ECE 598NSG/498NSU
Deep Learning in Hardware
Fall 2020

Computational Transforms – Reducing the Complexity of Convolutions

Naresh Shanbhag
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

http://shanbhag.ece.uiuc.edu

Outline

- FFT-based methods
- Winograd-based methods
- Strassen-based methods

- These methods reduce the **computational intensity (CI) of CNNs by reducing complexity of convolutional layers**
  - CONV layers are compute bound → contributes most to CI

Impact of Reduced CI

- Reduced CI implies increased decision throughput (DT)

$$DT = \frac{AP}{CI} = \min\left[\frac{PP}{CI}, OI \times \left(\frac{BW}{CI}\right)\right] = \min\left[\frac{PP}{CI}, \frac{BW}{M_{dec}}\right]$$

- Reduced CI implies lower energy cost

$$E_{dec} = CI\left(E_P + \frac{E_M}{OI}\right)$$

FFT-based Methods

Fast Training of Convolutional Networks through FFTs

(2014)

Michael Mathieu
Courant Institute of Mathematical Sciences
New York University
mathieu@cs.nyu.edu

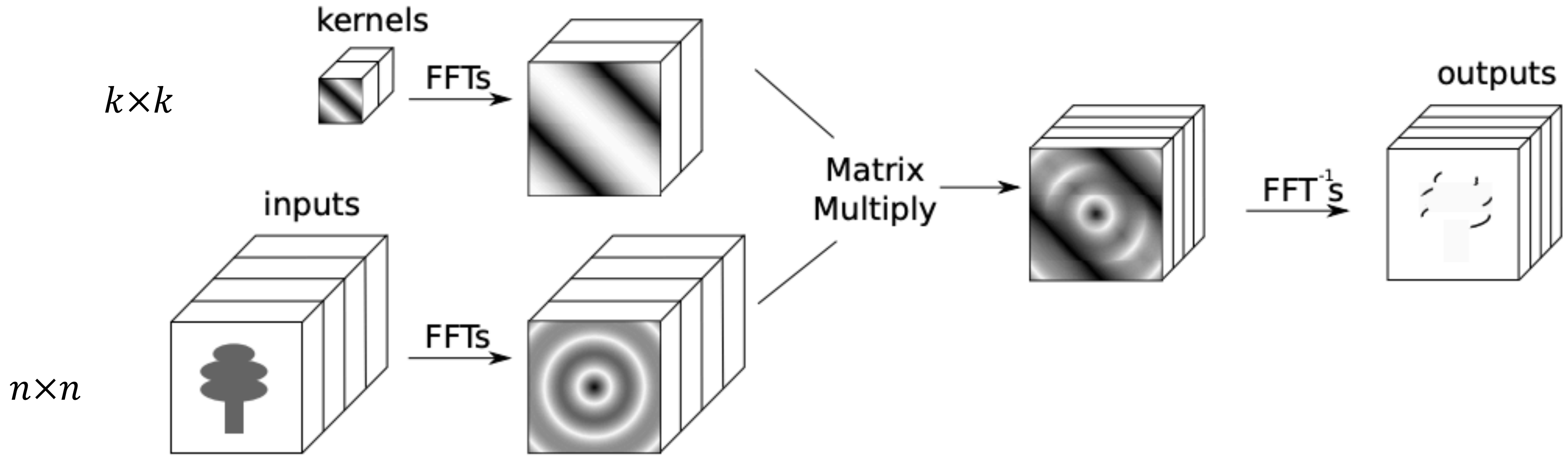
Mikael Henaff
Courant Institute of Mathematical Sciences
New York University
mbh305@nyu.edu

Yann LeCun
Courant Institute of Mathematical Sciences
New York University
yann@cs.nyu.edu

- Use FFTs for implementing convolutions in CNNs
- Effective when the number and size of input FMs is large
- For both inference and training

Method

$$y = W \odot X \leftrightarrow y = DFT^{-1}[DFT(W) * DFT(X)]$$



- need to zero pad kernels to size n

Complexity Comparison

- Direct method: $(n-k+1)^2k^2$ = (# of outputs) × (real MACs/output)

- 2D FFT method: 1 n^2-block length FFT needs –

$$Cn^2\log_2 n^2 \rightarrow 2Cn^2\log_2 n \qquad \text{(real MACs)}$$

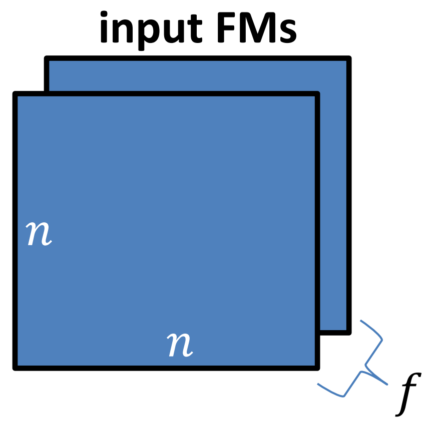
- FFT-based 2D convolution needs: 3 n^2-point FFTs + n^2 complex multiplies

$$6C(n^2\log_2 n) + 4n^2 \quad \text{(real MACs)}$$

- kernel FFTs need to be computed once during inference and reused

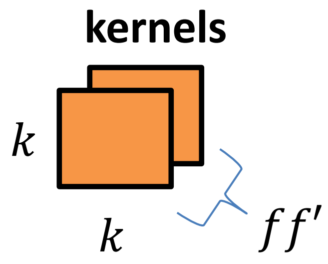
Example

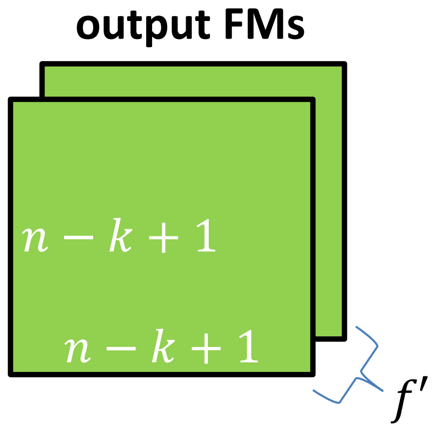
- f: # of input FMs; f': # of output FMs
- image size: $n \times n$; kernel size: $k \times k$; minibatch size: S



- Direct method:

$$S \times f' \times f (n-k+1)^2 k^2$$

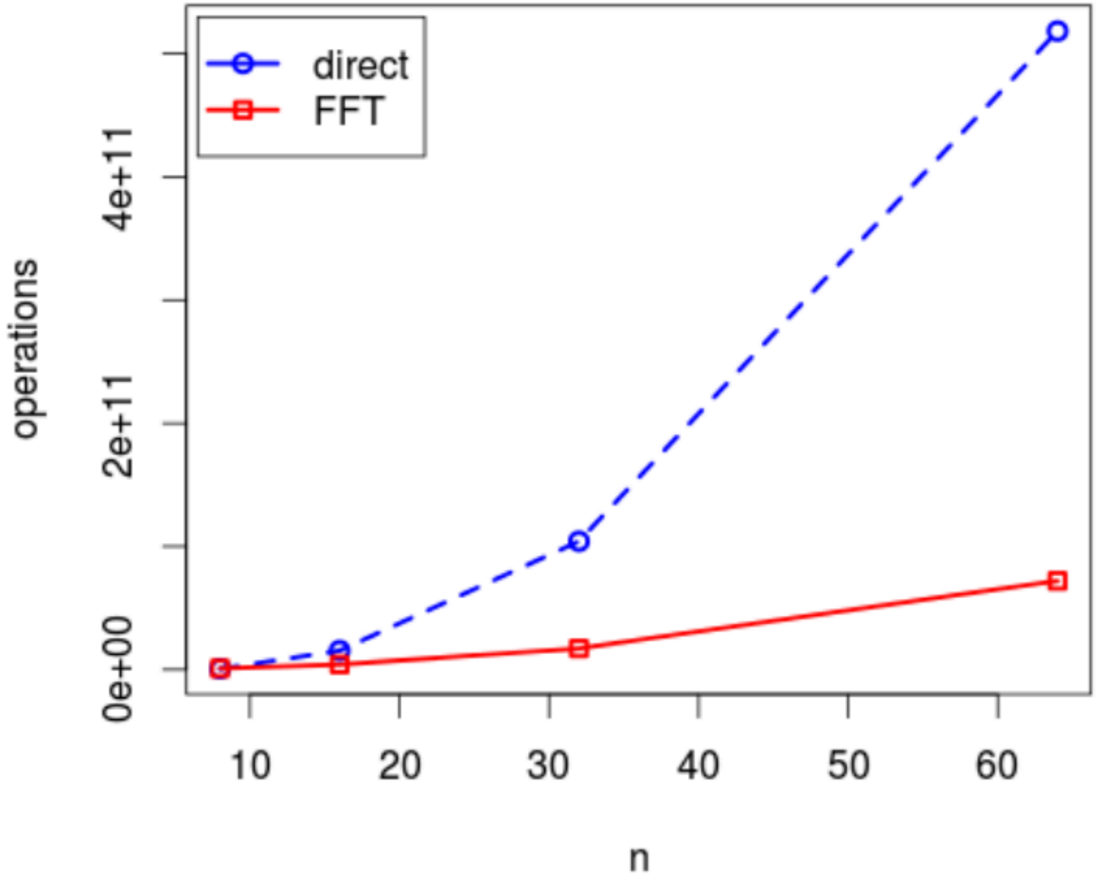




- FFT method:

$$\underbrace{(2Cn^2 \log_2 n)(S \times f + f' \times f)}_{\textbf{FFT}} + \underbrace{4S \times f' \times f \times n^2}_{\textbf{pointwise multiplies}} + \underbrace{S \times f' \times (2Cn^2 \log_2 n)}_{\textbf{IFFT}}$$

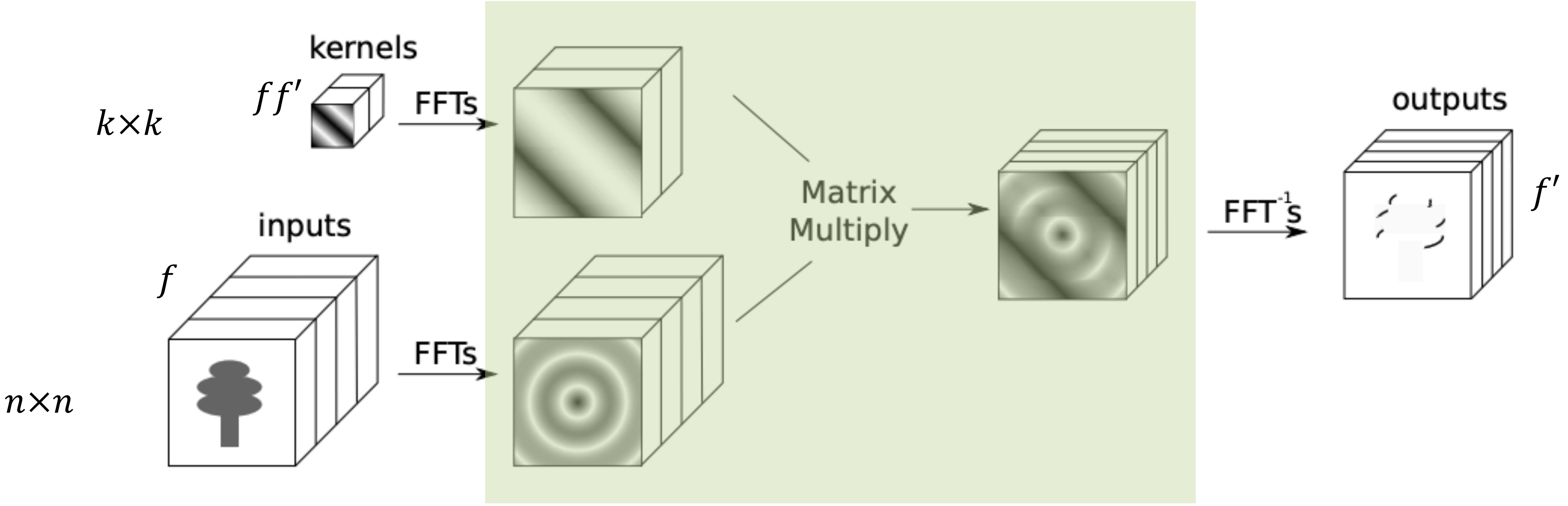
Computational Savings



(image size)

- $S = 128, f = 96, f' = 256, k = 7$
- 2D FFTs parallelized by concatenating two 1D FFTs and running those in parallel

Memory Requirements



- Additional memory needed to store frequency-domain FMs
- $S(f+f')+ff'$ memory locations of size n^2 is needed
- Use conjugate symmetry property of real-valued inputs to reduce memory requirements to: $4n(n+1)[S(f+f')+ff']$ bytes

Compute Time (ms)

(k, n, f, f')	$(11, 32, 3, 96)$	$(7, 32, 96, 256)$	$(5, 16, 256, 384)$	$(5, 16, 384, 384)$	$(3, 16, 384, 384)$
			updateOutput		
Torch7 (custom)	5	178	74	111	57
CudaConv	16	221	98	146	86
FFT	**3**	**34**	**34**	**49**	**49**
			updateGradInput		
Torch7 (custom)	-	197	**76**	**116**	**62**
CudaConv	-	261	108	161	77
FFT	-	**92**	**76**	**116**	116
			accGradParameters		
Torch7 (custom)	39	285	116	174	96
CudaConv	32	403	195	280	178
FFT	**2**	**33**	**32**	**48**	**47**
			Total		
Torch7 (custom)	44	660	266	401	215
CudaConv	48	885	401	587	341
FFT	**5**	**159**	**142**	**213**	**212**

- huge speed-ups!

Ideas for Improvement

- can use larger kernels since they are zero padded anyway
- learn kernels in the frequency-domain directly
- implement non-linearity in frequency domain to avoid IFFT

- impact on sparsity of kernels?

Strength Reduction

Use Case

point-wise complex multiplies

$$y = W \odot X \leftrightarrow y = DFT^{-1}[DFT(W) * DFT(X)]$$

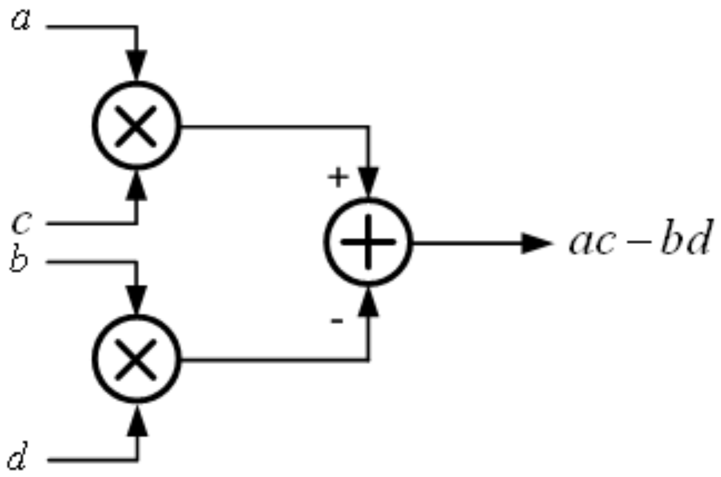
- further reduces the complexity of FFT-based methods by reducing the complexity of point-wise complex multiplications

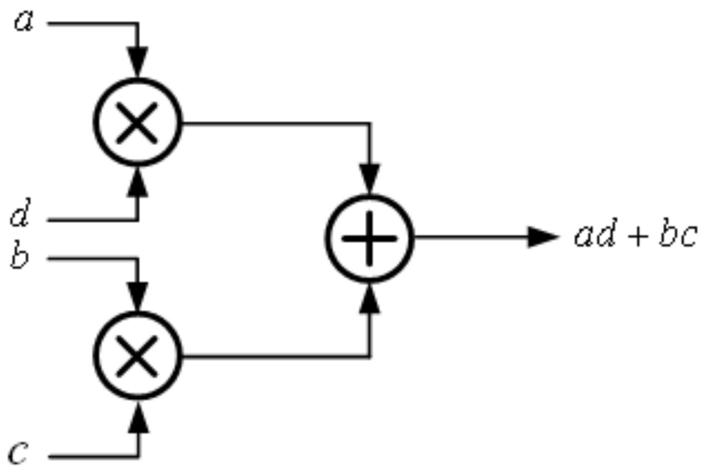
The Method

- Consider complex scalar multiplication

$$(a+jb)(c+jd) = ac - bd + j(ad+bc)$$

- A complex multiplier: 4 real multipliers and 2 real adders

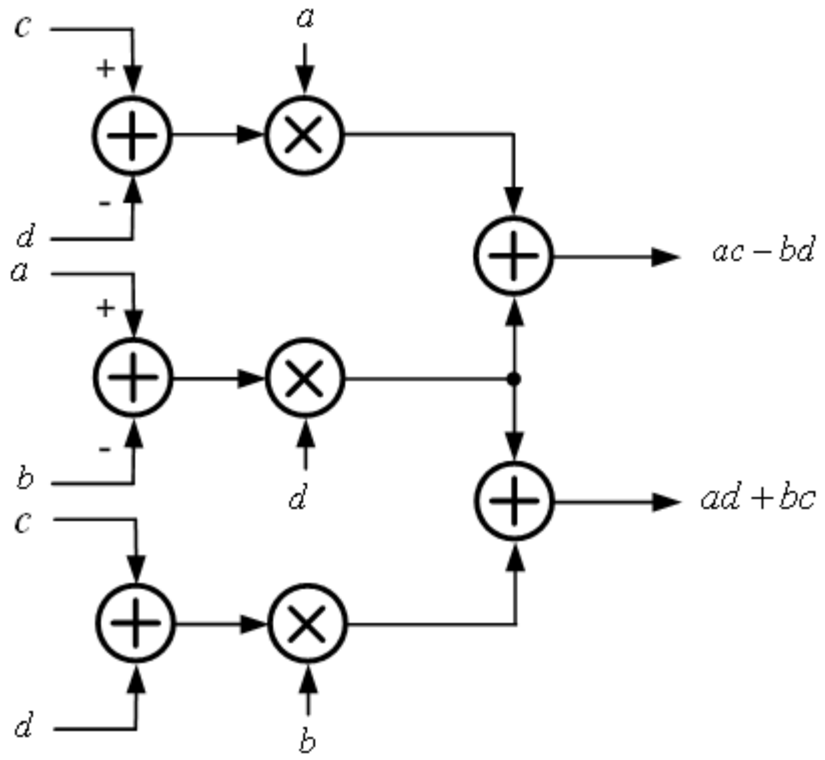




Strength-reduced Complex Multiplier

- Express complex multiplication as:

$$(a-b)d+a(c-d)=ac-bd$$
$$(a-b)d+b(c+d)=ad+bc$$



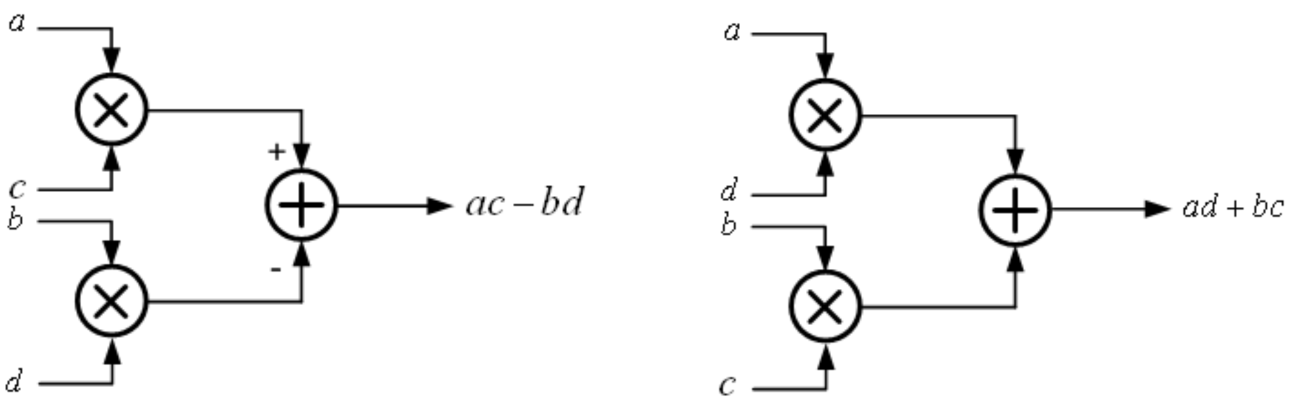
- Alternative formulation

$$X=(a-b)(c+d)=ac+ad-bc-bd$$
$$Y=bc;\ Z=ad$$
$$ac-bd=X+Y-Z;\ ad+bc=Y+Z$$

- 3 (vs. 4) real multipliers and 5 (vs. 2) real adders

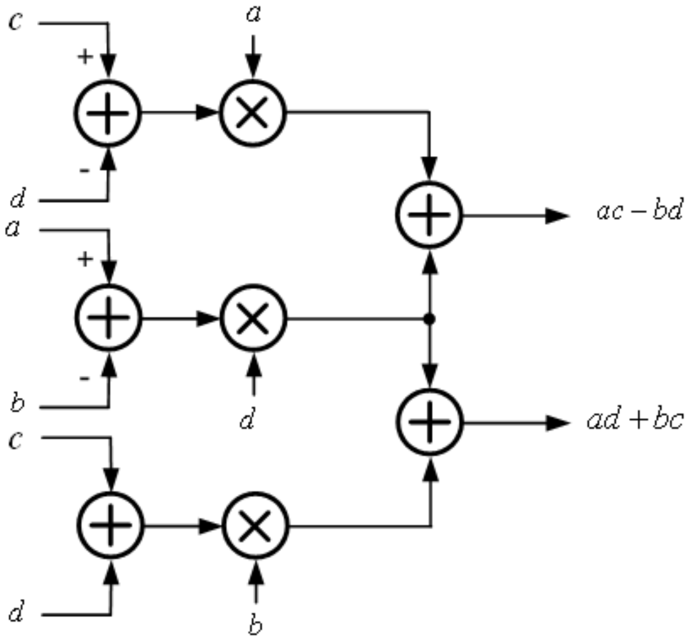
Critical Path Delay

original



$$T_{cp,o} = T_M + T_A$$

Strength-reduced



$$T_{cp,sr} = T_M + 2T_A$$

- Critical path delay increases

Example

- $(a,b) = 12bit$ signed operands
- $(c,d) = 8bit$ signed operands

$$(a-b)d + a(c-d) = ac - bd$$
$$(a-b)d + b(c+d) = ad + bc$$

	Conventional	Strength-Reduced
Multiplier	4 $M_{(12,8)}$ = 320 FA	$M_{(13,8)}$ +2$M_{(12,9)}$ = 270 FA
Adder	2 $A_{(20,20)}$ = 40 FA	$A_{(12,12)}$ + 2$A_{(8,8)}$ +2$A_{(21,21)}$= 49(70) FA
Total **FA:** 1b –full Add	360 FA	319 (340) FA

- *MxN*-bit multiplier complexity: (N-2)(M-2)+2M+4 (FAs)

Strength-reduced FFT-based methods

$$\underbrace{(2Cn^2\log_2 n)(S\times f+f'\times f)}_{\textbf{FFT}}+\underbrace{4S\times f'\times f\times n^2}_{\textbf{pointwise multiplies}}+\underbrace{S\times f'\times(2Cn^2\log_2 n)}_{\textbf{IFFT}}$$

$$\Downarrow$$

$$\underbrace{(2Cn^2\log_2 n)(S\times f+f'\times f)}_{\textbf{FFT}}+\underbrace{3S\times f'\times f\times n^2}_{\textbf{pointwise multiplies}}+\underbrace{S\times f'\times(2Cn^2\log_2 n)}_{\textbf{IFFT}}$$

Strength-reduced Real-valued Convolutions via Polyphase Method

Polyphase Representation

- Let $x[2k]$ and $x[2k+1]$ represent even and odd samples of $x[n]$, respectively.
- Let $h_0[n]$ and $h_1[n]$ represent the even and odd coefficients of $h[n]$, i.e.,

$$H(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} = H_0(z^2) + z^{-1} H_1(z^2)$$

$$H_0(z) = a_0 + a_2 z^{-1} \qquad H_1(z) = a_1 + a_3 z^{-1}$$

- This representation of *h[n]* is called a polyphase representation.

$$X(z) = X_0(z^2) + z^{-1} X_1(z^2)$$

Block Diagram

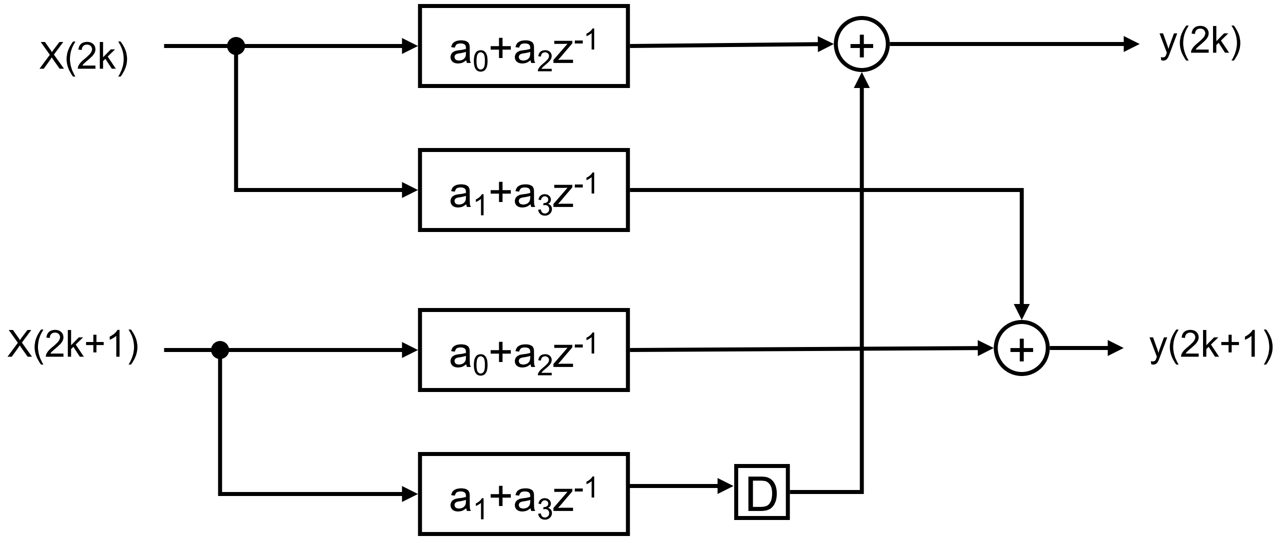
- Therefore:

$$Y(z) = H(z)X(z) = [H_0(z^2) + z^{-1}H_1(z^2)]\times[X_0(z^2) + z^{-1}X_1(z^2)]$$

$$Y_{2k} = X_{2k}H_0 + X_{2k+1}z^{-2}H_1 \qquad Y_{2k+1} = X_{2k}H_1 + X_{2k+1}H_0$$

4, length-2 filters shown below. No change in complexity:

2-parallel FIR filter



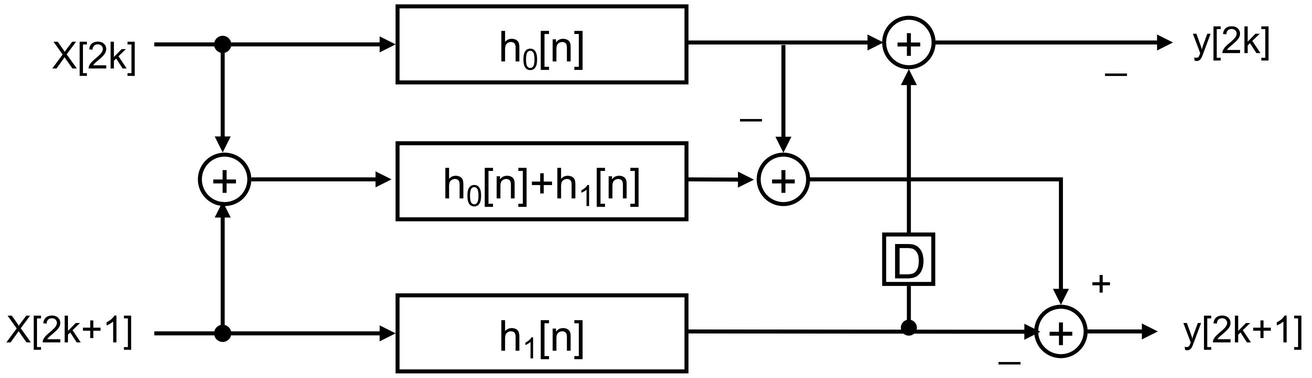
Strength-Reduced Real Convolution

- compute the following filter output:

$$(X_{2k} + X_{2k+1})\,(H_0 + H_1) = X_{2k}H_0 + (X_{2k}H_1 + X_{2k+1}H_0) + X_{2k+1}H_1$$

- bracketed term in RHS is the desired output *y*[2*k*+1]; the remaining terms are the outputs of the top and bottom filters
- The following strength-reduced parallel FIR filter is easily derived:

A strength-reduced convolver



Winograd Method

Shmuel Winograd. Arithmetic complexity of computations, volume 33. Siam, 1980.

Minimal Complexity 1D Convolution Algorithm

- $F(M,R)$: minimal complexity algorithm for computing M outputs of an R-tap kernel
- $\mu(F(M,R))$: multiplicative complexity of $F(M,R)$

$$\mu\big(F(M,R)\big) = M + R - 1$$

- Note: $M + R - 1$ is also equal to the number of inputs needed to generate M outputs of an R-tap kernel

Example: F(2,3)

- Standard algorithm: $2 \times 3 = 6$ multiplications and 4 additions
- Winograd's minimal algorithm: $\mu\big(F(2,3)\big) = 2 + 3 - 1 = 4$

- $F(2,3)$: $\mu\big(F(2,3)\big) = 2 + 3 - 1 = 4$ multiplications and 11 additions

Post-processing

$$y_1 = w_1 x_1 + w_2 x_2 + w_3 x_3 = m_1 + m_2 + m_3$$
$$y_2 = w_2 x_1 + w_3 x_2 + w_4 x_3 = m_2 - m_3 - m_4$$

Pre-processing

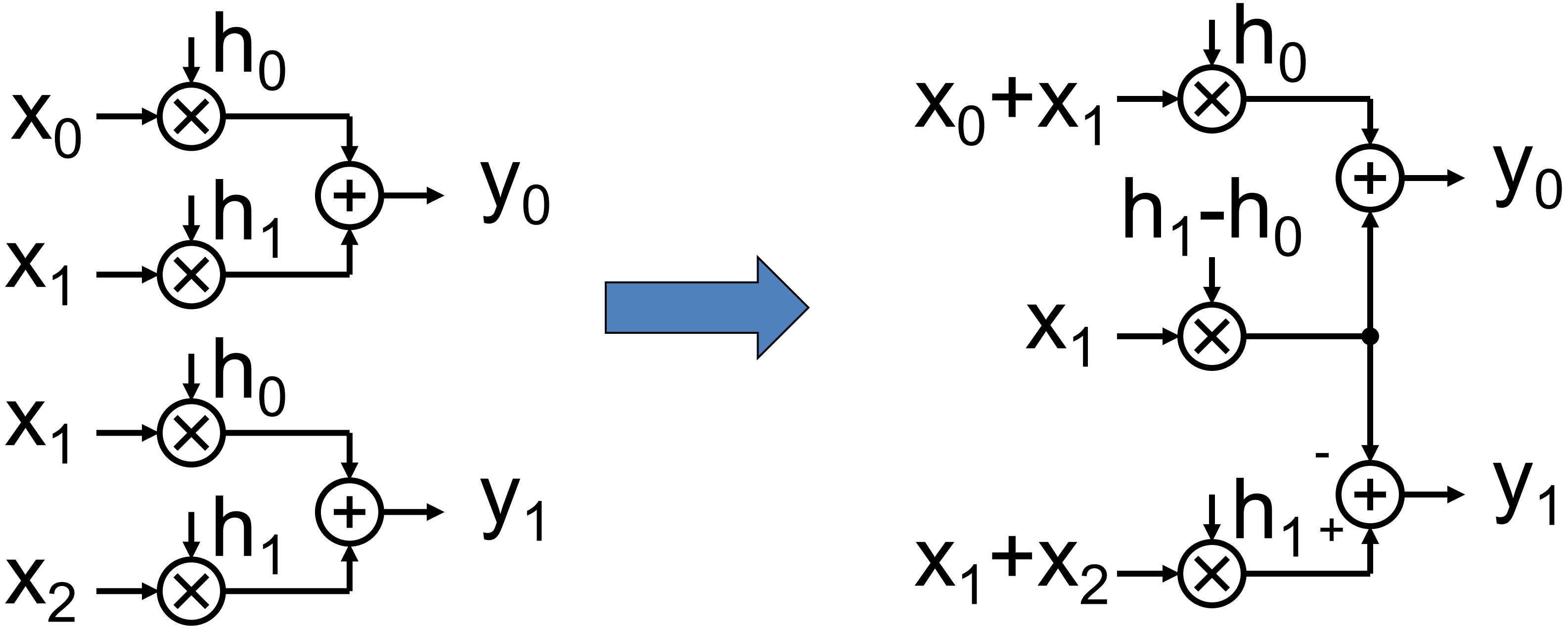
$$m_1 = (w_1 - w_3) x_1; m_2 = (w_2 + w_3) \frac{x_1 + x_2 + x_3}{2}$$
$$m_4 = (w_2 - w_4) x_3; m_3 = (w_3 - w_2) \frac{x_1 - x_2 + x_3}{2}$$

Winograd's Approach

F(2,2): two-tap, two output filter

$y_0 = x_0h_0+x_1h_1 = (x_0+x_1)h_0 + x_1(h_1-h_0)$

$y_1 = x_1h_0+x_2h_1 = (x_1+x_2)h_1 - x_1(h_1-h_0)$



Winograd's Approach for F(4,4)

$$\begin{pmatrix} \begin{pmatrix} x_0 & x_1 \\ x_1 & x_2 \end{pmatrix} & \begin{pmatrix} x_2 & x_3 \\ x_3 & x_4 \end{pmatrix} \\ \begin{pmatrix} x_2 & x_3 \\ x_3 & x_4 \end{pmatrix} & \begin{pmatrix} x_4 & x_5 \\ x_5 & x_6 \end{pmatrix} \end{pmatrix} \begin{pmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{pmatrix} = \begin{pmatrix} X_0 & X_1 \\ X_1 & X_2 \end{pmatrix} \begin{pmatrix} H_0 \\ H_1 \end{pmatrix} = \begin{pmatrix} Y_0 \\ Y_1 \end{pmatrix}$$

$$M_1 = (X_0 - X_1)H_0; \quad Y_0 = M_1 + M_2; \quad Y_1 = M_2 - M_3;$$

$$M_2 = X_1(H_0 + H_1)$$

$$M_3 = (X_1 - X_2)H_1$$

- F(16,16): 256 multiplies + 240 adds (conventional)
  81 multiplies + 260 adds (Winograd)
- F(m,n): minimum number of multiplies = m+n-1

Minimal Complexity 2D Convolution Algorithm

- minimal complexity needed to compute $M{\times}N$ outputs of an $R{\times}S$ kernel

$$\mu\big(F(M{\times}N, R{\times}S)\big) = \mu(M,R){\times}\mu(N,S) = (M+R-1)(N+S-1)$$

- also requires 1 multiplication per input

Strassen Method

Minimizing Computation in Convolutional Neural Networks

Jason Cong and Bingjun Xiao

Computer Science Department,
University of California,
Los Angeles, CA 90095, USA
{cong,xiao}@cs.ucla.edu

The Method

- Wish to compute: $Y = WX$
- Block partitioning step:

$$W = \begin{pmatrix} W_{1,1} & W_{1,2} \\ W_{2,1} & W_{2,2} \end{pmatrix},\ X = \begin{pmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{pmatrix},\ Y = \begin{pmatrix} Y_{1,1} & Y_{1,2} \\ Y_{2,1} & Y_{2,2} \end{pmatrix}$$

$$\begin{aligned} Y_{1,1} &= W_{1,1} \times X_{1,1} + W_{1,2} \times X_{2,1} \\ Y_{1,2} &= W_{1,1} \times X_{1,2} + W_{1,2} \times X_{2,2} \\ Y_{2,1} &= W_{2,1} \times X_{1,1} + W_{2,2} \times X_{2,1} \\ Y_{2,2} &= W_{2,1} \times X_{1,2} + W_{2,2} \times X_{2,2} \end{aligned}$$

- No change in the total number of multiplies – 8 multiplies & 4 additions

Pre & Post Processing

Pre-processing

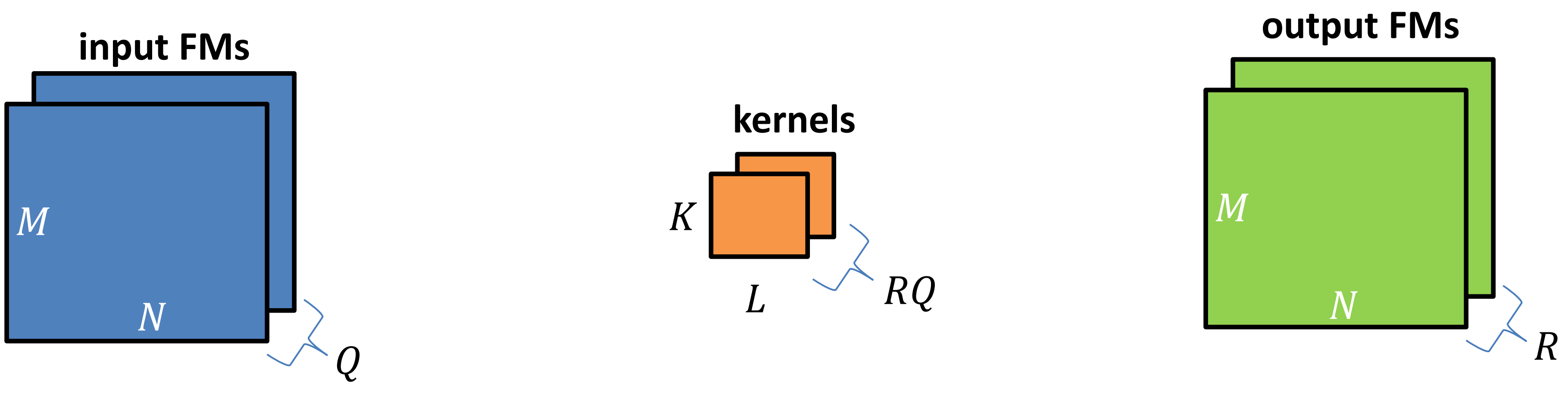
$$
\begin{aligned}
M_1 &:= (W_{1,1} + W_{2,2}) \times (X_{1,1} + X_{2,2}) \\
M_2 &:= (W_{2,1} + W_{2,2}) \times X_{1,1} \\
M_3 &:= W_{1,1} \times (X_{1,2} - X_{2,2}) \\
M_4 &:= W_{2,2} \times (X_{2,1} - X_{1,1}) \\
M_5 &:= (W_{1,1} + W_{1,2}) \times X_{2,2} \\
M_6 &:= (W_{2,1} - W_{1,1}) \times (X_{1,1} + X_{1,2}) \\
M_7 &:= (W_{1,2} - W_{2,2}) \times (X_{2,1} + X_{2,2})
\end{aligned}
$$

Post-processing

$$
\begin{aligned}
Y_{1,1} &= M_1 + M_4 - M_5 + M_7 \\
Y_{1,2} &= M_3 + M_5 \\
Y_{2,1} &= M_2 + M_4 \\
Y_{2,2} &= M_1 - M_2 + M_3 + M_6
\end{aligned}
$$

- requires 7 multiplies and 18 additions → 1 multiply better (much) more than 14× more complex than 1 add

Convolutional Matrix Multiply (MM)



- Wish to compute: $Y = W \times X$ (element-wise multiplies are convolutions between FM and filter kernel)

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_Q \end{pmatrix}, \vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_R \end{pmatrix}, W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1Q} \\ w_{21} & w_{22} & \cdots & w_{2Q} \\ \vdots & \vdots & \ddots & \vdots \\ w_{R1} & w_{R2} & \cdots & w_{RQ} \end{pmatrix}$$

Properties of Convolutional MM

- additivity of filter kernels: $W1 + W2 = W3$ (element-wise addition)
- associativity:

$$(W1 + W2) \times X = W1 \times X + W2 \times X.$$

$$W \times (X1 + X2) = W \times X1 + W \times X2.$$

- Apply Strassen method to Convolutional MM

Course Web Page

https://courses.grainger.illinois.edu/ece598nsg/fa2020/
https://courses.grainger.illinois.edu/ece498nsu/fa2020/

http://shanbhag.ece.uiuc.edu