Survey of DNN Hardware

Hardware are targeting deep learning

- CPU
  - Intel Knights Landing
  - Intel Knight Mill
- GPU
  - PASCAL
  - VOLTA
- System for deep learning
  - Nvidia DGX-1 (2016)
- Cloud Systems for Deep Learning
- SOCs for Deep Learning Inference
- FPGAs for Deep Learning

CPUs Are Targeting Deep Learning

Intel Knights Landing (2016)

- 7 TFLOPS FP32
- 16GB MCDRAM– 400 GB/s
- 245W TDP
- 29 GFLOPS/W (FP32)
- 14nm process

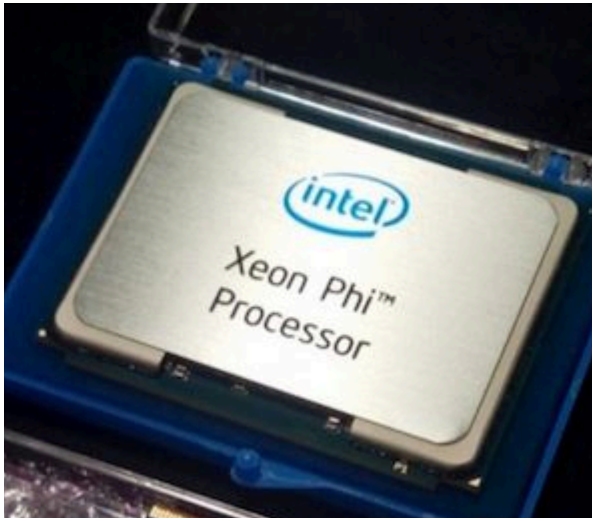
Knights Mill: next gen Xeon Phi “optimized for deep learning”

Intel announced the addition of new vector instructions for deep learning (AVX512-4VNNIW and AVX512-4FMAPS), October 2016

Image Source: Intel, Data Source: Next Platform

CPUs Are Targeting Deep Learning

- Knights Mill
  - Intel's codename for a Xeon Phi product specialized in deep learning,
  - Initially released in December 2017
  - Knights Mill includes optimizations for better utilization of AVX-512 instructions and enables 4-way hyperthreading.
  - Single-precision and variable-precision floating-point performance increased.



GPUs Are Targeting Deep Learning

Nvidia PASCAL GP100 (2016)

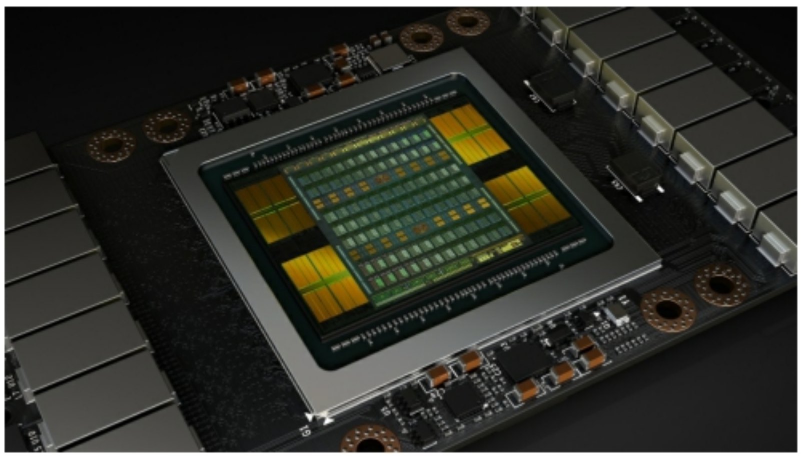
- **10/20 TFLOPS FP32/FP16**
- 16GB HBM – 750 GB/s
- 300W TDP
- 33/67 GFLOPS/W (FP32/FP16)
- 16nm process
- 160GB/s NV Link

FP16 support to perform two FP16 operations on a single precision core for faster dep learning computation

Source: Nvidia

GPUs Are Targeting Deep Learning

Nvidia VOLTA GV100 (2017)



- 15 TFLOPS FP32
- 16GB HBM2 – 900 GB/s
- 300W TDP
- 50 GFLOPS/W (FP32)
- 12nm process
- 300GB/s NV Link2

Tensor Core….

Source: Nvidia

GV100 – “Tensor Core”

$$\underbrace{D}_{\text{FP16 or FP32}} = \left(\underbrace{\begin{matrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{matrix}}_{\text{FP16}} \right) \left(\underbrace{\begin{matrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{matrix}}_{\text{FP16}} \right) + \left(\underbrace{\begin{matrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{matrix}}_{\text{FP16 or FP32}} \right)$$

Efficient Execution of 4x4 FP16 Multiplication and Addition

Tensor Core….

- 120 TFLOPS (FP16)
- 400 GFLOPS/W (FP16)

Systems for Deep Learning

Nvidia DGX-2 (2018)

- 2 peta FLOPS
- 16× Tesla V100, Dual Xeon
- 512GB GPU Memory
- 12 NVIDIA NVSwitch
- Optimized DL Software
- 7 TB SSD Storage
- Dual 10GbE, 8X 100Gb
- 10000W

Cloud Systems for Deep Learning

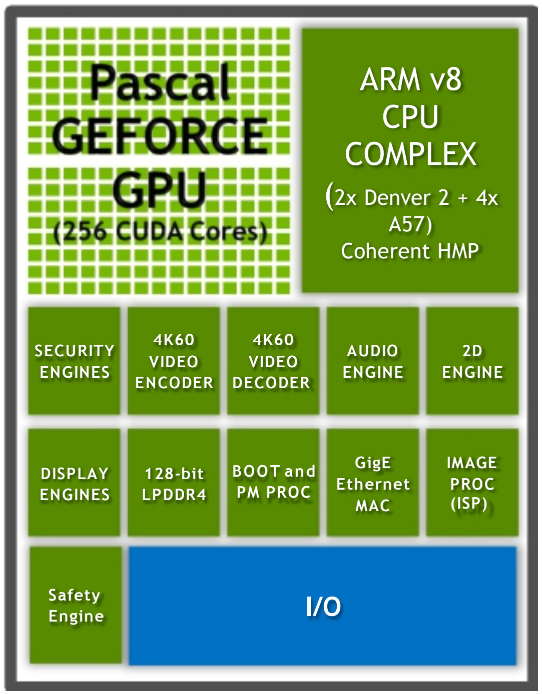
Facebook’s Deep Learning Machine

- Open Rack Compliant
- Powered by 8 Tesla M40 GPUs
- 2x Faster Training for Faster Deployment
- 2x Larger Networks for Higher Accuracy

Source: Facebook

SOCs for Deep Learning Inference

Nvidia Tegra - Parker



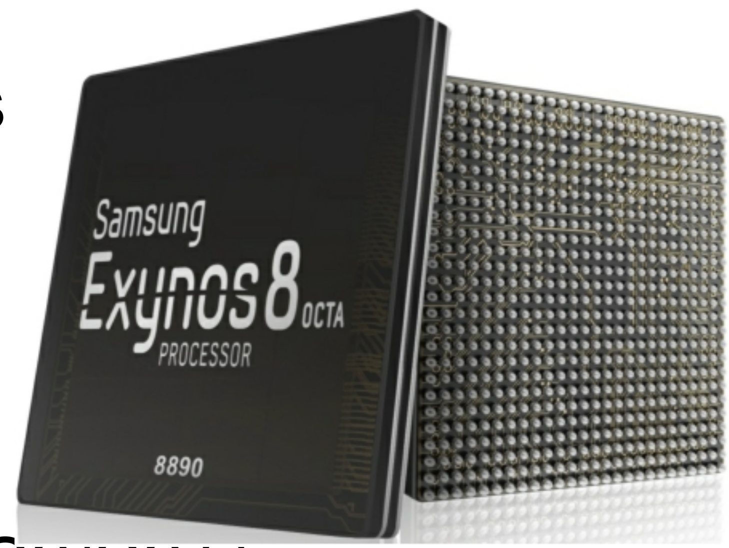
- GPU: 1.5 TeraFLOPS FP16
- 4GB LPDDR4 @ 25.6 GB/s
- 15 W TDP
  (1W idle, <10W typical)
- 100 GFLOPS/W (FP16)
- 16nm process

Xavier: next gen Tegra to be an “AI supercomputer”

Source: Nvidia

Mobile SOCs for Deep Learning

- Samsung Exynos (ARM Mali)
  - Exynos 8 Octa 8890
  - GPU: 0.26 TFLOPS
  - LPDDR4 @ 28.7 GB/s
  - 14nm process
- Source: Wikipedia
- Newer version
  - Exynos 9 Octa 8895 (S9/S9+)
  - Exynos 9 Octa 9820 (S10/S10+)



FPGAs for Deep Learning

Intel/Altera Stratix 10

- 10 TFLOPS FP32
- HBM2 integrated
- Up to 1 GHz
- 14nm process
- 80 GFLOPS/W

Xilinx Virtex UltraSCALE+

- DSP: up to 21.2 TMACS
- DSP: up to 890 MHz
- Up to 500Mb On-Chip Memory
- 16nm process

Kernel Computation

Convolution (CONV) Layer

- Convert to matrix mult. using the **Toeplitz Matrix**

Filter, Input Fmap, Output Fmap

Convolution:

$$\underset{\text{Filter}}{\begin{bmatrix}1&2\\3&4\end{bmatrix}} * \underset{\text{Input Fmap}}{\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}} = \underset{\text{Output Fmap}}{\begin{bmatrix}1&2\\3&4\end{bmatrix}}$$

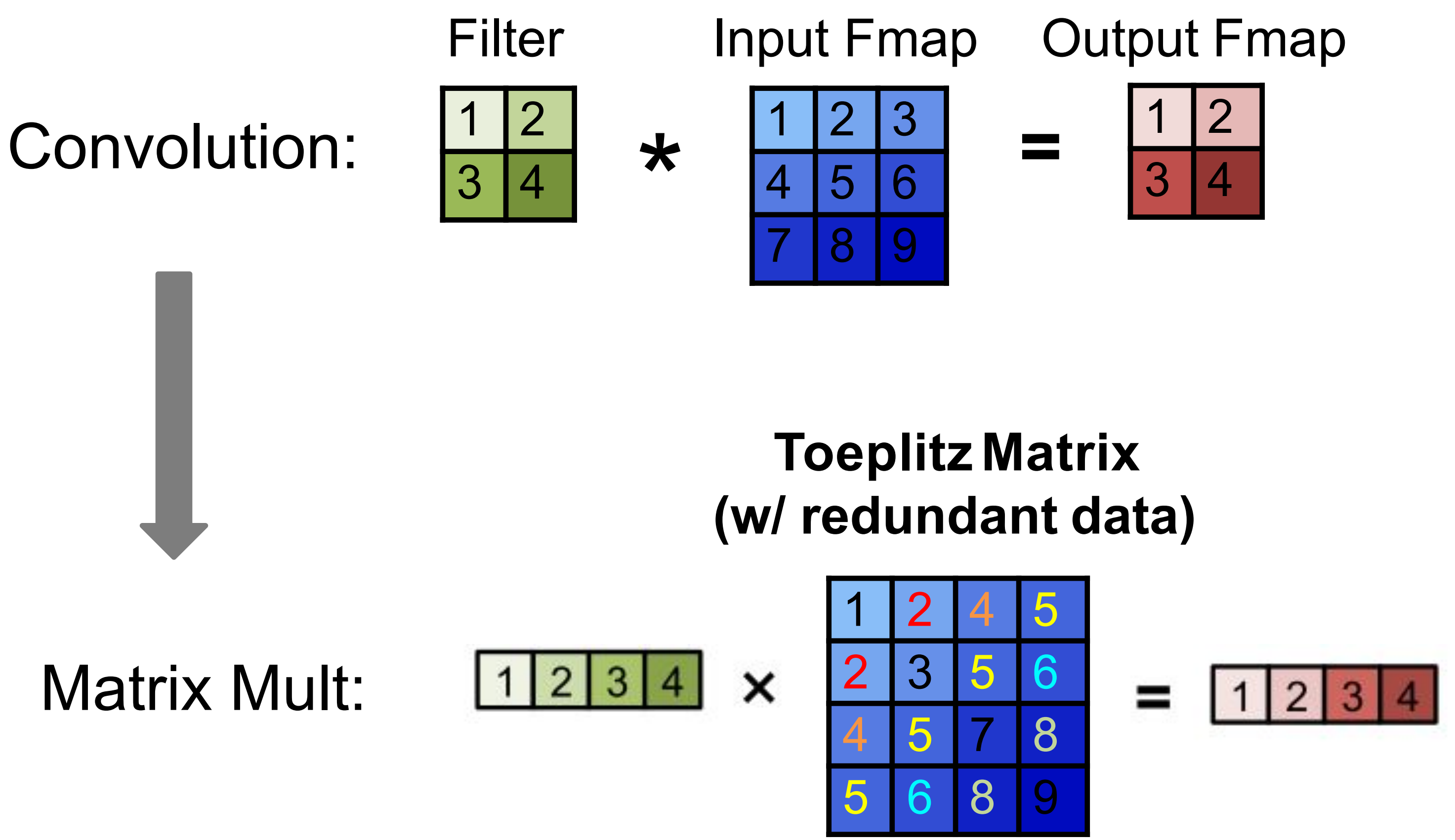
Toeplitz Matrix
(w/ redundant data)

Matrix Mult:

$$\begin{bmatrix}1&2&3&4\end{bmatrix} \times \begin{bmatrix}1&2&4&5\\2&3&5&6\\4&5&7&8\\5&6&8&9\end{bmatrix} = \begin{bmatrix}1&2&3&4\end{bmatrix}$$

Convolution (CONV) Layer

- Convert to matrix mult. using the **Toeplitz Matrix**



Data is repeated

Computational Transforms

Computation Transformations

- Goal: Bitwise same result, but reduce number of operations
- Focuses mostly on compute

Strassen

$$\overbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}^{B} = \overbrace{\begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}}^{C}$$

8 multiplications + 4 additions

P1 = a(f – h)
P2 = (a + b)h
P3 = (c + d)e
P4 = d(g – e)

P5 = (a + d)(e + h)
P6 = (b - d)(g + h)
P7 = (a – c)(e + f)

$$AB = \begin{bmatrix} P5 + P4 - P2 + P6 & P1 + P2 \\ P3 + P4 & P1 + P5 - P3 - P7 \end{bmatrix}$$

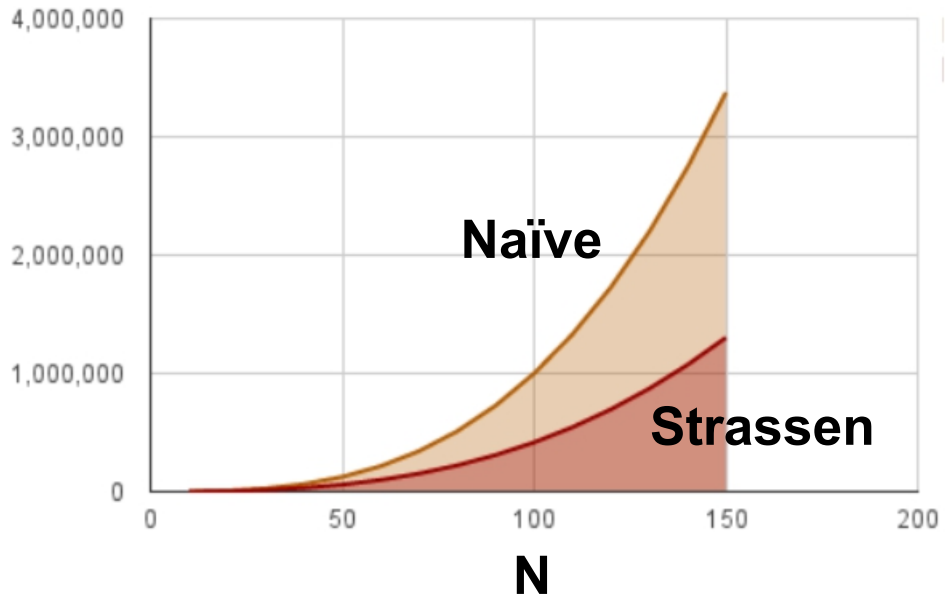
7 multiplications + 18 additions

[Cong et al., ICANN, 2014]

Strassen

- Reduce the complexity of matrix multiplication from **Θ(N3)** to **Θ(N2.807)** by reducing multiplication

Complexity



Comes at the price of reduced numerical stability and requires significantly more memory

Image Source: http://www.stoimen.com/blog/2012/11/26/computer-algorithms-strassens-matrix-multiplication/

Winograd 1D – F(2,3)

- Targeting convolutions instead of matrix multiply
- Notation: F(size of output, filter size)

$$F(2,3) = \begin{array}{c} \text{input} \\ \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \end{array} \begin{array}{c} \text{filter} \\ \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \end{array} = \begin{bmatrix} d_0\, g_0 + d_0\, g_1 + d_0\, g_2 \\ d_1\, g_0 + d_1\, g_1 + d_1\, g_2 \end{bmatrix}$$

6 multiplications + 4 additions

[Lavin et al., ArXiv 2015]

Winograd 1D – F(2,3)

- Targeting convolutions instead of matrix multiply
- Notation: F(size of output, filter size)

$$F(2,3) = \begin{array}{c} \text{input} \\ \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \end{array} \begin{array}{c} \text{filter} \\ \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} \end{array} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

$$m_1 = (d_0 - d_2)g_0 \qquad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$
$$m_4 = (d_1 - d_3)g_2 \qquad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

4 multiplications + 12 additions + 2 shifts

[Lavin et al., ArXiv 2015]

Winograd 2D - F(2x2, 3x3)

- 1D Winograd is nested to make 2D Winograd

Filter * Input Fmap = Output Fmap

$$\begin{bmatrix} g_{00} & g_{01} & g_{02} \\ g_{10} & g_{11} & g_{12} \\ g_{20} & g_{21} & g_{22} \end{bmatrix} * \begin{bmatrix} d_{00} & d_{01} & d_{02} & d_{03} \\ d_{10} & d_{11} & d_{12} & d_{13} \\ d_{20} & d_{21} & d_{22} & d_{23} \\ d_{30} & d_{31} & d_{32} & d_{33} \end{bmatrix} = \begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix}$$

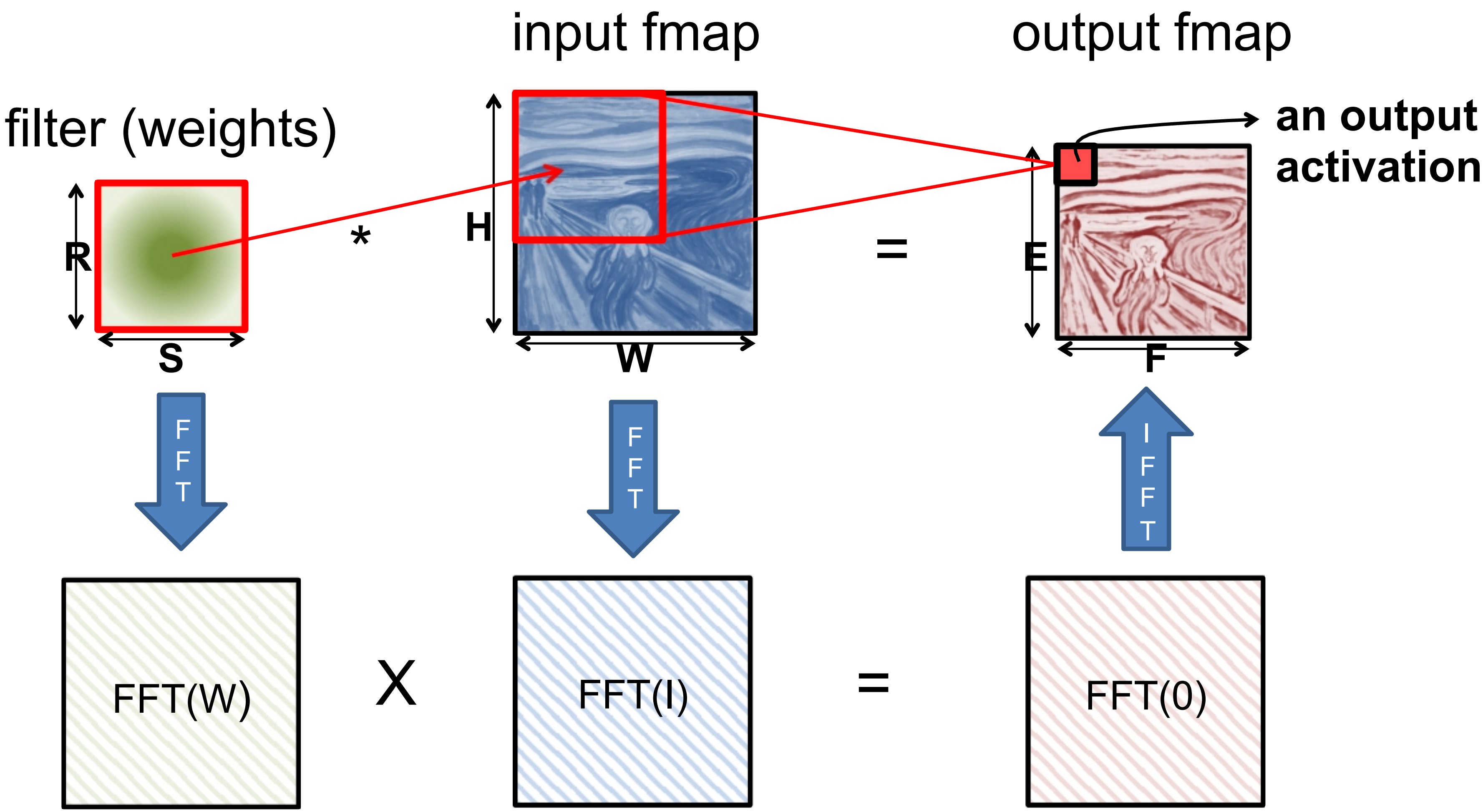
Original: 36 multiplications

Winograd: 16 multiplications → 2.25 times reduction

Winograd Summary

- Winograd is an optimized computation for convolutions
- It can significantly reduce multiplies
  - For example, for 3x3 filter by 2.25X
- But, each filter size is a different computation.

FFT Flow



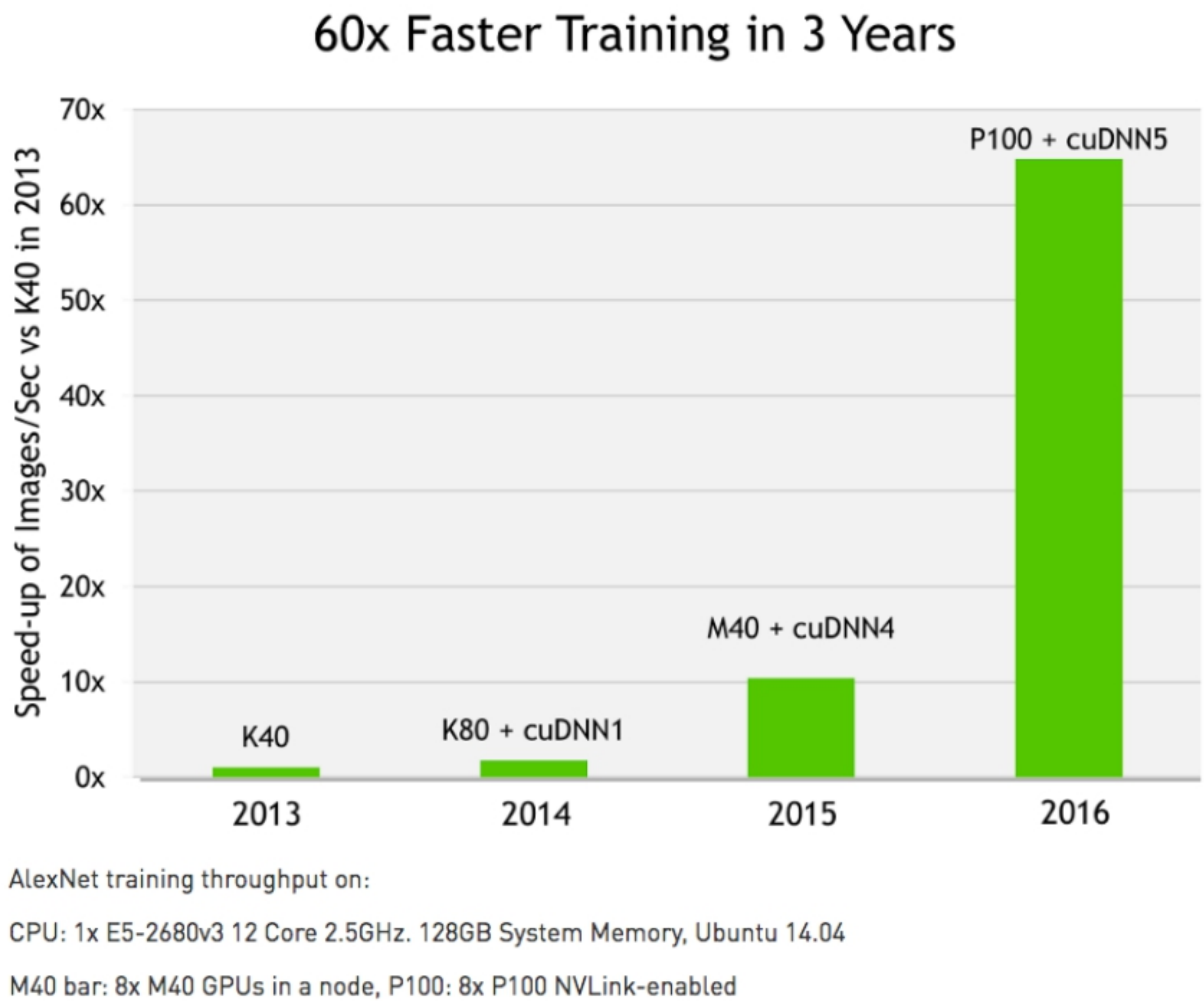
input fmap
output fmap
filter (weights)
an output
activation
R
S
*
H
W
=
E
F
F
F
T
F
F
T
I
F
F
T
FFT(W)
X
FFT(I)
=
FFT(0)

FFT Overview

- **Convert filter and input to frequency domain to make convolution a simple multiply then convert back to time domain.**
- **Convert direct convolution $O(N_o^2 N_f^2)$ computation to $O(N_o^2 \log_2 N_o)$**
- **So note that computational benefit of FFT decreases with decreasing size of filter**

[Mathieu et al., ArXiv 2013, Vasilache et al., ArXiv 2014]

cuDNN: Speed up with Transformations



Source: Nvidia

Backup Slides