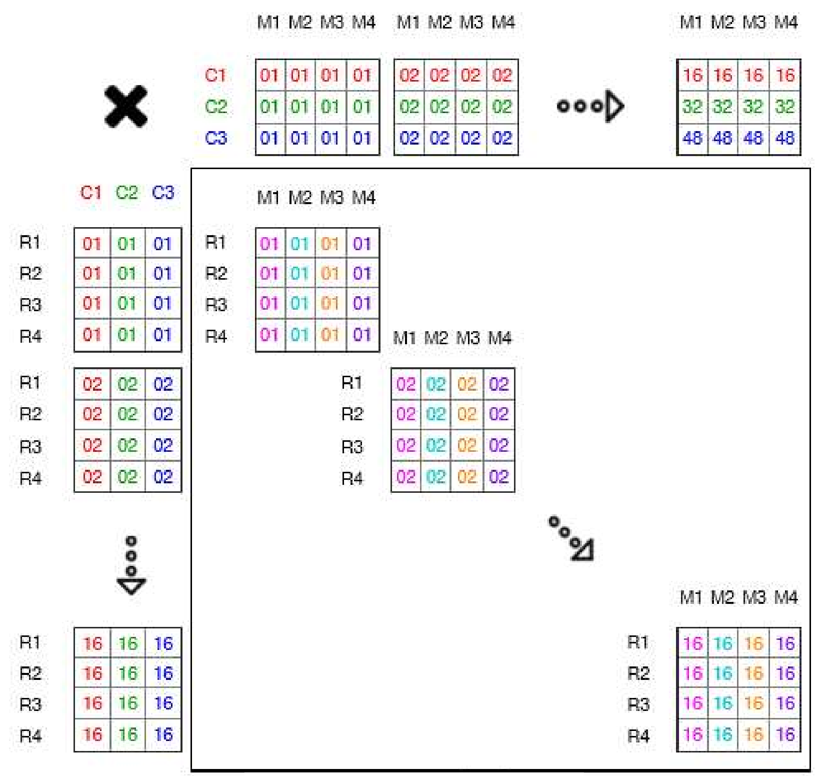


arm

Winograd execution using Matrix of GEMMs

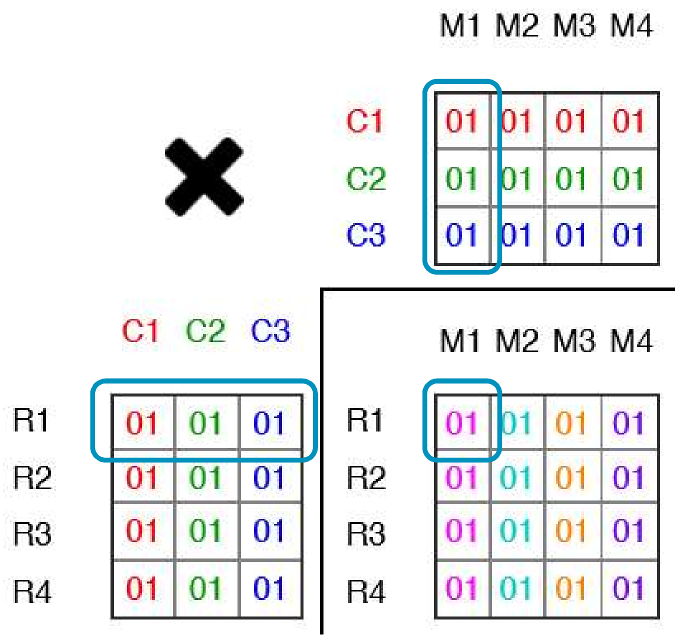


Winograd execution using Matrix of GEMMs



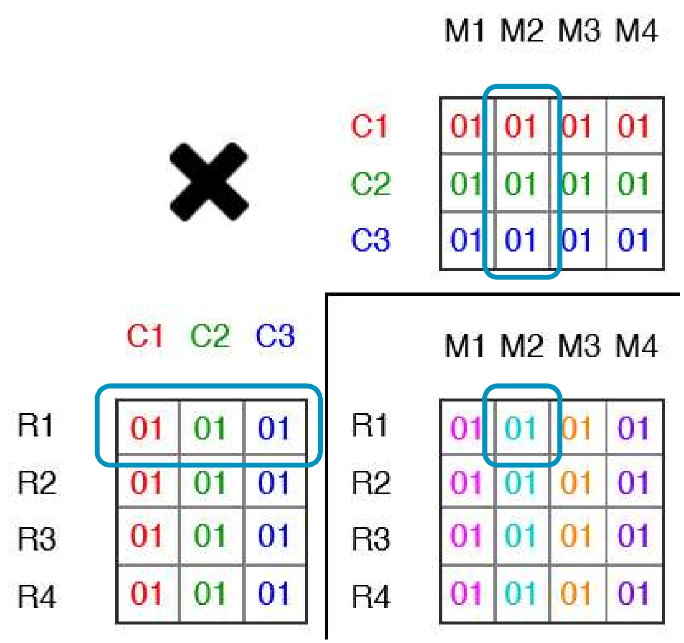
(d)

Zoom on individual GEMM

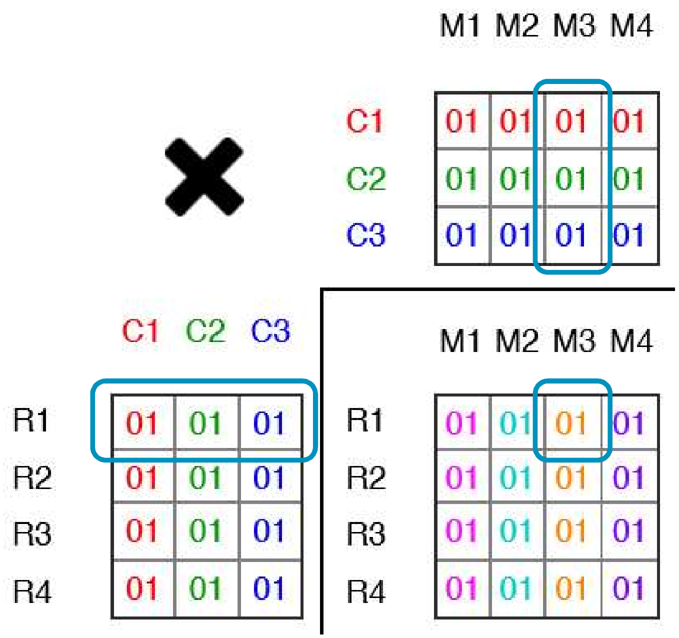


arm

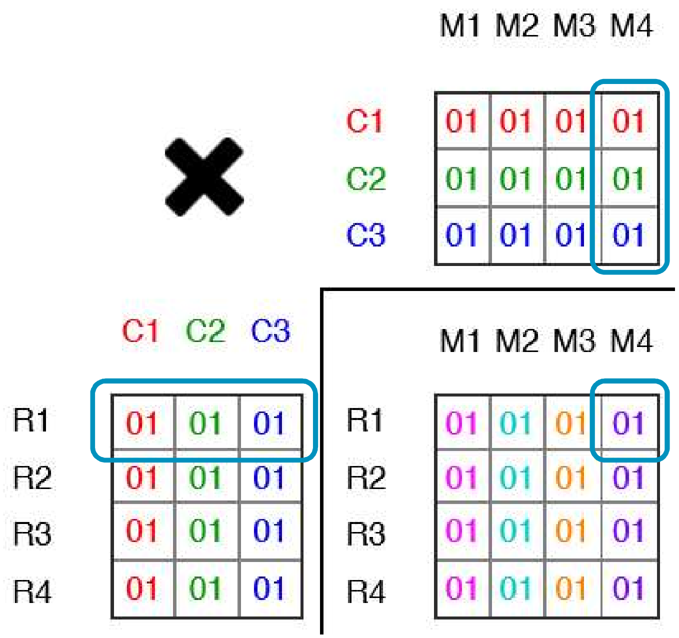
Zoom on individual GEMM



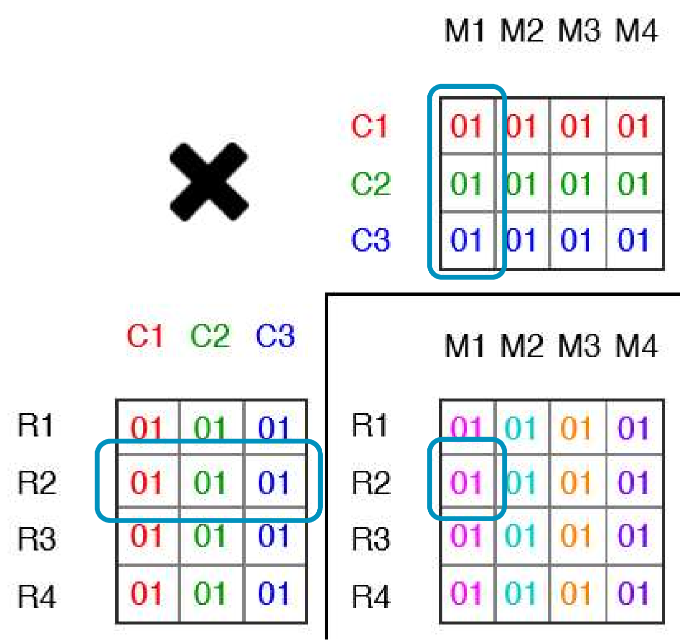
Zoom on individual GEMM



Zoom on individual GEMM

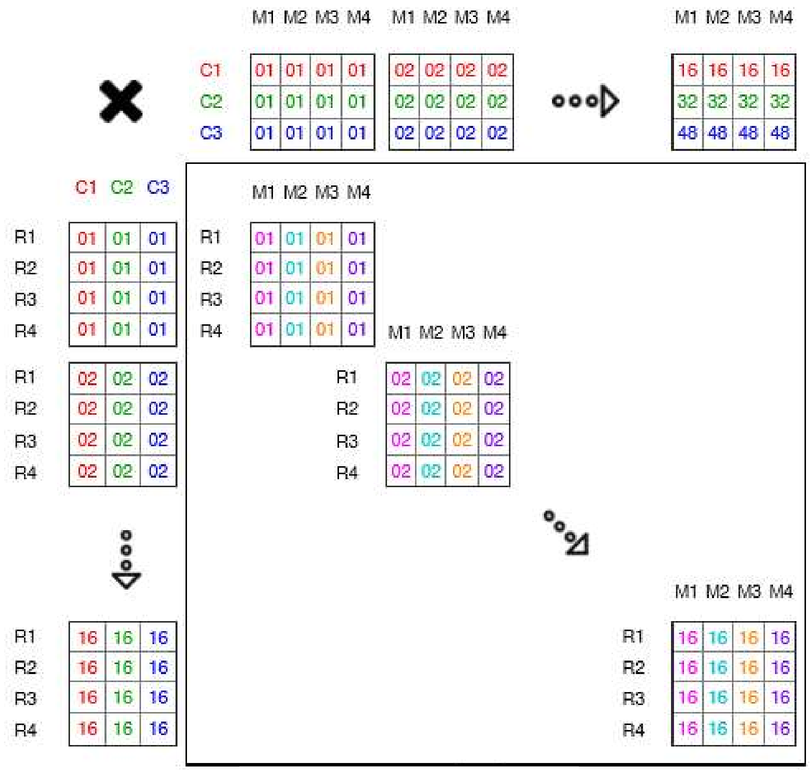


Zoom on individual GEMM



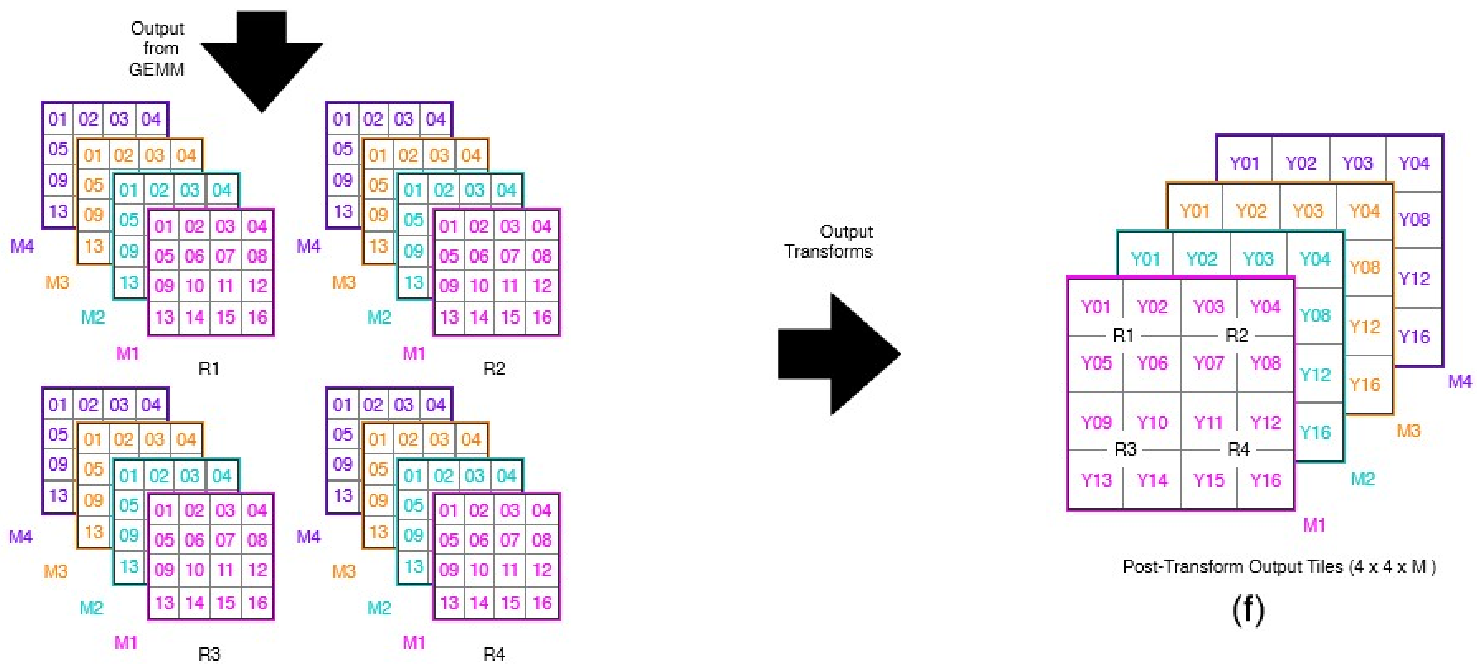
arm

Winograd execution using Matrix of GEMMs



arm

Winograd execution using Matrix of GEMMs



Pre-Transform Output Tiles (R x 4 x 4 x M)

(e)

(f)

Results

Experimental Setup

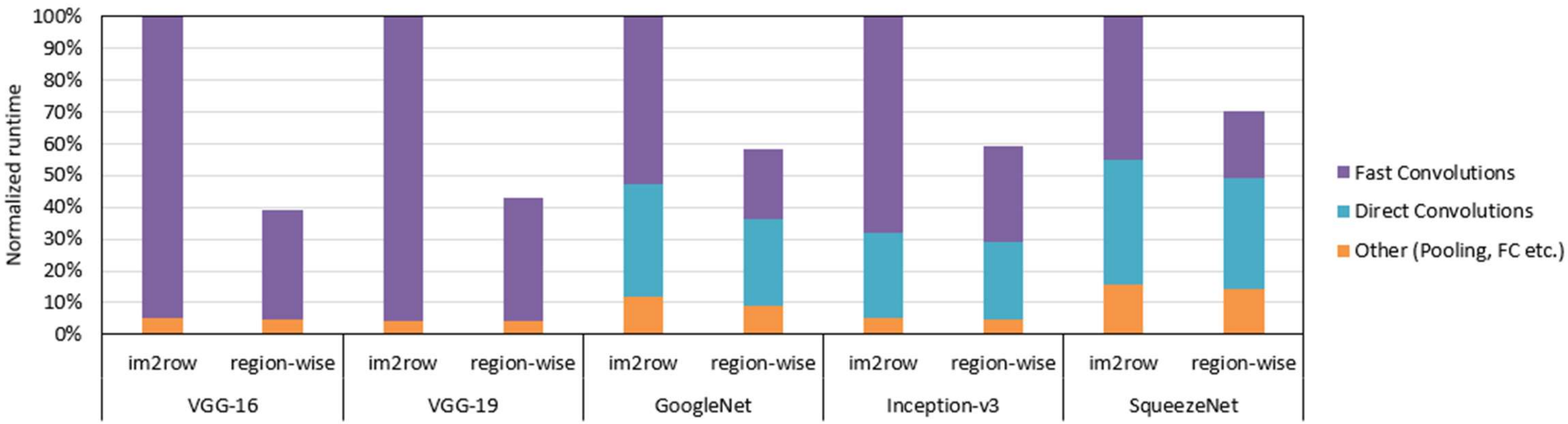
Platform: Huawei HiKey960 Development Platform – 4xA73 cluster

Networks: VGG19, VGG16, GoogleNet, Inception-v3, SqueezeNet

Other: FP32, batchsize 1, 4x multi-threaded through Arm Compute Library (ArmCL)

Measured individual per-layer performance as well as end-to-end run-time, compared with highly optimized conventional ‘im2row GEMM’ convolution strategy

Benchmark Results



Conclusion

- ML is coming to the edge, hard and fast
- ARM CPUs are already widely deployed at the edge, so optimizing for performance here has immediate impact
- Winograd domain is an alternative to conventional im2row/GEMM convolution that reduces math, but requires care to fully realize benefit
- When done properly, can provide as much as a 2.5x speedup on real hardware for end-to-end model inference

Benefits now available in ArmCL!

Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
شكرًا
תודה