INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

A GENTLE INTRODUCTION TO HARDWARE ARCHITECTURES FOR DNN ACCELERATION

Vaader Reading Group, 06/05/21

Mickaël Dardaillon - mdardail@insa-rennes.fr

Menu du jour

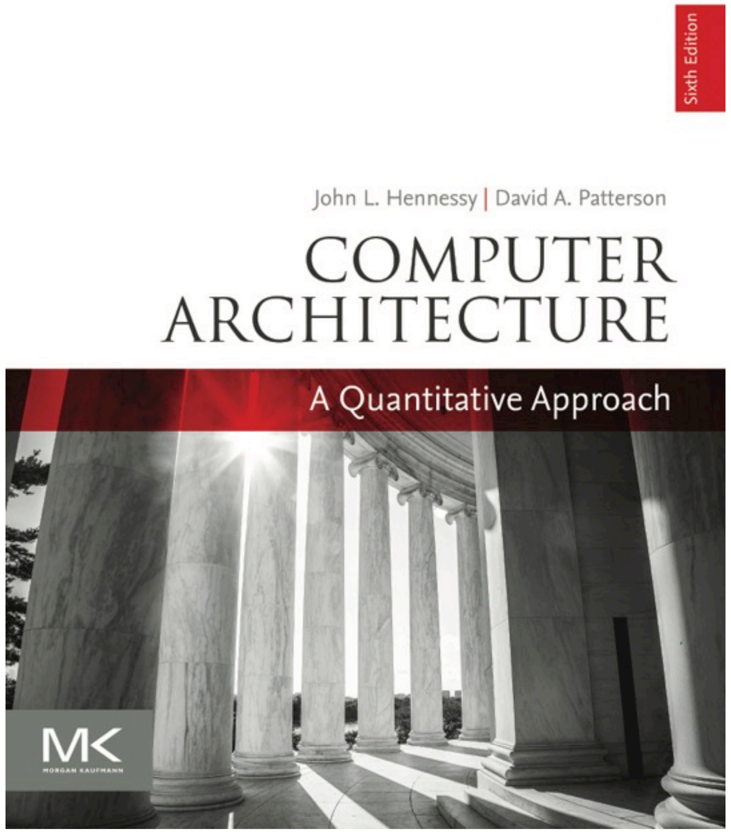
DNN layers
Architecture support

GPUs
SIMT Machines

Systolic Arrays
Domain-Specific Architectures

DNN Optimizations
for hardware acceleration

Chapter 4: Data-Level Parallelism
Chapter 7: Domain-Specific Architectures



DNN LAYERS

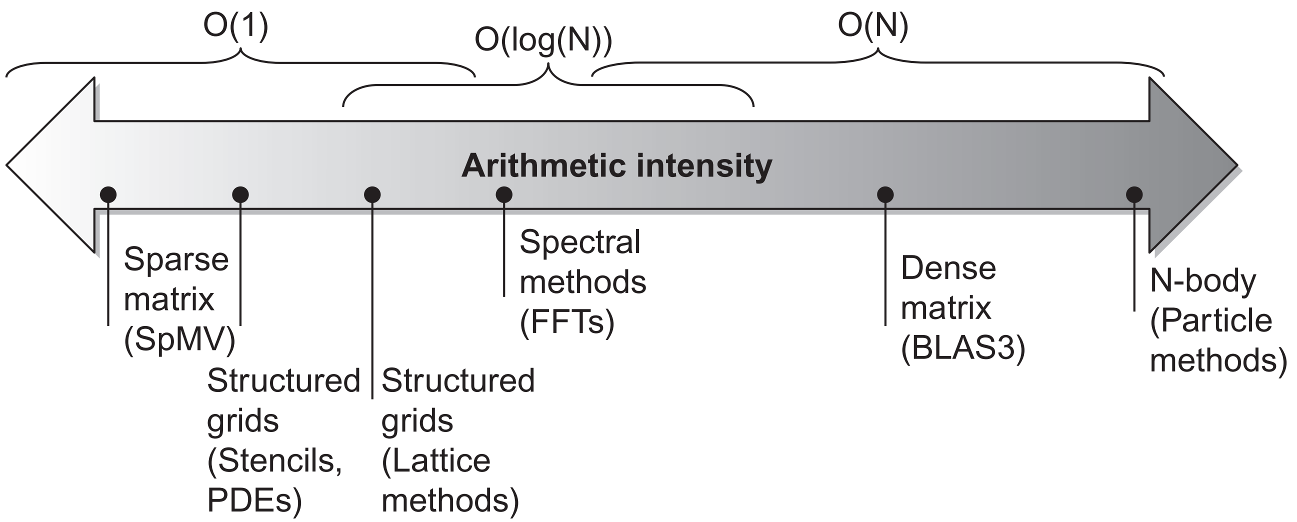
Architecture support

Arithmetic Intensity

Arithmetic intensity is the ratio of floating-point operations per byte of memory accessed.

It is computed by taking the total number of operations for a program and dividing it by the number of data bytes transferred during program execution.

$$arithmetic\ intensity = \frac{nb\ operations}{nb\ data\ access}$$



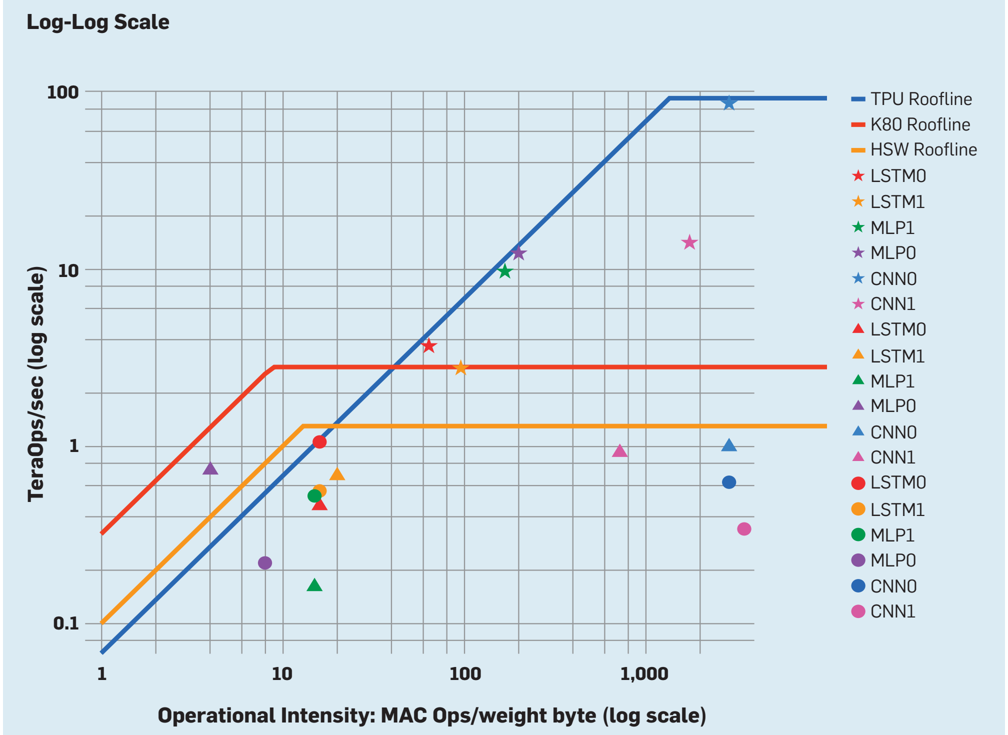
J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 2018.

Roofline model

A visual performance model to find bound and bottlenecks.

The horizontal line shows the peak computational performance, the diagonal is the peak memory performance.

Figure 3. The rooflines of TPUs, CPUs, and GPUs combined into a single log-log graph. Stars are for the TPU, triangles for the K80, and circles for Haswell; all TPU stars are at or above the other two rooflines.



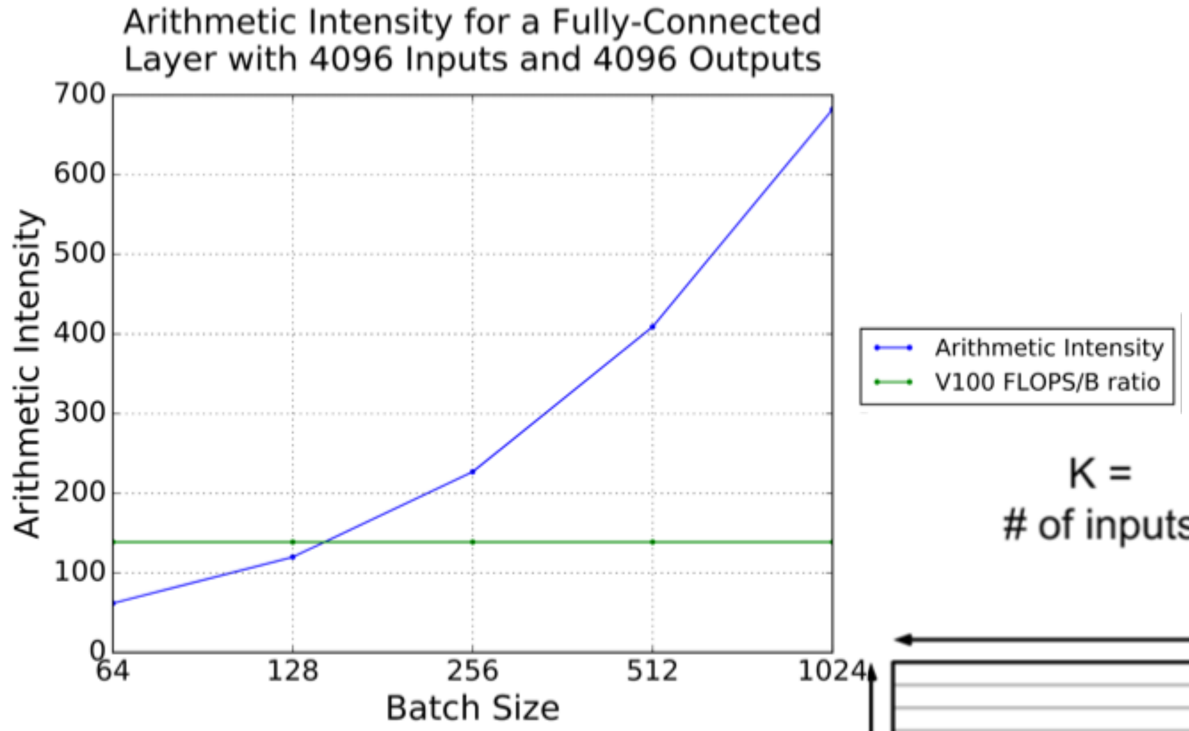
An operational intensity is a vertical line, which cross the roofline at its peak performance.

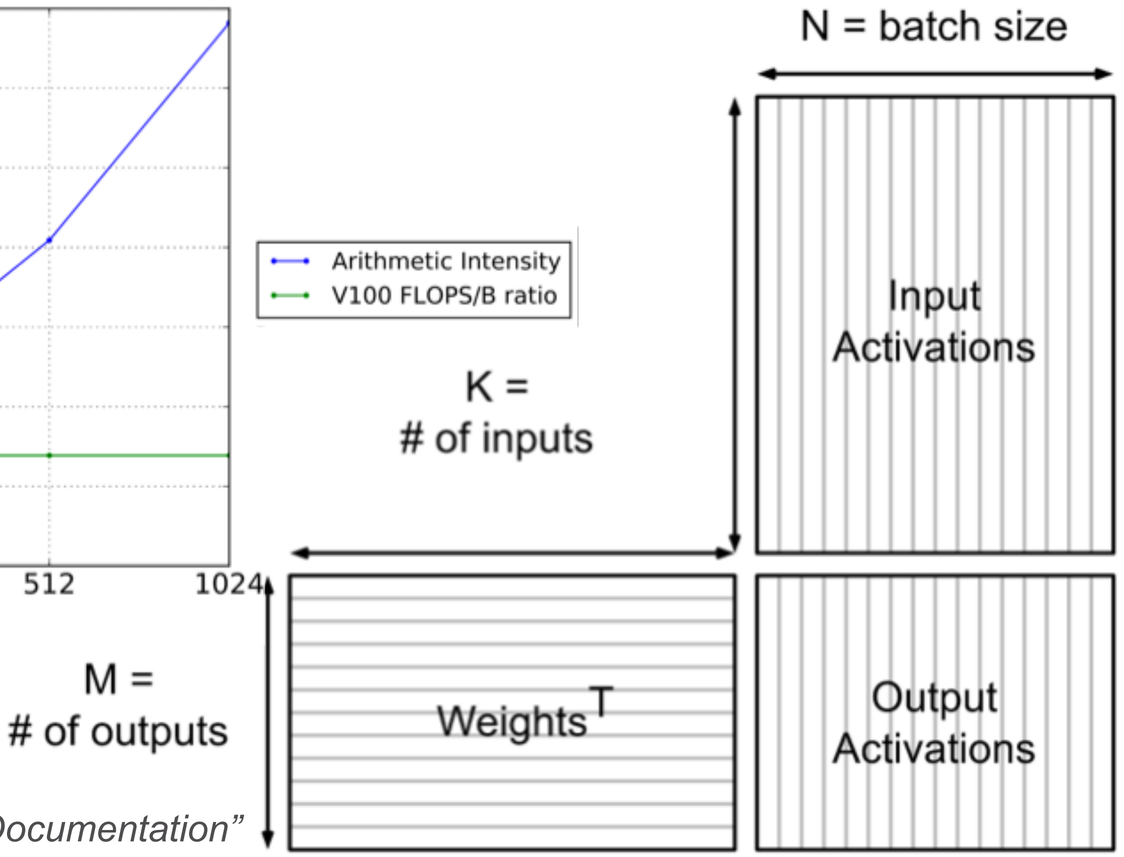
N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," Commun. ACM, vol. 61, no. 9, pp. 50–59, 2018.

Fully Connected Layer

Equivalent to Matrix Multiplication

$$intensity = \frac{2 \times N \times M \times K}{2 \times (N \times K + M \times K + N \times M)}$$

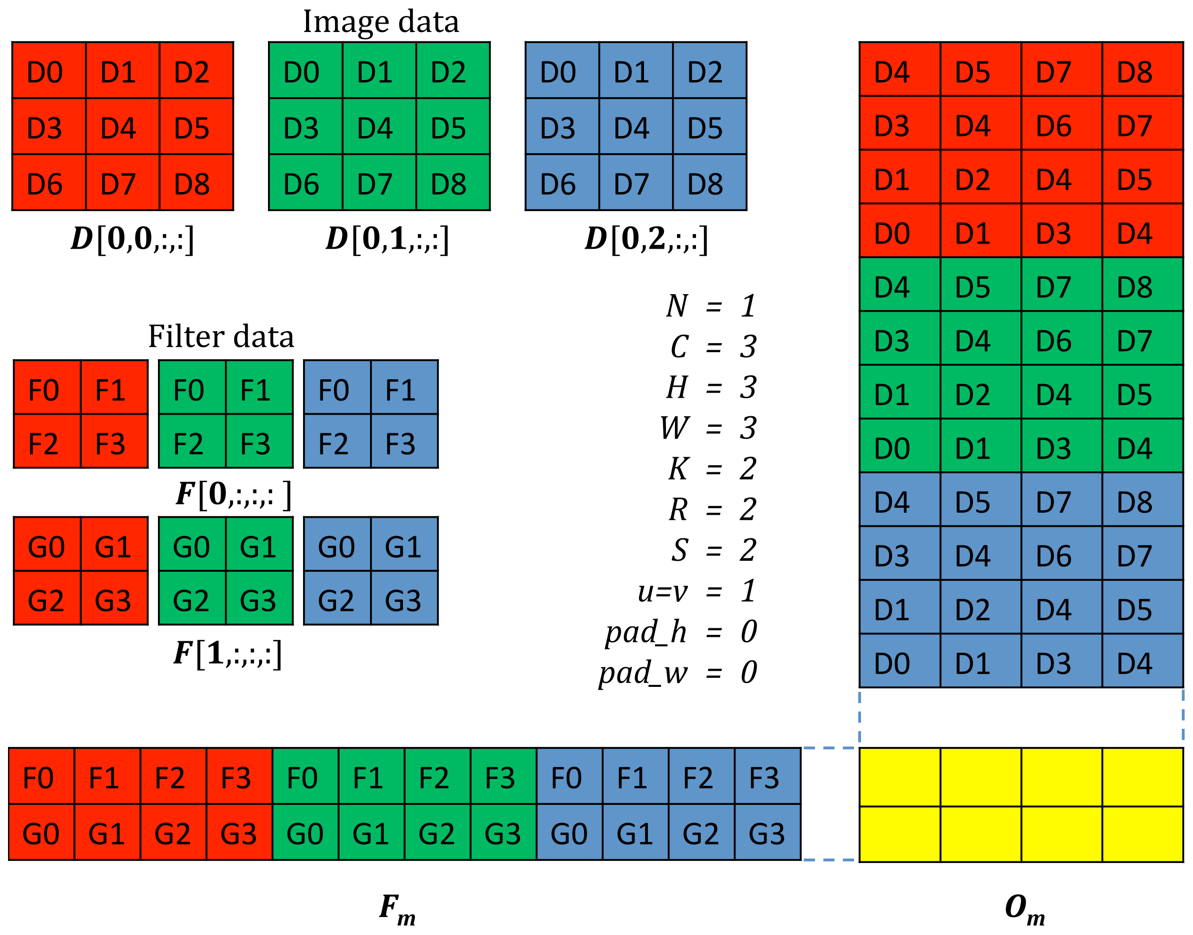




"Nvidia Deep Learning Performance Documentation"

Equivalent to Matrix Multiplication

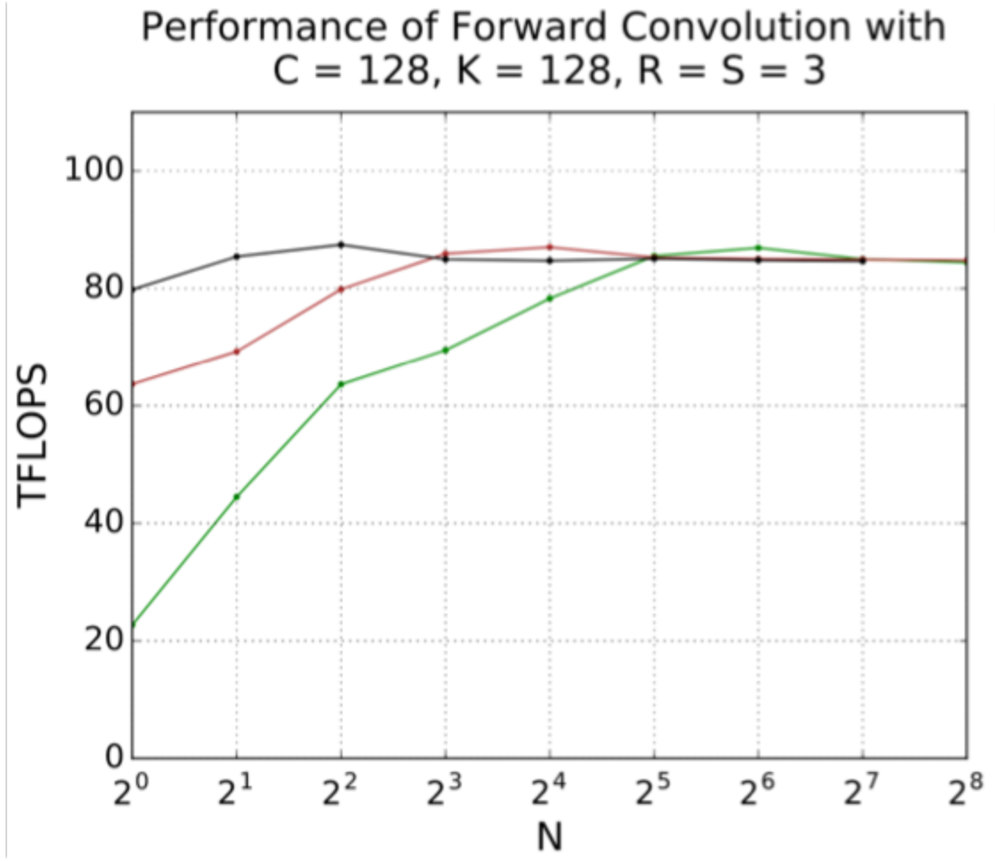
Virtual memory duplication

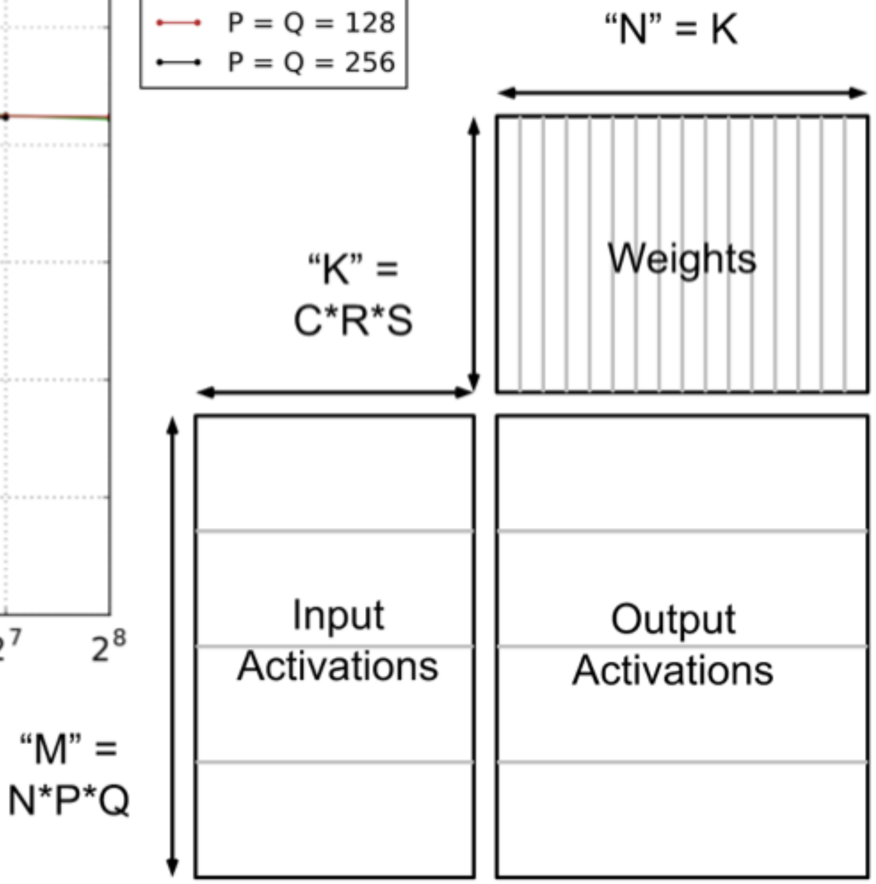


S. Chetlur et al., “cuDNN: Efficient Primitives for Deep Learning”

Convolutional Layer

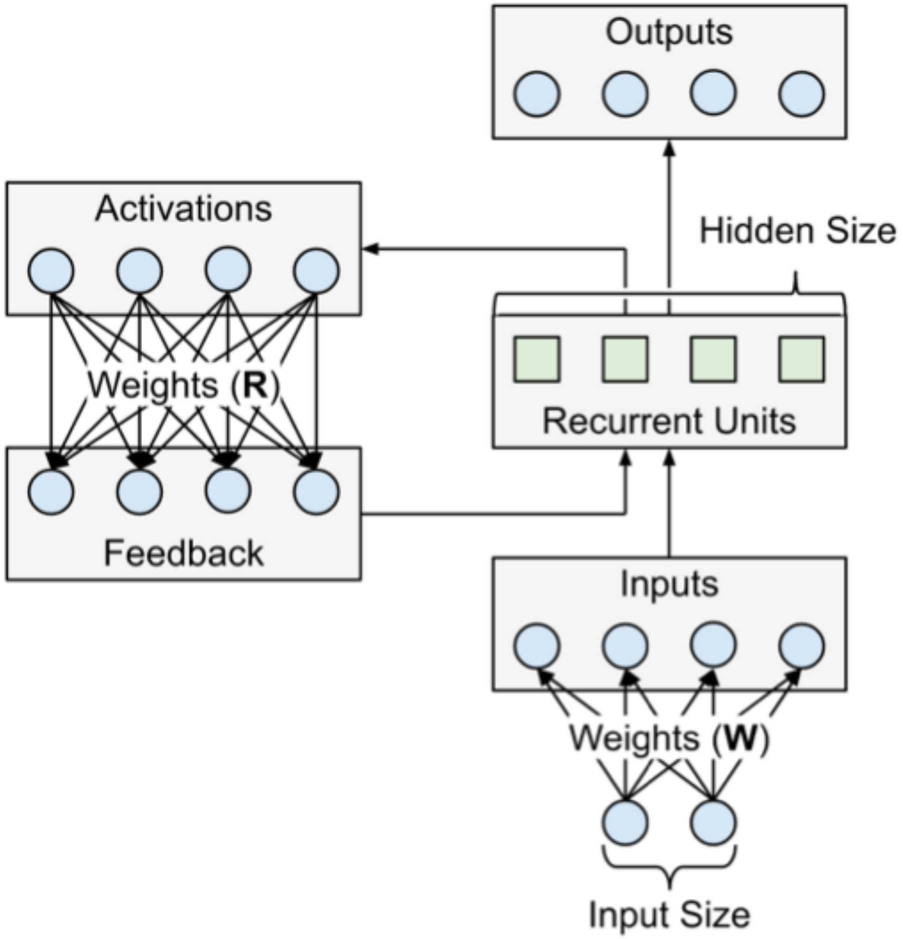
$$intensity = \frac{2\times(N\times K\times P\times Q)\times(C\times R\times S)}{2\times(N\times C\times H\times W + K\times C\times R\times S + N\times K\times P\times Q)}$$

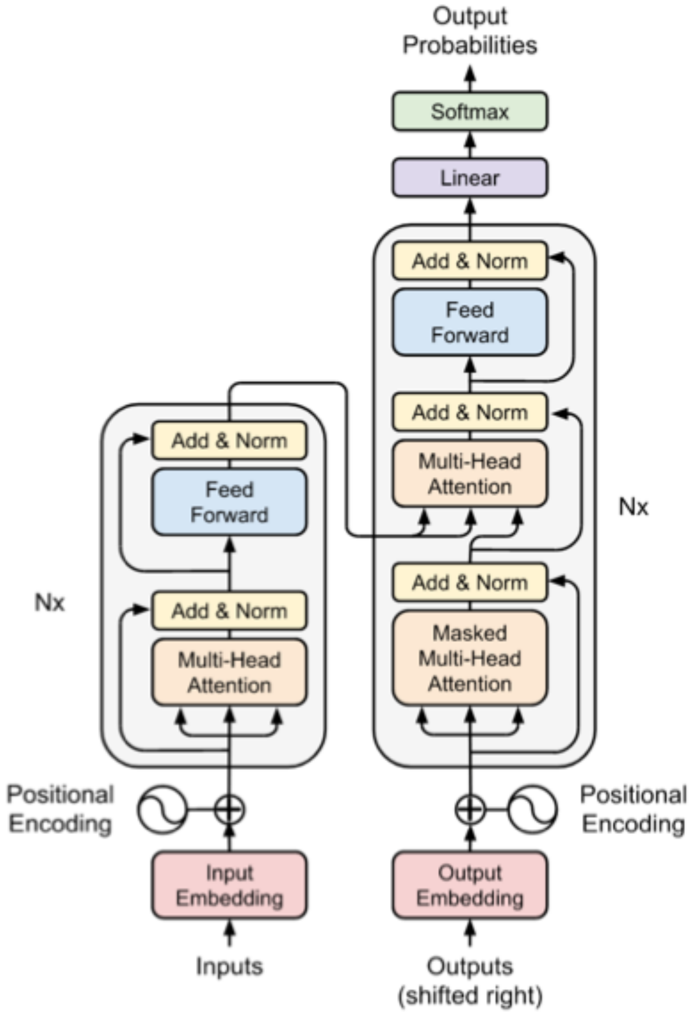




"Nvidia Deep Learning Performance Documentation"

Equivalents to Matrix Multiplication





Forget DNN, let's implement Matrix Multiplication!

"Nvidia Deep Learning Performance Documentation"

GPUS

SIMT Machines

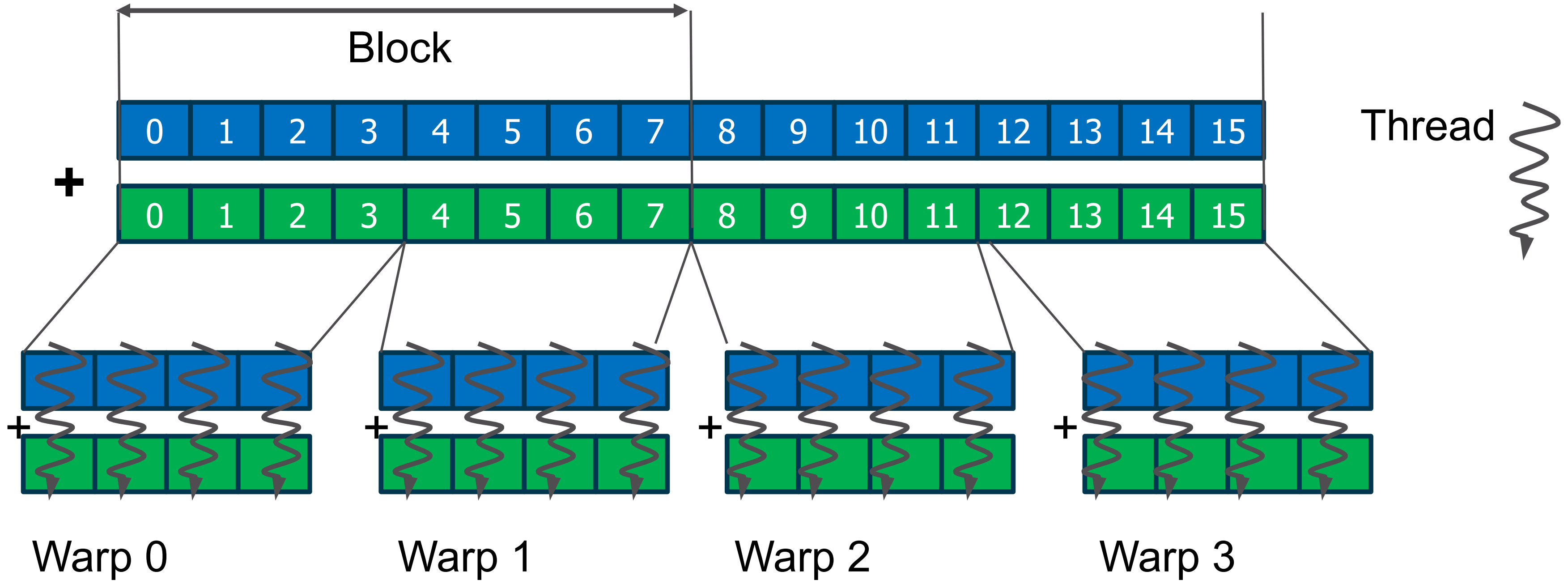
Single Instruction Multiple Threads (SIMT)

Scalar $C = A + B$

```
for(i = 0; i < N; i++) {
  C[i] = A[i] + B[i];
}
```

SIMT $C = A + B$

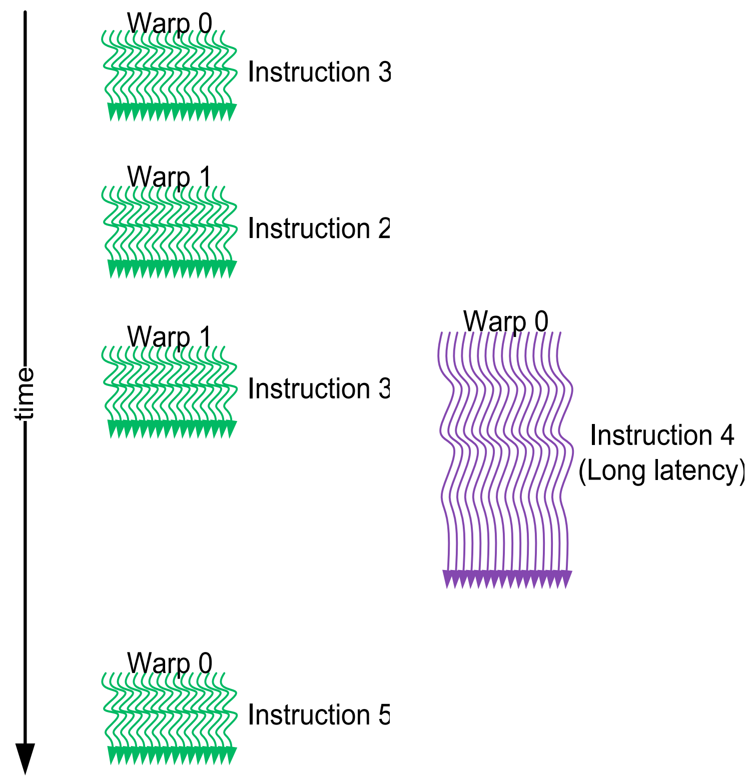
```
kernel(){
  tid = blkDim * blkId + thId
  C[tid] = A[tid] + B[tid]
}
```



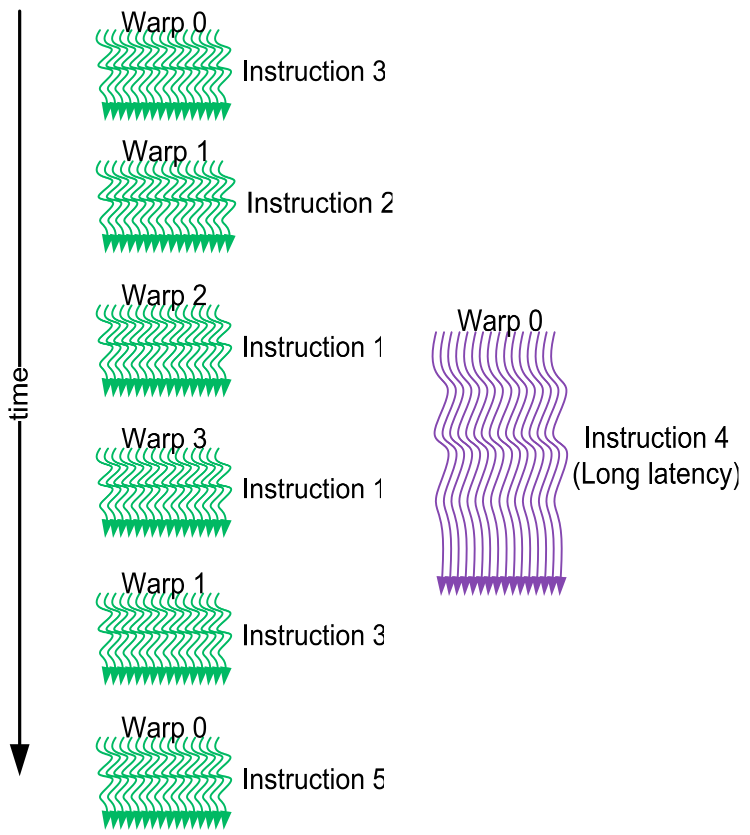
J. Gómez Luna and O. Mutlu, "Computer Architecture: GPU Programming," ETH Zurich, Fall 2020.

Latency hiding

2 active Warps



4 active Warps

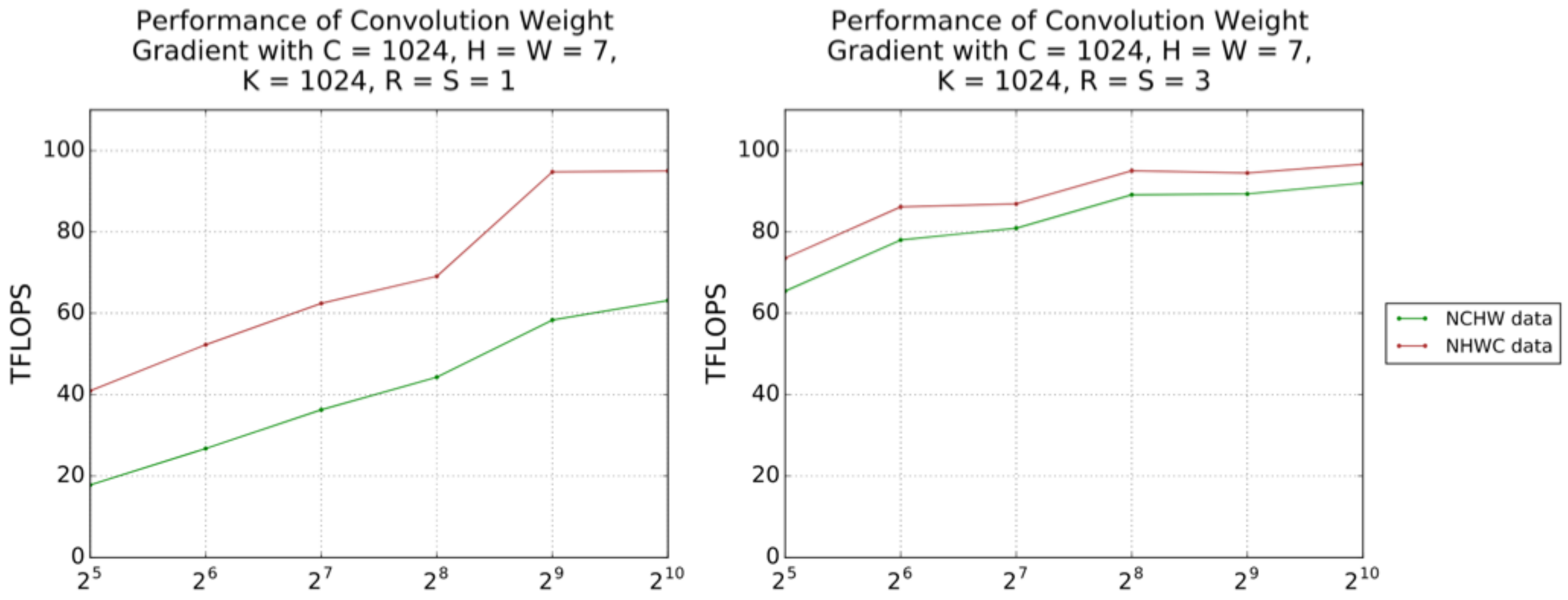


J. Gómez Luna and O. Mutlu, "Computer Architecture: GPU Programming," ETH Zurich, Fall 2020.

Data layout has an effect on performance

Shared memory is interleaved (banked)

Typically 32 banks on Nvidia GPUs



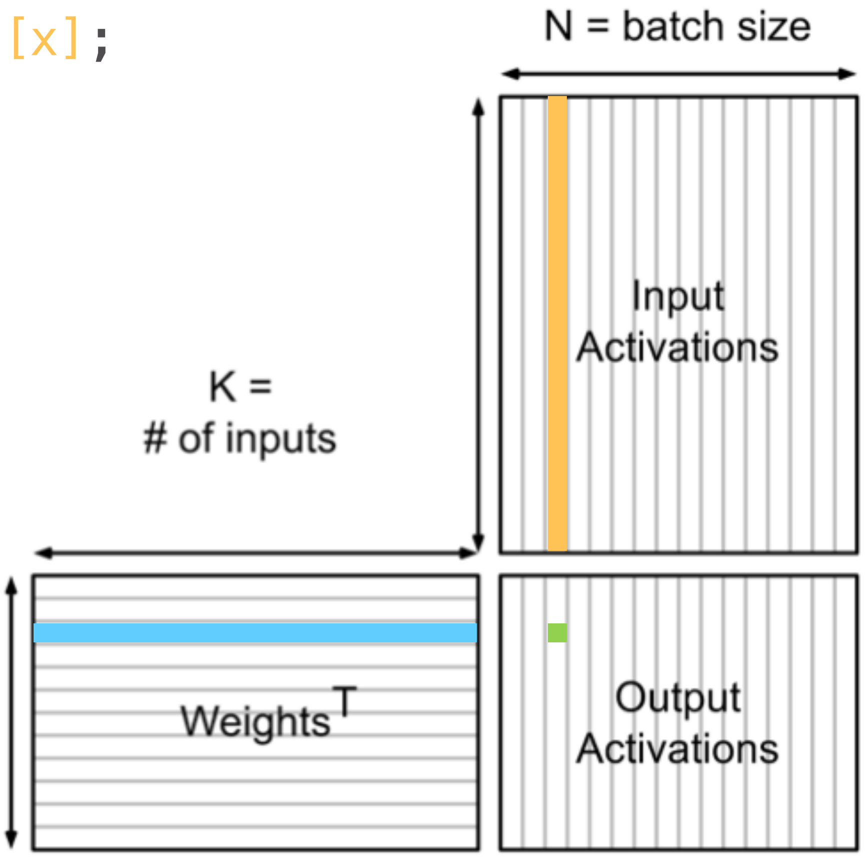
"Nvidia Deep Learning Performance Documentation"

Matrix Multiplication: SIMT way

Naïve implementation

```
kernel(){
  y = blkDim.y * blkId.y + thId.y;
  x = blkDim.x * blkId.x + thId.x;
  for (k = 0; k < K; k++) {
    C[y][x] += A[y][k] * B[k][x];
}}
```

$$intensity = \frac{2\times K}{2\times(K+K)} = \frac{1}{2}$$



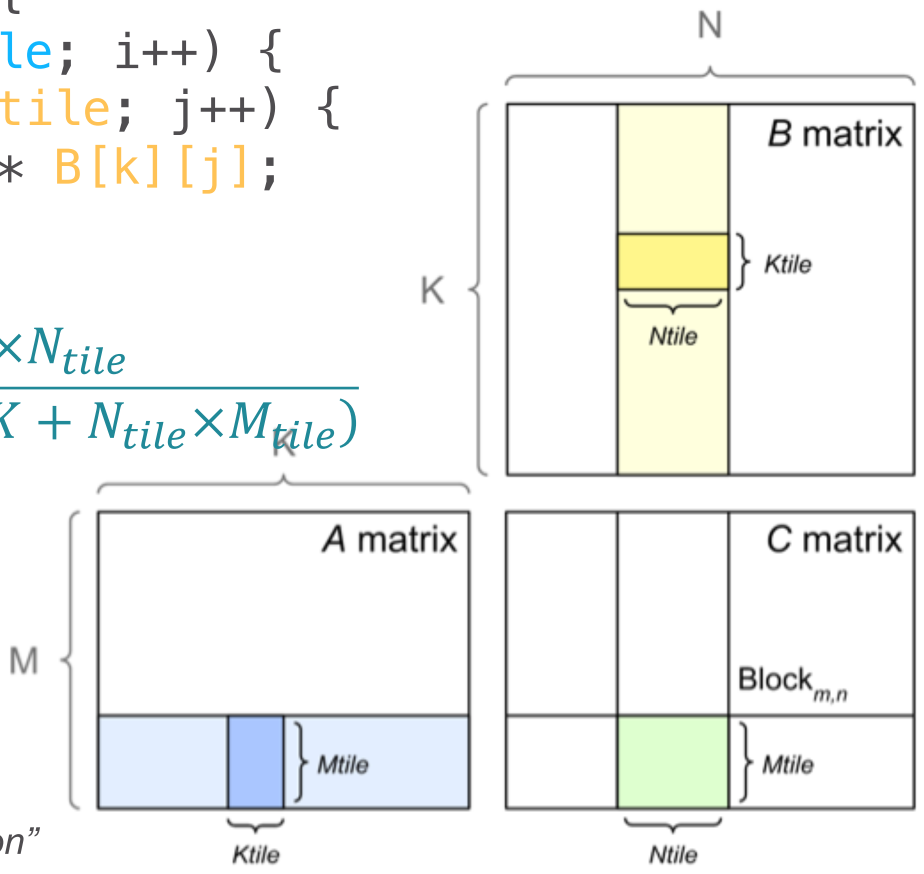
"Nvidia Deep Learning Performance Documentation"

Matrix Multiplication: SIMT way

Tiled implementation

```
kernel(){
  y = (blkDim.y * blkId.y + thId.y) * Mtile;
  x = (blkDim.x * blkId.x + thId.x) * Ntile;
  for (k = 0; k < K; k ++) {
    for (i = y; i < y + Mtile; i++) {
      for (j = x; j < x + Ntile; j++) {
        C[i][j] += A[i][k] * B[k][j];
}}}}
```

$$intensity = \frac{2\times K\times M_{tile}\times N_{tile}}{2\times(N_{tile}\times K + M_{tile}\times K + N_{tile}\times M_{tile})}$$



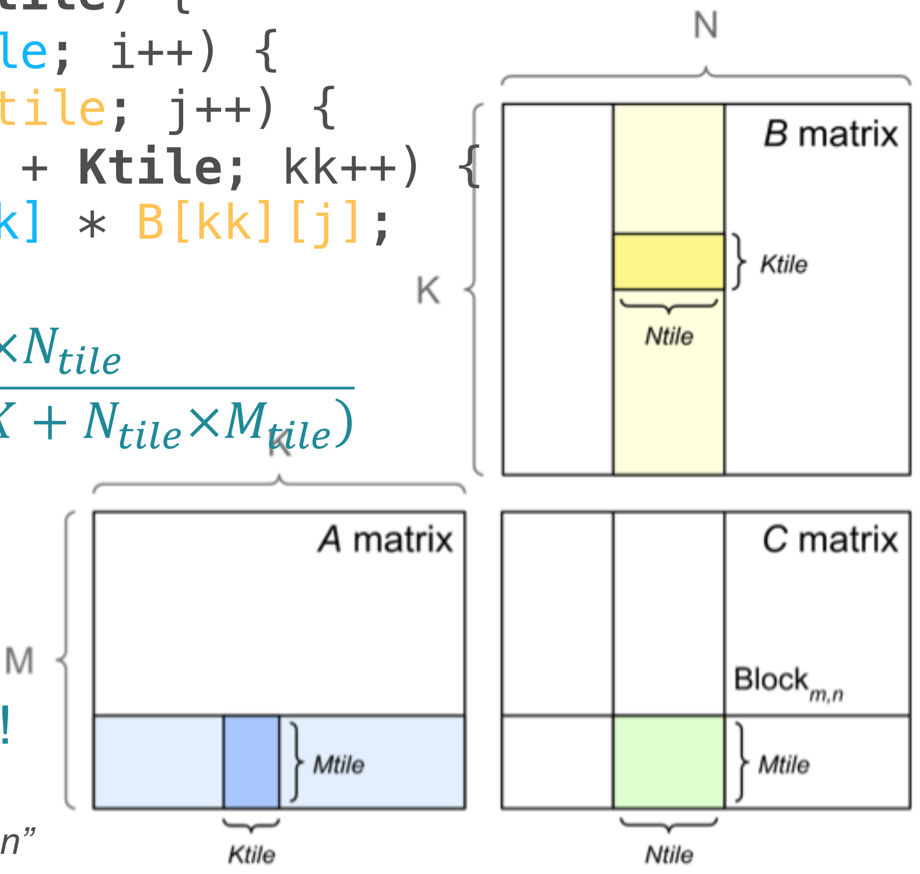
“Nvidia Deep Learning Performance Documentation”

Matrix Multiplication: SIMT way

Tiled² implementation

```
kernel(){
  y = (blkDim.y * blkId.y + thId.y) * Mtile;
  x = (blkDim.x * blkId.x + thId.x) * Ntile;
  for (k = 0; k < K; k += Ktile) {
    for (i = y; i < y + Mtile; i++) {
      for (j = x; j < x + Ntile; j++) {
        for (kk = k; kk < k + Ktile; kk++) {
          C[i][j] += A[i][kk] * B[kk][j];
}}}}}
```

$$intensity = \frac{2\times K\times M_{tile}\times N_{tile}}{2\times(N_{tile}\times K + M_{tile}\times K + N_{tile}\times M_{tile})}$$



Too complex?

- Use cuBLAS and cuDNN!
- Follow Nvidia optimization doc!

"Nvidia Deep Learning Performance Documentation"

Nvidia GPUs

Jetson Nano (Maxwell, 2014)

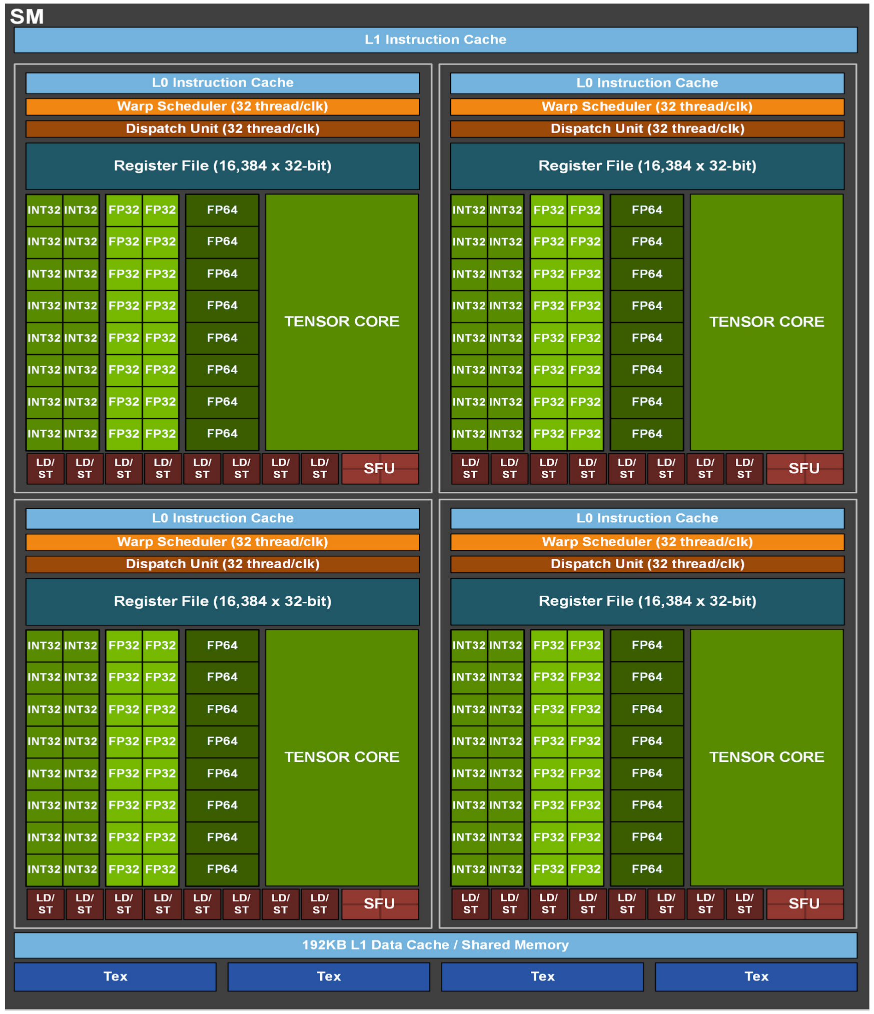
- 4 Streaming Multi-processors
- 32 CUDA cores / SM
- ~235 GFLOPS FP32
- 15 W

RTX 8000 (Turing, 2018)

- 72 Streaming Multi-processors
- 64 CUDA cores / SM
- 16,3 TFLOPS FP32
- 295 W

RTX A6000 (Ampere, 2020)

- 84 Streaming Multi-processors
- 128 CUDA cores / SM
- 38,7 TFLOPS FP32
- 300 W


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

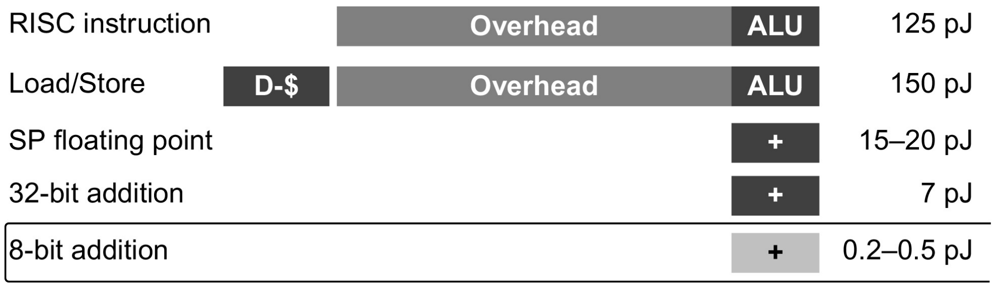
"NVIDIA A100 Tensor Core GPU Architecture"

SYSTOLIC ARRAYS

Domain-Specific Architectures

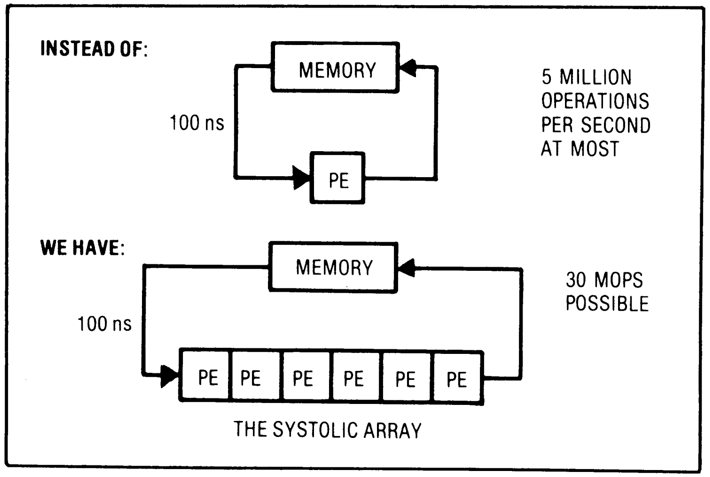
Guidelines for DSA

- *Use **dedicated memories** to minimize the distance over which data is moved.*
- *Invest the resources saved from dropping advanced microarchitectural optimizations into **more arithmetic units or bigger memories**.*
- *Use the **easiest form of parallelism** that matches the domain.*
- ***Reduce data size** and type to the simplest needed for the domain.*
- *Use a **domain-specific programming language** to port code to the DSA.*



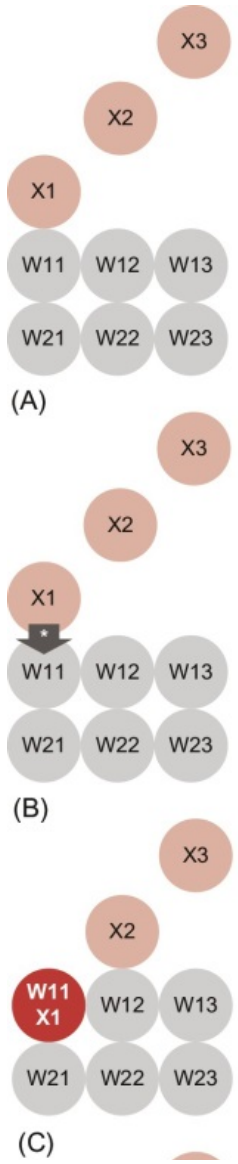
RISC instruction		Overhead	ALU	125 pJ
Load/Store	D-$	Overhead	ALU	150 pJ
SP floating point			+	15–20 pJ
32-bit addition			+	7 pJ
8-bit addition			+	0.2–0.5 pJ

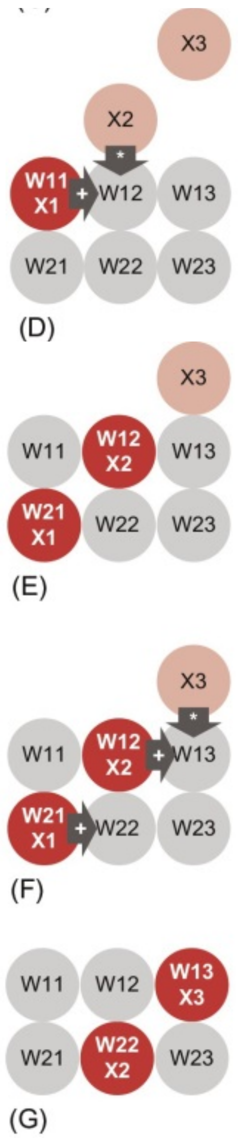
J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 2018.

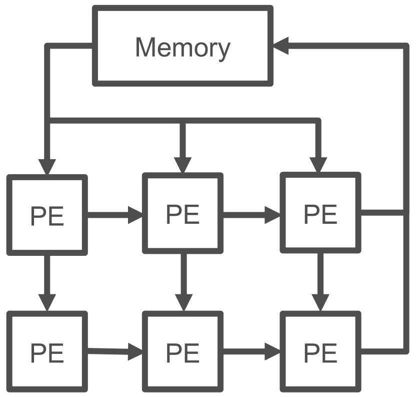


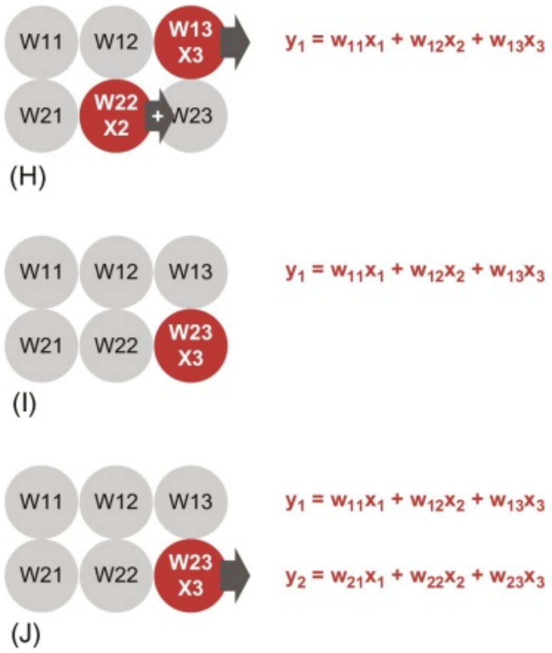
Kung, "Why systolic architectures?" Computer, vol. 15, pp. 37–46, 1982

Matrix Multiplication: systolic array way





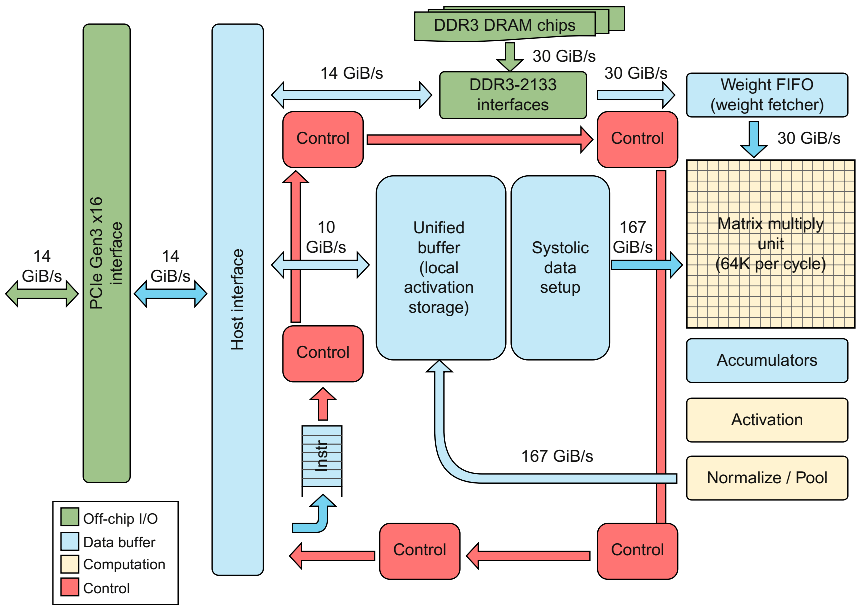




J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 2018.

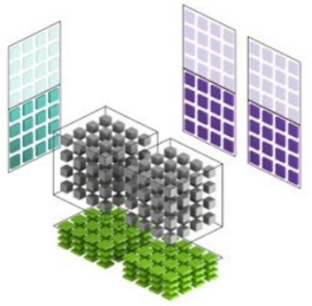
Tensor Processing Unit (TPU v3, 2018)

- 256 x 256 grid
- 90 TOPS INT8
- 75 W



RTX 8000 (Turing, 2018)

- 4 x 4 x 4 grid / tensor core
- 8 TC x 72 SM
- 130 TFLOPS FP16
- 295 W



RTX A6000 (Ampere, 2020)

- 4 x 4 x 4 grid / tensor core
- 4 TC x 84 SM
- 154 TFLOPS FP16
- 300 W

“NVIDIA A100 Tensor Core GPU Architecture”

J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 2018.

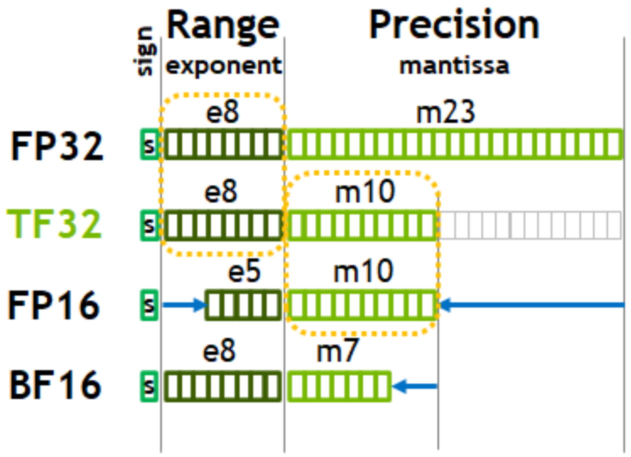
DNN OPTIMIZATIONS

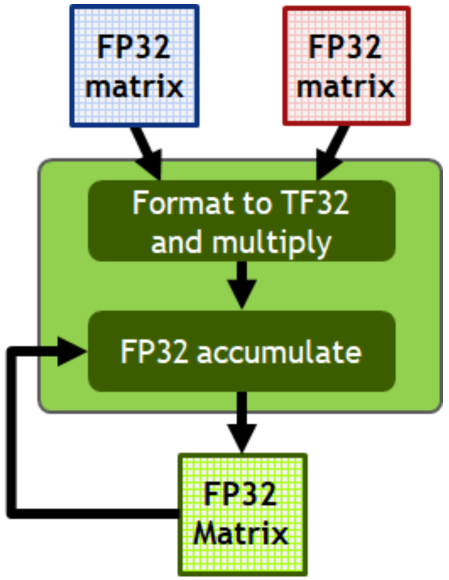
for hardware acceleration

Big impact on performance, but platform specific

RTX A6000 (Ampere, 2020)

- 77 TFLOPS TF32
- 154 TFLOPS BF16/FP16
- 309 TOPS INT8
- 619 TOPS INT4





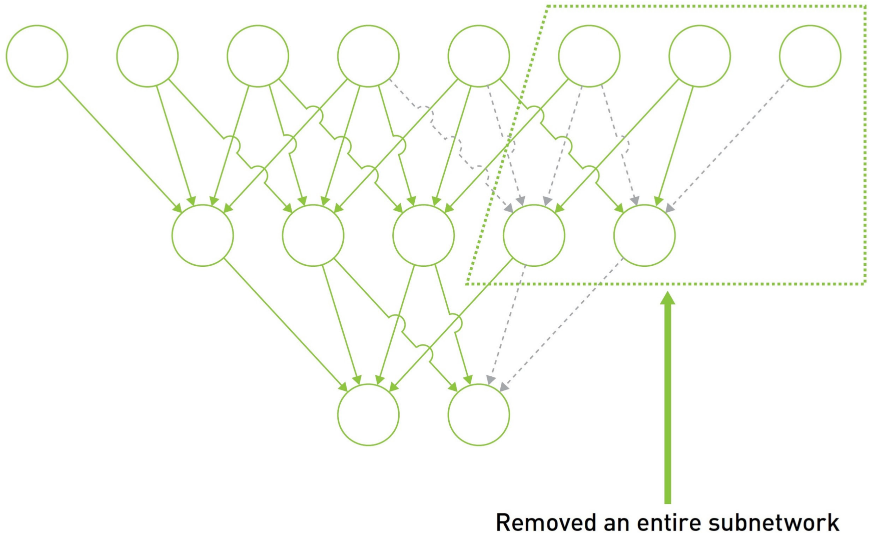
"NVIDIA A100 Tensor Core GPU Architecture"

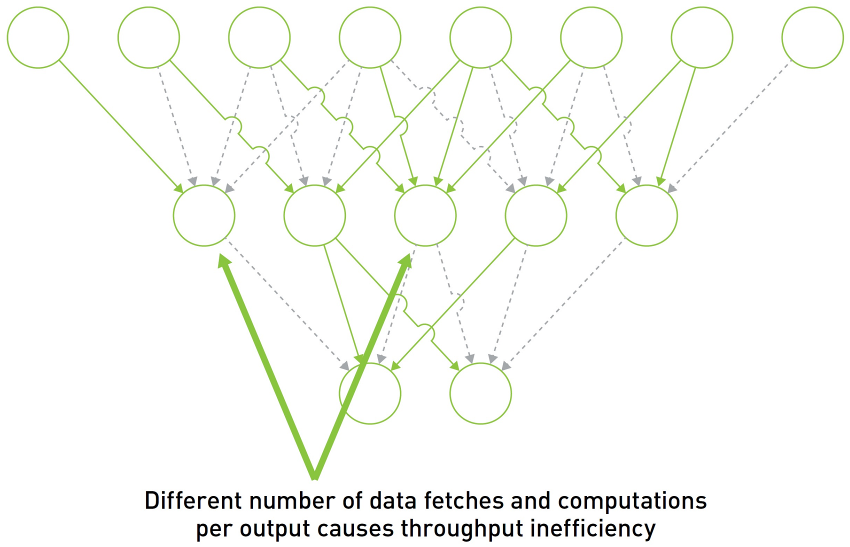
Coarse grain sparsity

Reduce workload and improve throughput

Fine grain sparsity

Irregular memory access reduce throughput

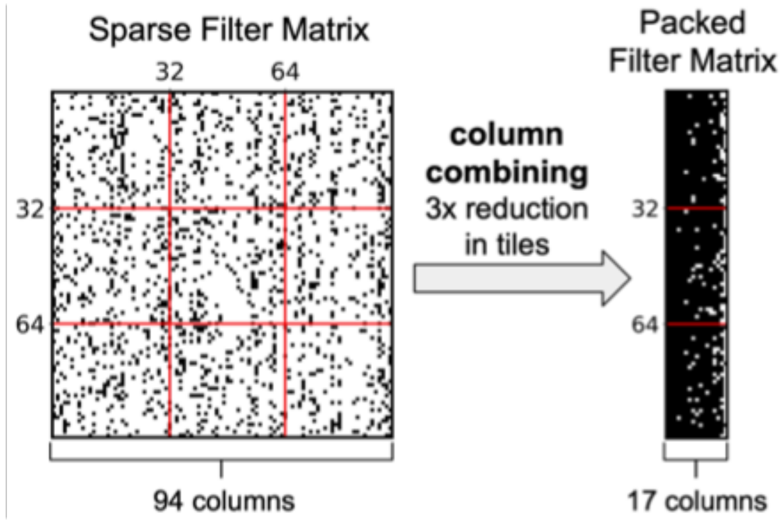



Different number of data fetches and computations
per output causes throughput inefficiency

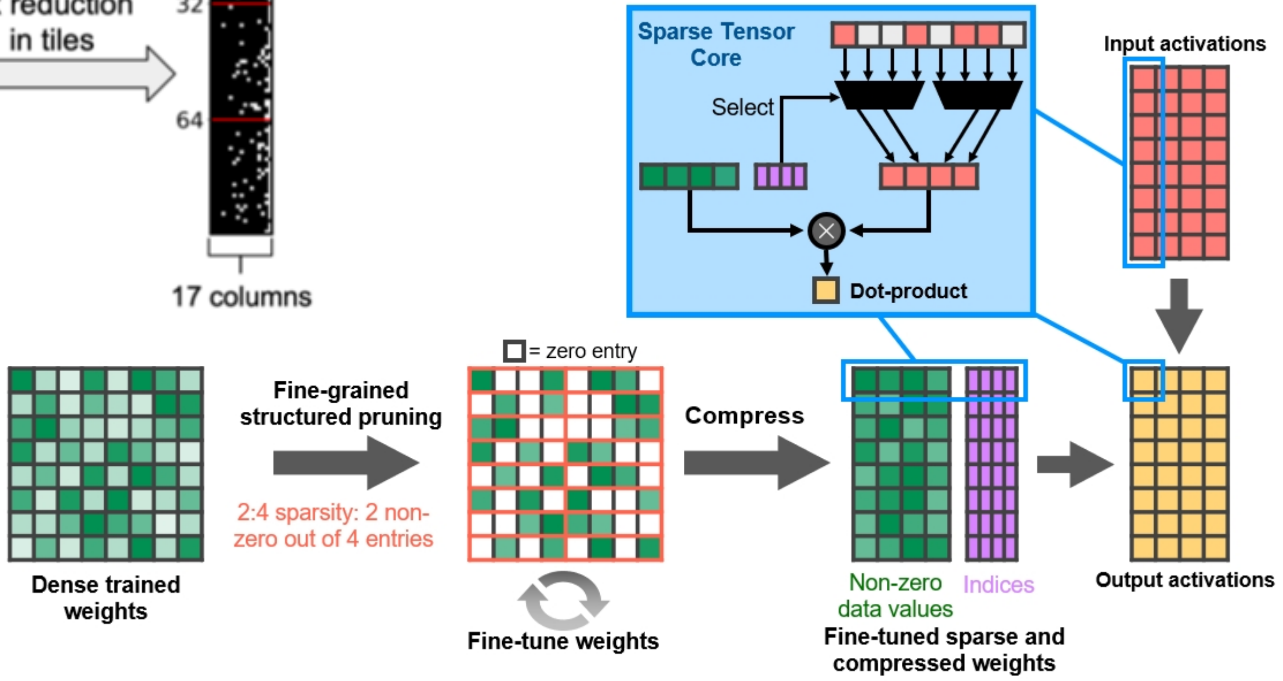
“NVIDIA A100 Tensor Core GPU Architecture”

Sparse matrix packing

Pack weights in smaller matrix



- Increase density and throughput
- Requires hardware support



“NVIDIA A100 Tensor Core GPU Architecture”

H. T. Kung, et al., “Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization,” ASPLOS 2019.

Bibliography

Nvidia documentation

- Nvidia, "Nvidia Deep Learning Performance Documentation," Jul. 2020.
- Nvidia, "NVIDIA A100 Tensor Core GPU Architecture," 2020.
- S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," 2014.

Bibliography

- J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Sixth Edition. Elsevier, 2018.
- J. Gómez Luna and O. Mutlu, "Computer Architecture: GPU Programming," ETH Zurich, Fall 2020.
- H. T. Kung, B. McDanel, and S. Q. Zhang, "Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization," *ASPLOS*, pp. 821–834, 2019.
- N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," *Commun. ACM*, Aug. 2018.
- Kung, "Why systolic architectures?," *Computer*, vol. 15, Jan. 1982.