Lecture 10:

Hardware Specialization

Parallel Computing
Stanford CS149, Fall 2025

Energy-constrained computing

Energy (Power x Time)-constrained computing

Mobile devices are energy constrained

- Limited battery life
- Heat dissipation without fan

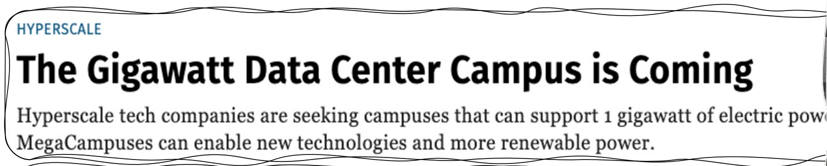
Supercomputers and data centers are energy constrained

- Due to shear scale of machine (100,000s of CPUs and GPUs)
- Power for datacenter
- Cooling for the data center

AI is Constrained by Energy

AI demands are growing exponentially

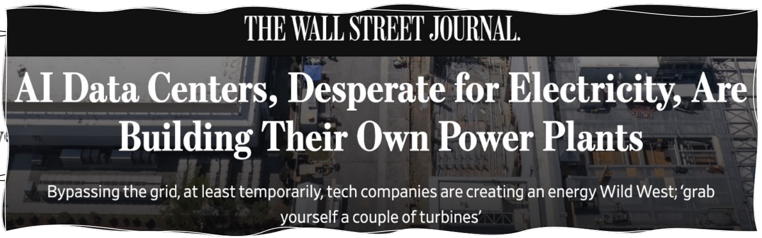
Data centers are heavily energy constrained



HYPERSCALE

The Gigawatt Data Center Campus is Coming

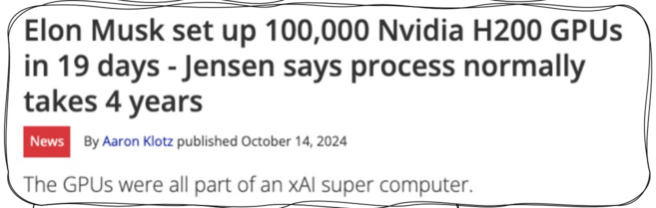
Hyperscale tech companies are seeking campuses that can support 1 gigawatt of electric powe
MegaCampuses can enable new technologies and more renewable power.



THE WALL STREET JOURNAL.

AI Data Centers, Desperate for Electricity, Are Building Their Own Power Plants

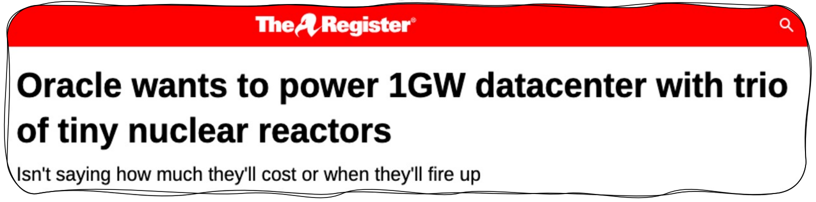
Bypassing the grid, at least temporarily, tech companies are creating an energy Wild West; 'grab yourself a couple of turbines'



Elon Musk set up 100,000 Nvidia H200 GPUs in 19 days - Jensen says process normally takes 4 years

News By Aaron Klotz published October 14, 2024

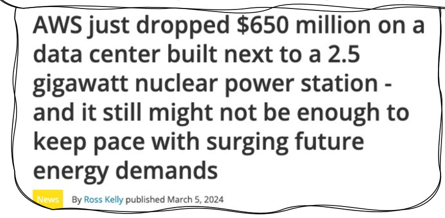
The GPUs were all part of an xAI super computer.



The Register

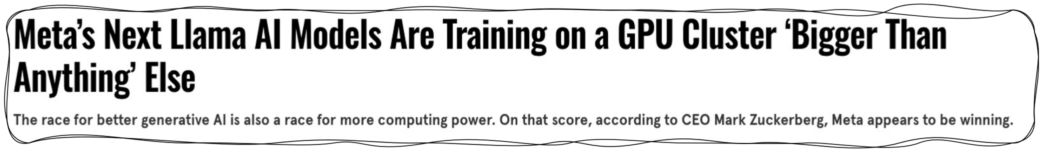
Oracle wants to power 1GW datacenter with trio of tiny nuclear reactors

Isn't saying how much they'll cost or when they'll fire up



AWS just dropped $650 million on a data center built next to a 2.5 gigawatt nuclear power station - and it still might not be enough to keep pace with surging future energy demands

News By Ross Kelly published March 5, 2024



Meta's Next Llama AI Models Are Training on a GPU Cluster 'Bigger Than Anything' Else

The race for better generative AI is also a race for more computing power. On that score, according to CEO Mark Zuckerberg, Meta appears to be winning.

Performance and Power

$$\underset{\textbf{FIXED}}{Power} = \underset{\uparrow}{\overset{\text{Performance}}{\frac{Ops}{second}}} \times \underset{\downarrow}{\overset{\text{Energy efficiency}}{\frac{Joules}{Op}}}$$

What is the magnitude of improvement from specialization?

Better energy efficiency ⇒ Specialization (fixed function)

Pursuing highly efficient processing…
(specializing hardware beyond just parallel CPUs and GPUs)

Why is a “general-purpose processor” so inefficient?

Wait… this entire class we’ve been talking about making efficient use out of multi-core CPUs and GPUs…
and now you’re telling me these platforms are “inefficient”?

Consider the complexity of executing an instruction on a modern processor…

Read instruction ⟶ Address translation, communicate with icache, access icache, etc.

Decode instruction ⟶ Translate op to uops, access uop cache, etc.

Check for dependencies/pipeline hazards

Identify available execution resource

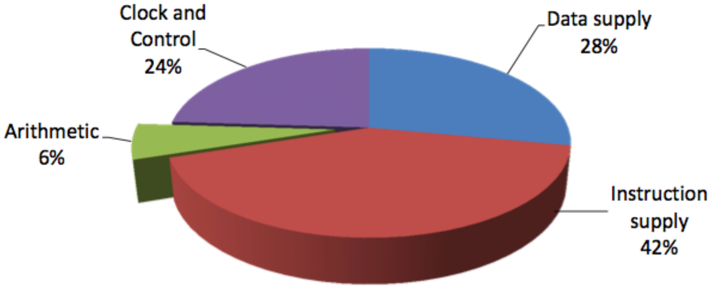
Use decoded operands to control register file SRAM (retrieve data)

Move data from register file to selected execution resource

Perform arithmetic operation

Move data from execution resource to register file

Use decoded operands to control write to register file SRAM



Efficient Embedded Computing [Dally et al. 08]

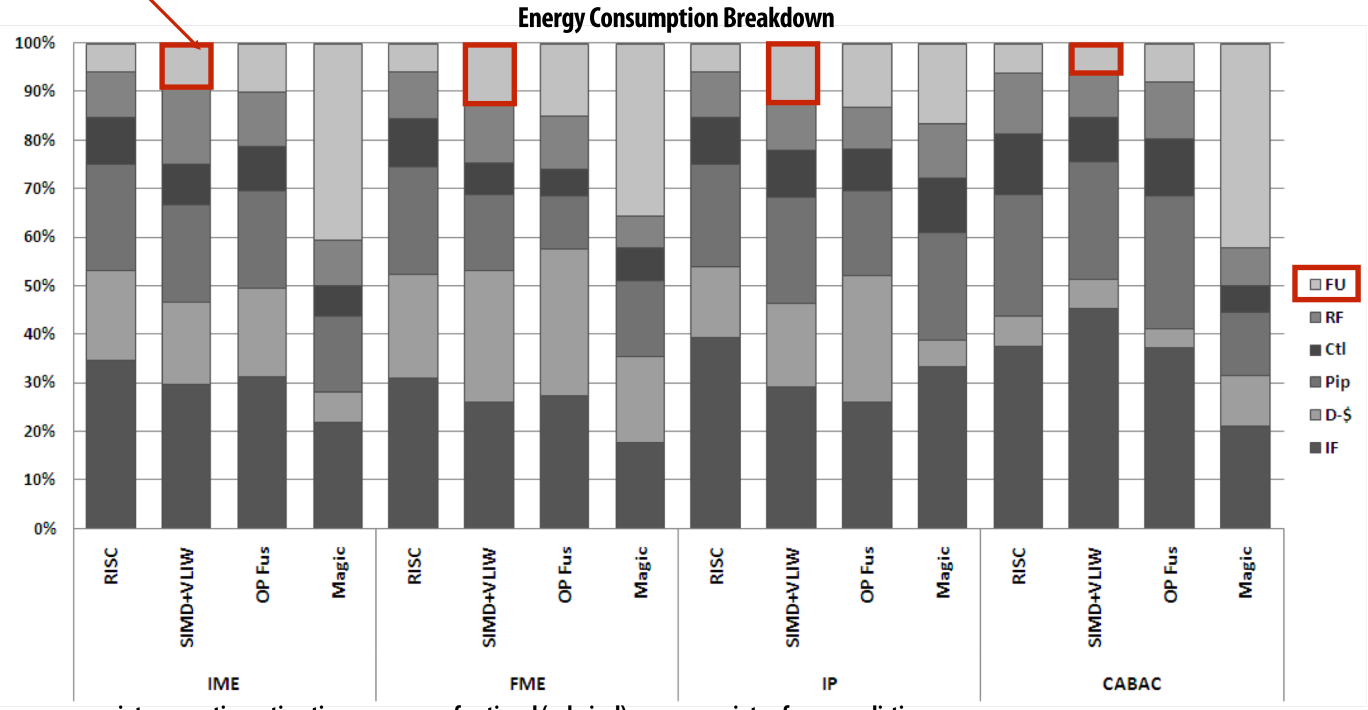
[Figure credit Eric Chung]

Review question:
How does SIMD execution reduce overhead of certain types of computations?
What properties must these computations have?

H.264 video encoding: fraction of energy consumed by functional units is small (even when using SIMD)

Even after encoding implemented with SIMD instruction

[Hameed et al. ISCA 2010]



integer motion estimation

fractional (subpixel) motion estimation

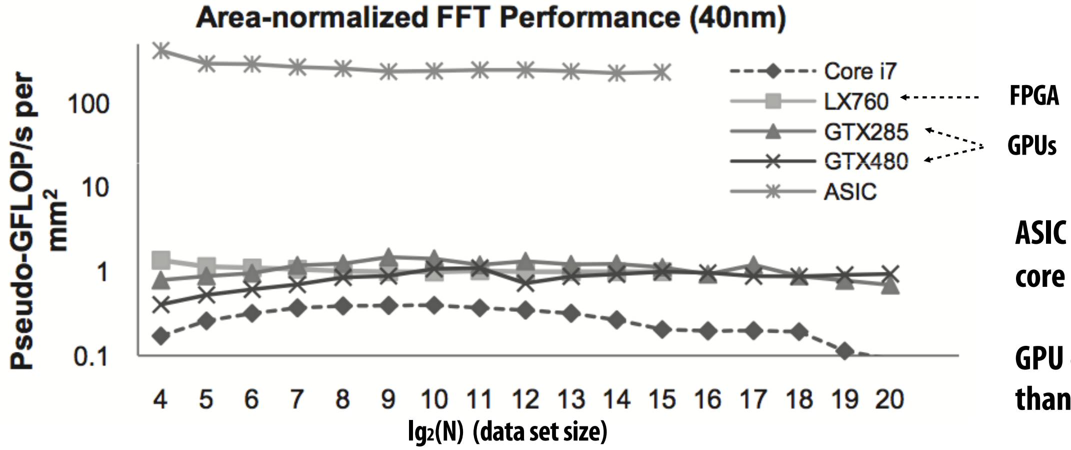
intra-frame prediction, DTC, quantization

arithmetic encoding

FU = functional units
RF = register fetch
Ctrl = misc pipeline control

Pip = pipeline registers (interstage)
D-$ = data cache
IF = instruction fetch + instruction cache

Fast Fourier transform (FFT): throughput and energy benefits of specialization

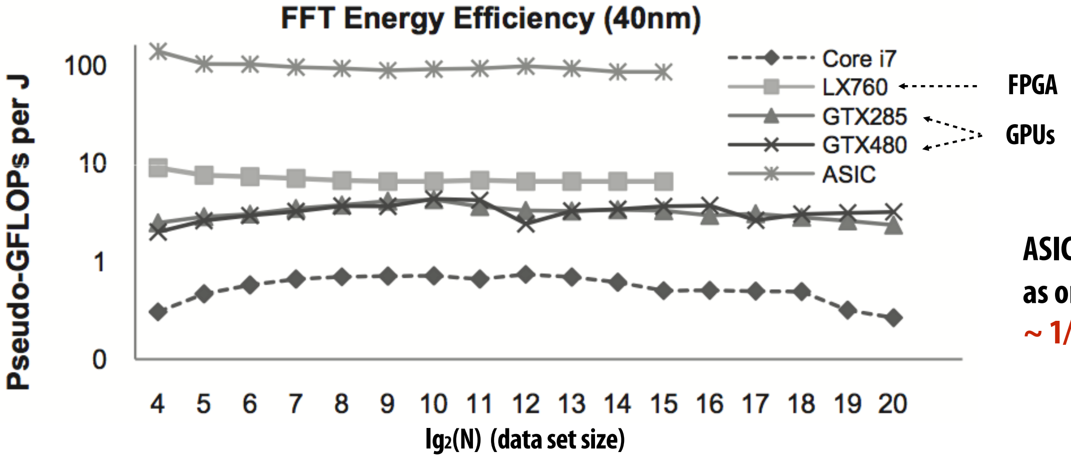


FPGA

GPUs

ASIC delivers same performance as one CPU core with **~ 1/1000th the chip area**.

GPU cores: ~ 5-7 times more area efficient than CPU cores.



FPGA

GPUs

ASIC delivers same performance as one CPU core using only **~ 1/100th the power**

[Chung et al. MICRO 2010]

Digital signal processors (DSPs)

Programmable processors, but simpler instruction stream control paths

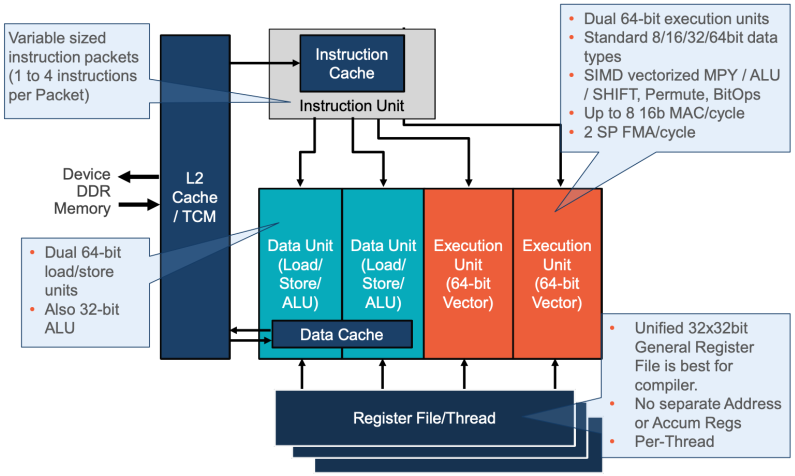
Complex instructions (e.g., SIMD/VLIW): perform many operations per instruction (amortize cost of control)

Example: Qualcomm Hexagon DSP

Used for modem, audio, and (increasingly) image processing on Qualcomm Snapdragon SoC processors

VLIW: "very-long instruction word"
Single instruction specifies multiple different operations to do at once (contrast to SIMD)

Below: innermost loop of FFT
Hexagon DSP performs 29 "RISC" ops per cycle



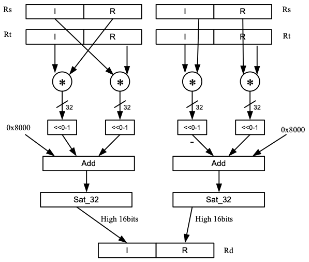
Variable sized instruction packets (1 to 4 instructions per Packet)
Instruction Cache
Instruction Unit
L2 Cache / TCM
Device DDR Memory
Dual 64-bit execution units
Standard 8/16/32/64bit data types
SIMD vectorized MPY / ALU / SHIFT, Permute, BitOps
Up to 8 16b MAC/cycle
2 SP FMA/cycle
Data Unit (Load/ Store/ ALU)
Data Unit (Load/ Store/ ALU)
Execution Unit (64-bit Vector)
Execution Unit (64-bit Vector)
Data Cache
Dual 64-bit load/store units
Also 32-bit ALU
Register File/Thread
Unified 32x32bit General Register File is best for compiler.
No separate Address or Accum Regs
Per-Thread

64-bit Load and

64-bit Store with post-update addressing

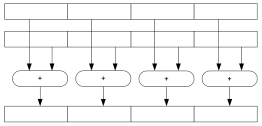
```
{ R17:16 = MEMD(R0++M1)
  MEMD(R6++M1) = R25:24
  R20 = CMPY(R20, R8):<<1:rnd:sat
  R11:10 = VADDH(R11:10, R13:12)
}:endloop0
```

Complex multiply with round and saturation



Zero-overhead loops
- Dec count
- Compare
- Jump top

Vector 4x16-bit Add



Hexagon DSP is in Google Pixel phone

Anton supercomputer for molecular dynamics

[Developed by DE Shaw Research]

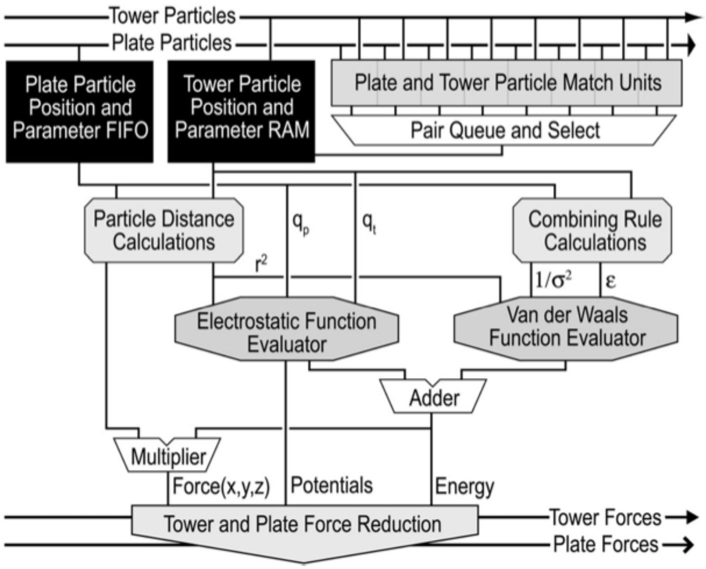
Anton 1 (2008) simulates time evolution of proteins

ASIC for computing particle-particle interactions (512 of them in machine)

Throughput-oriented subsystem for efficient fast-fourier transforms

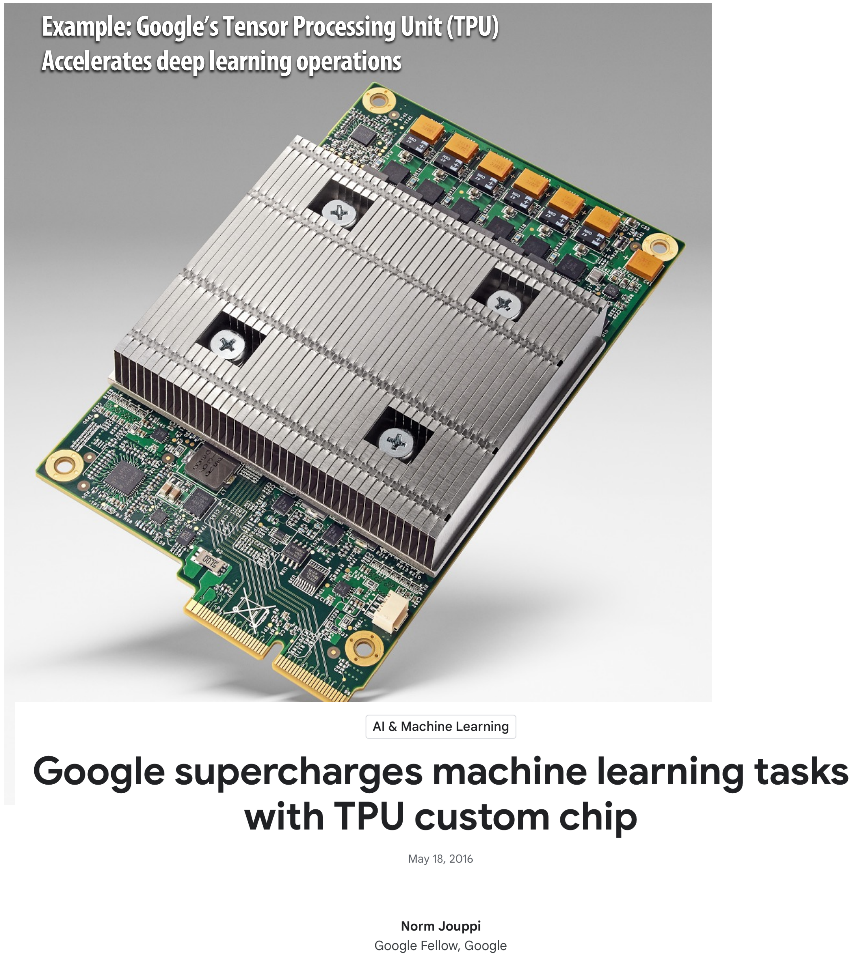
Custom, low-latency communication

network designed for communication patterns of N-body simulations


Tower Particles
Plate Particles
Plate Particle Position and Parameter FIFO
Tower Particle Position and Parameter RAM
Plate and Tower Particle Match Units
Pair Queue and Select
Particle Distance Calculations
q_p
q_t
Combining Rule Calculations
r^2
$1/\sigma^2$
ε
Electrostatic Function Evaluator
Van der Waals Function Evaluator
Adder
Multiplier
Force(x,y,z)
Potentials
Energy
Tower and Plate Force Reduction
Tower Forces
Plate Forces

Anton 3 (2025) is approximately 20 times faster than a contemporary GPU

Specialized processors for evaluating deep networks



Example: Google's Tensor Processing Unit (TPU)
Accelerates deep learning operations

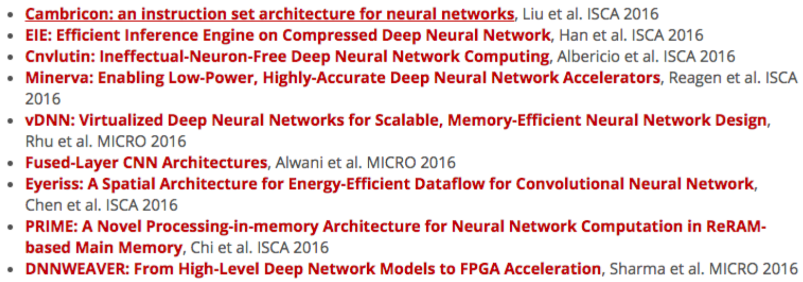
AI & Machine Learning

Google supercharges machine learning tasks with TPU custom chip

May 18, 2016

Norm Jouppi
Google Fellow, Google

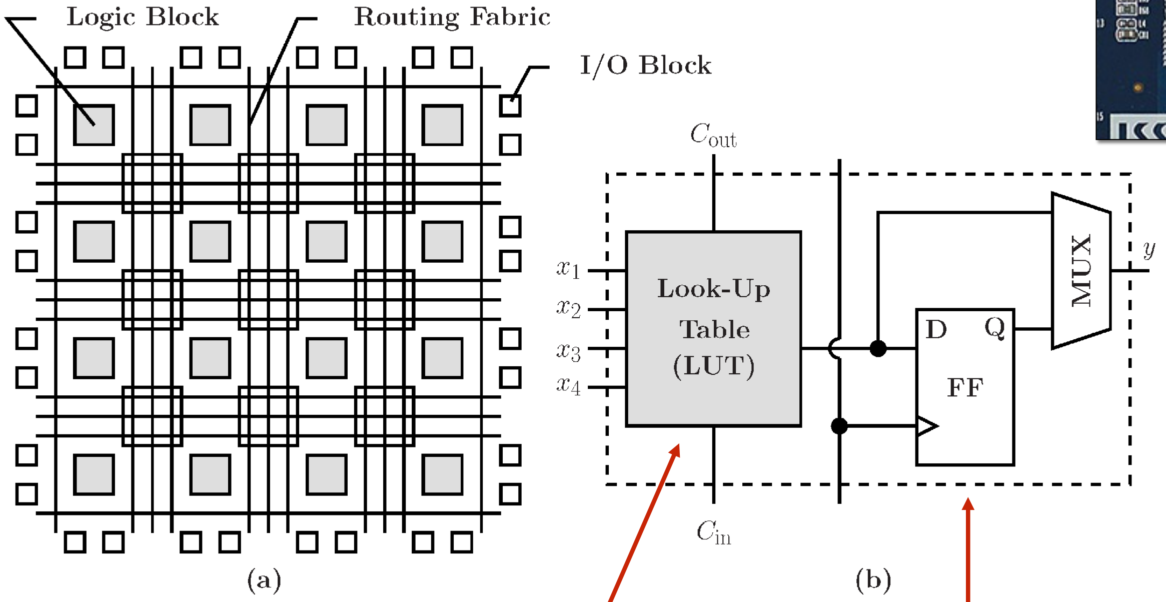
Countless papers followed at top computer architecture research conferences on the topic of ASICs or accelerators for deep learning or evaluating deep networks…



- **Cambricon: an instruction set architecture for neural networks**, Liu et al. ISCA 2016
- **EIE: Efficient Inference Engine on Compressed Deep Neural Network**, Han et al. ISCA 2016
- **Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing**, Albericio et al. ISCA 2016
- **Minerva: Enabling Low-Power, Highly-Accurate Deep Neural Network Accelerators**, Reagen et al. ISCA 2016
- **vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design**, Rhu et al. MICRO 2016
- **Fused-Layer CNN Architectures**, Alwani et al. MICRO 2016
- **Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Network**, Chen et al. ISCA 2016
- **PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory**, Chi et al. ISCA 2016
- **DNNWEAVER: From High-Level Deep Network Models to FPGA Acceleration**, Sharma et al. MICRO 2016

FPGAs (Field Programmable Gate Arrays)

- Middle ground between an ASIC and a processor
- FPGA chip provides array of logic blocks, connected by interconnect
- Programmer-defined logic implemented directly by FGPA



Flip flop (a register)

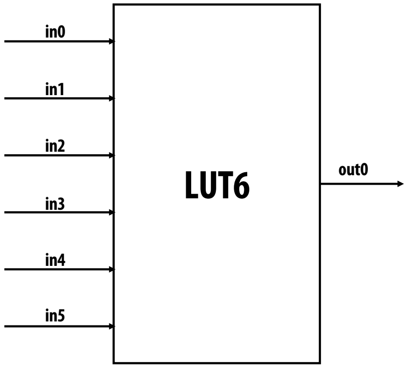
Programmable lookup table (LUT)

Image credit: Bai et al. 2014

Specifying combinational logic as a LUT

Example: 6-input, 1 output LUT in Xilinx Virtex-7 FPGAs

- Think of a LUT6 as a 64 element table



Example:
6-input AND

In	Out
0	0
1	0
2	0
3	0
⋮	⋮
63	1

40-input AND constructed by chaining outputs of eight LUT6's (delay = 3)

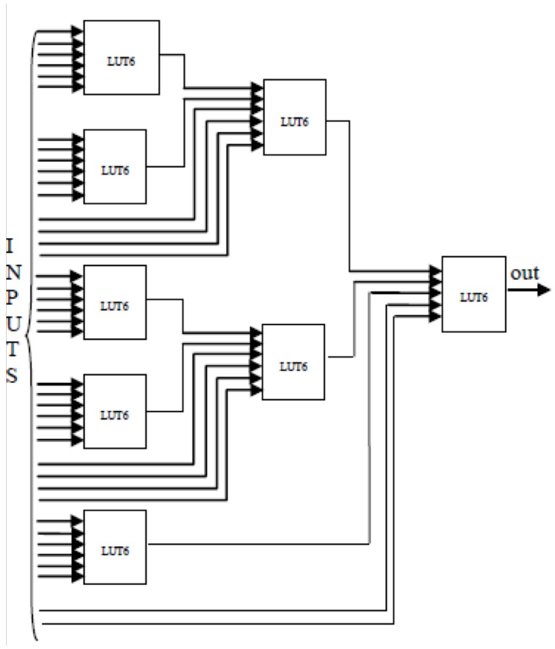
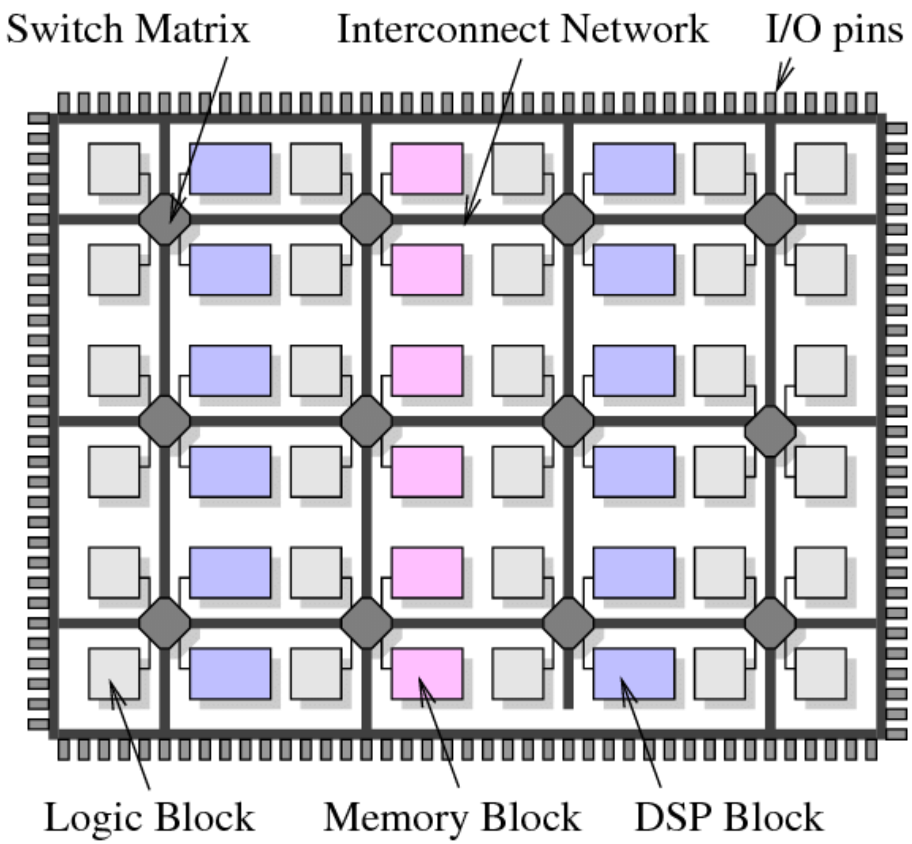


Image credit: [Zia 2013]

Modern FPGAs



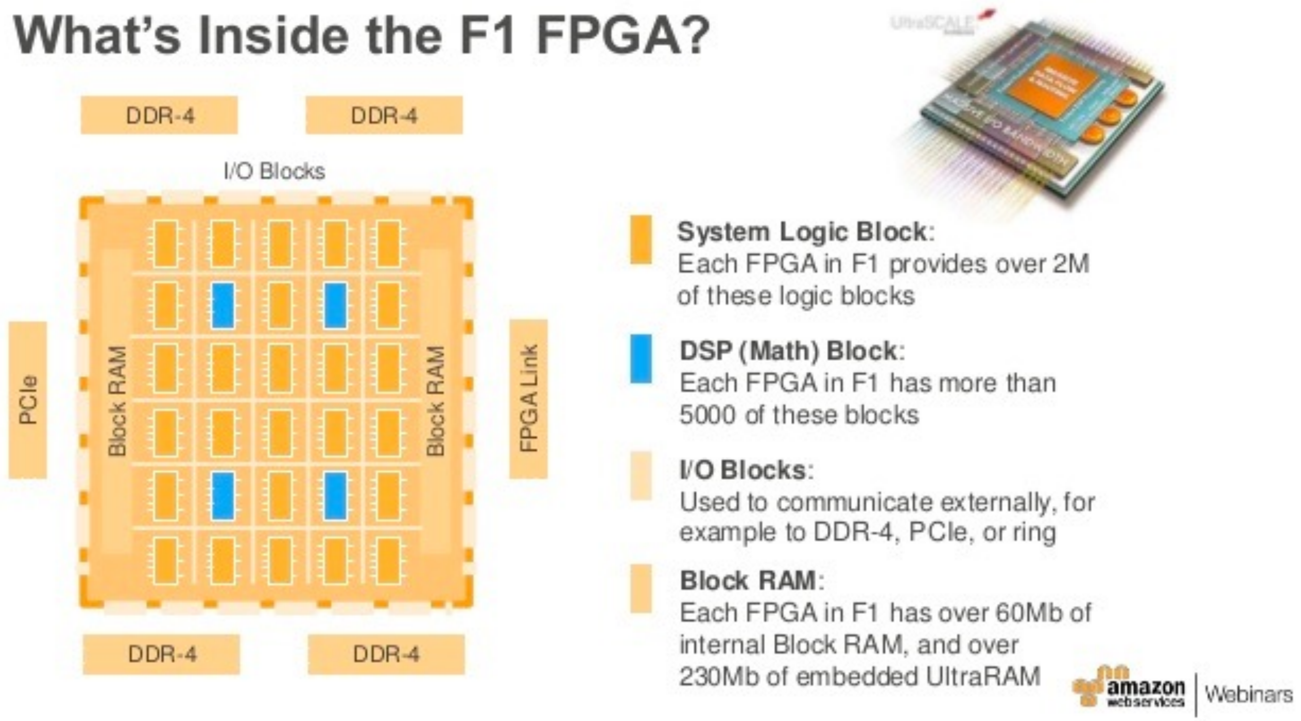
A lot of area devoted to hard gates

- Memory blocks (SRAM)
- DSP blocks (multiplier)
- CPUs (ARM, RISC-V)

Program with a hardware description language (e.g. Verilog, EE108)

Amazon EC2 F1/F2

FPGA's are now available on Amazon cloud services



What's Inside the F1 FPGA?

🟧 **System Logic Block:** Each FPGA in F1 provides over 2M of these logic blocks

🟦 **DSP (Math) Block:** Each FPGA in F1 has more than 5000 of these blocks

▯ **I/O Blocks:** Used to communicate externally, for example to DDR-4, PCIe, or ring

🟧 **Block RAM:** Each FPGA in F1 has over 60Mb of internal Block RAM, and over 230Mb of embedded UltraRAM

Efficiency benefits of compute specialization

Rules of thumb: compared to high-quality C code on CPU...

Throughput-maximized processor architectures: e.g., GPU cores

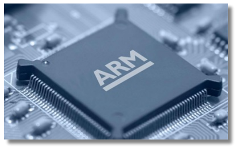
- Approximately 10x improvement in perf / watt
- Assuming code maps well to wide data-parallel execution and is compute bound

Fixed-function ASIC ("application-specific integrated circuit")

- Can approach 100-1000x or greater improvement in perf/watt
- Assuming code is compute bound and is not floating-point math

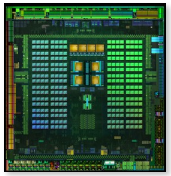
Efficiency vs. Programability

Energy-optimized CPU



Easiest to program

Throughput-oriented processor (GPU)

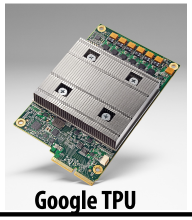


~10X more efficient

Programmable DSP

HEXAGON™

Domain Specific Accelerator



~20X

Limited domain of programmability with DSLs (e.g. DNN)

FPGA/ reconfigurable logic

~50X???
(jury still out)

Difficult to program (making it easier is active area of research)

ASIC

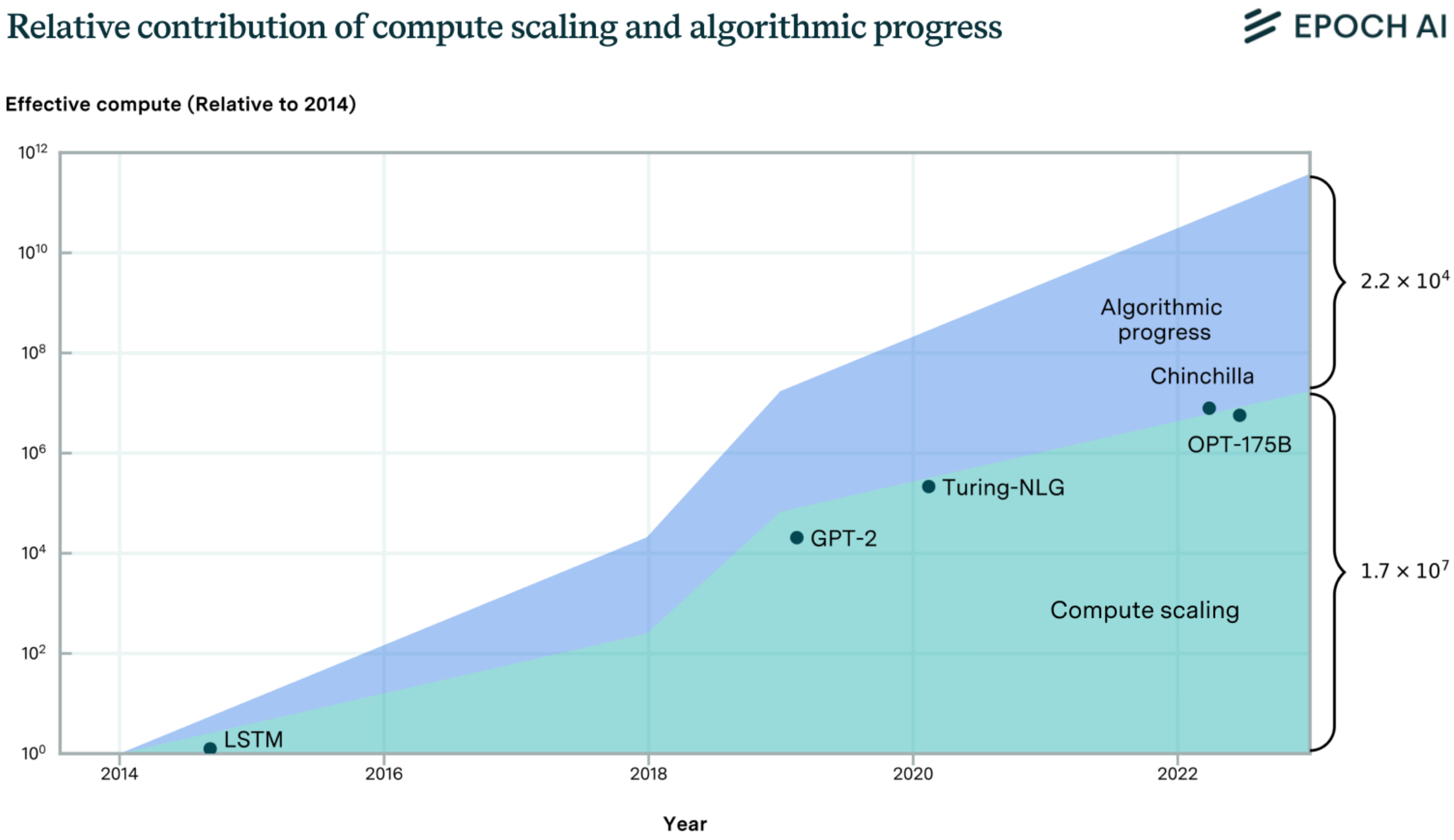
Video encode/decode,
Audio playback,
Camera RAW processing,
neural nets (future?)

~100-1000X
more efficient

Not programmable + costs 10-100's millions of dollars to design / verify / create

Credit: Pat Hanrahan for this slide design

AI Progress Relies on Hardware Improvement



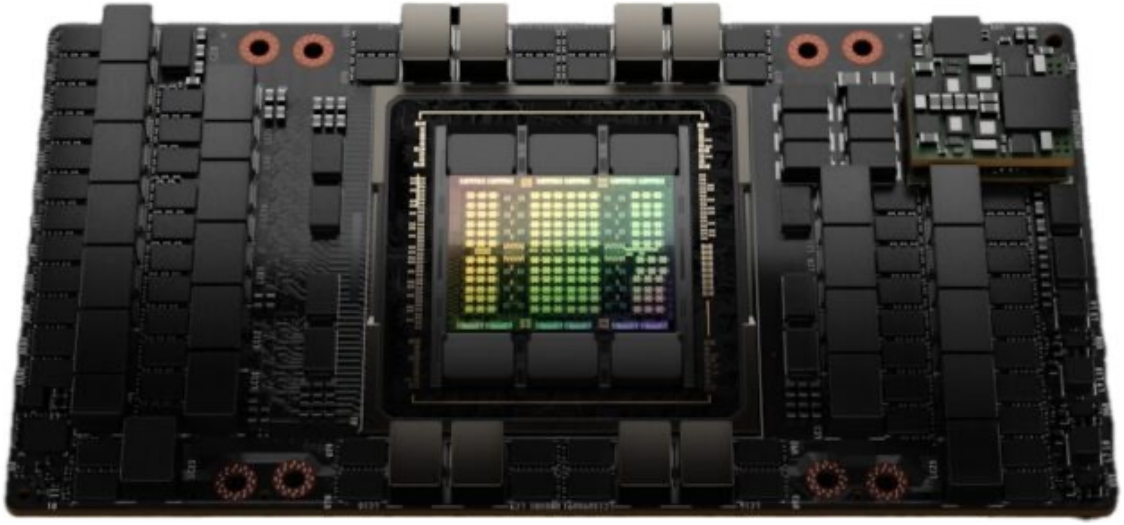
Algorithm

Compute

AI Models on GPUs

Many high-performance AI model implementations target GPUs

- High arithmetic intensity computations (computational characteristics similar to dense matrix-matrix multiplication)
- Benefit from flop-rich GPU architectures
- Highly-optimized library of kernels exist for GPUs (cuDNN)



NVIDIA H100

Why might a GPU be a sub-optimal platform for AI Model Acceleration?

(Hint: is a general purpose processor needed?)

Characteristics of An Ideal AI Model Accelerator

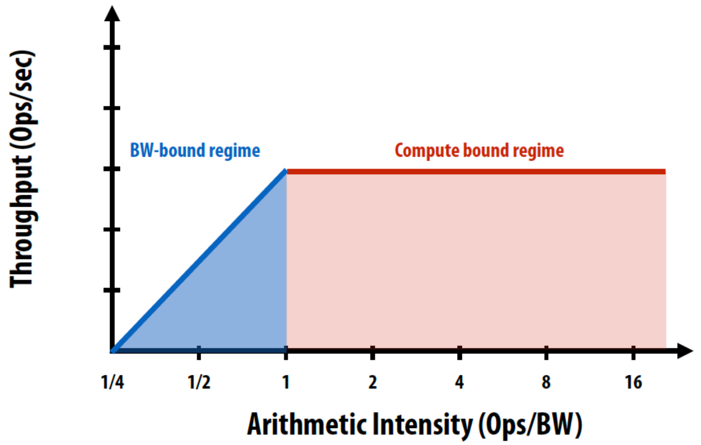
High peak TFLOPs and energy efficiency

High memory bandwidth

Simple to program for high-performance

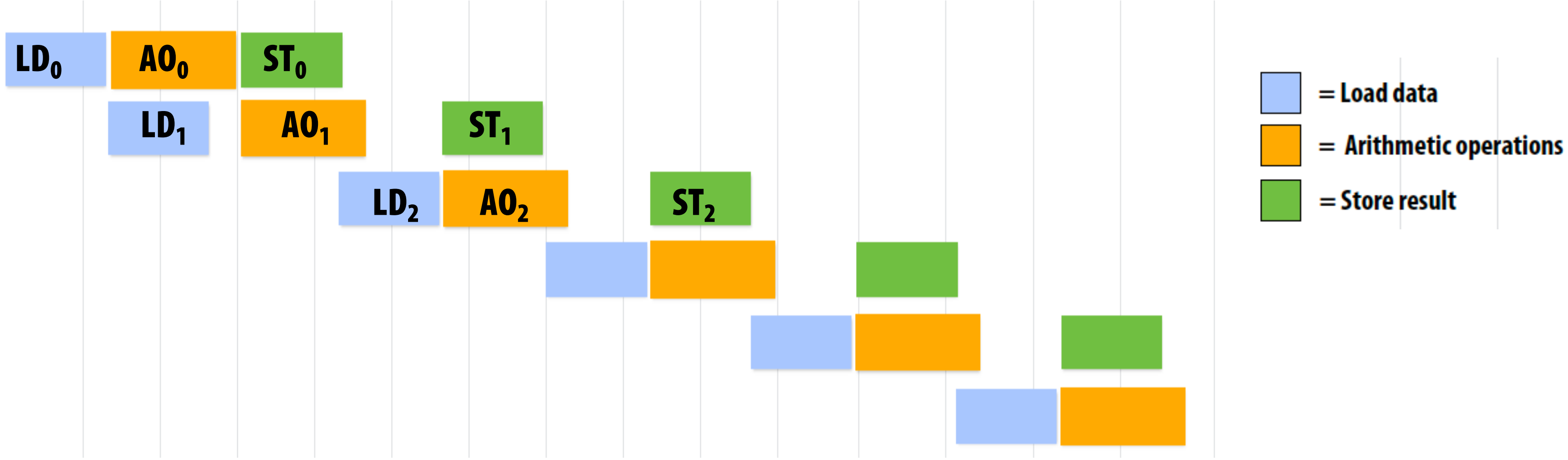
Reaches performance bound on compute-bound models

Reaches performance bound on BW-bound models



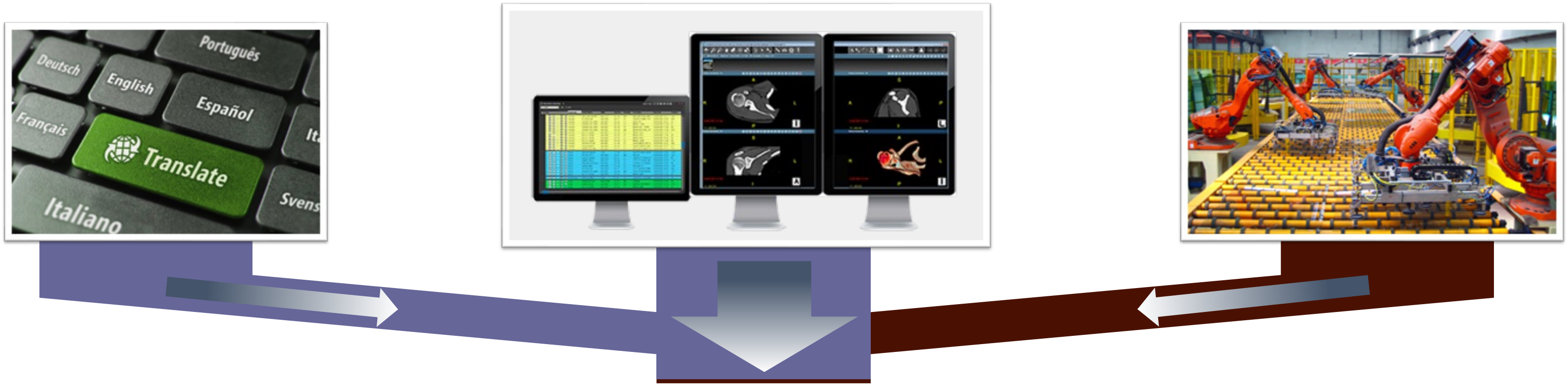
Asynchronous (Nonblocking) Execution

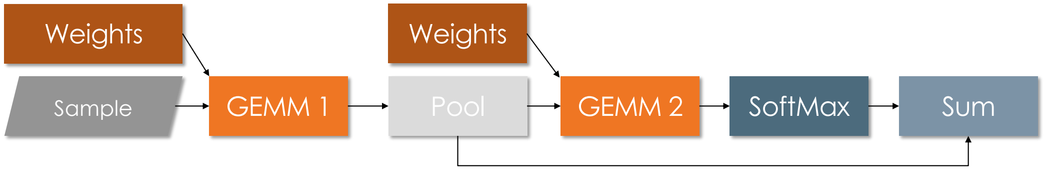
LD_0
AO_0
ST_0
LD_1
AO_1
ST_1
LD_2
AO_2
ST_2



Start later operations before earlier operations are complete

AI Models are Dataflow Graphs




Weights

Sample

GEMM 1

Weights

Pool

GEMM 2

SoftMax

Sum

Ideal AI Model Accelerator

Tiled AI accelerator programming model

- CUTLASS
- Triton
- Thunderkittens

Feature	Why?
Tiled tensors (e.g. 16 x 16, 32 x 32)	Max TFLOPS on GEMM Low instr. overhead

GEMM computation is cheap, but data movement is expensive

- Silicon area
- Watts
- Nanoseconds

Ideal: Minimize cost of Data Movement

Feature	Why?
Tiled tensors (e.g. 16 x 16, 32 x 32)	Max TFLOPS on GEMM Low instr. overhead
Asynchronous compute	Overlap compute and memory access
Asynchronous memory access	Overlap compute and memory access
Asynchronous chip-to-chip communication	Overlap compute, memory and communication

Ideal: Avoid Off-chip Data Access

Feature	Why?
Tiled tensors (e.g. 16 x 16, 32 x 32)	Max TFLOPS on GEMM Low instr. overhead
Asynchronous compute	Overlap compute and memory access
Asynchronous memory access	Overlap compute and memory access
Asynchronous chip-to-chip communication	Overlap compute, memory and communication
Compute unit to compute unit comm.	Fusion and pipelining Streaming Dataflow

Special instruction support

Recall: compute specialization = energy efficiency

Rules of thumb: compared to high-quality C code on CPU...

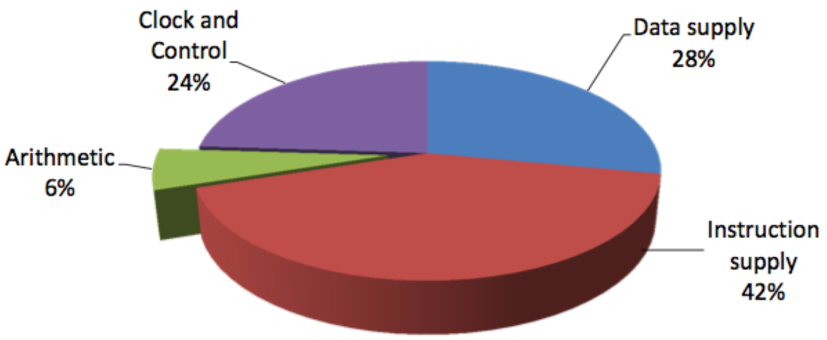
Throughput-maximized processor architectures: e.g., GPU cores

- Approximately 10x improvement in perf / watt
- Assuming code maps well to wide data-parallel execution and is compute bound

Fixed-function ASIC ("application-specific integrated circuit")

- Can approach 100-1000x or greater improvement in perf/watt
- Assuming code is compute bound and

  and is not floating-point math



Efficient Embedded Computing [Dally et al. 08]

[Figure credit Eric Chung]

[Source: Chung et al. 2010 , Dally 08]

Recall: data movement has high energy cost

Rule of thumb in modern system design: always seek to reduce amount of data movement in a computer

"Ballpark" numbers

- Integer op: ~ 1 pJ *
- Floating point op: ~20 pJ *
- Reading 64 bits from small local SRAM (1mm away on chip): ~ 26 pJ
- Reading 64 bits from low power mobile DRAM (LPDDR): ~1200 pJ

[Sources: Bill Dally (NVIDIA), Tom Olson (ARM)]

* Cost to just perform the logical operation, not counting overhead of instruction decode, load data from registers, etc.

Amortize overhead of instruction stream control using more complex instructions

Estimated overhead of programmability (instruction stream, control, etc.)

- Half-precision FMA (fused multiply-add)	2000%
- Half-precision DP4 (vec4 dot product)	500%
- Half-precision 4x4 MMA (matrix-matrix multiply + accumulate)	27%

Key principle: amortize cost of instruction stream processing across many operations of a single complex instruction

Numerical data formats

		Range	Accuracy
FP32	1 8 23 S E M	10^{-38} - 10^{38}	.000006%
FP16	1 5 10 S E M	$6\text{x}10^{-5}$ - $6\text{x}10^{4}$	.05%
Int32	1 31 S M	0 – $2\text{x}10^{9}$	Exact
Int16	1 15 S M	0 – $6\text{x}10^{4}$	Exact
Int8	1 7 S M	0 – 127	Exact
BF16	**1 8 7** **S E M**	**BF16: Same range as FP32, but lower accuracy**	
BF8 E4M3	**1 4 3** **S E M**	**0 - 448**	
BF8 E5M2	**1 5 2** **S E M**	**0 - 57344**	

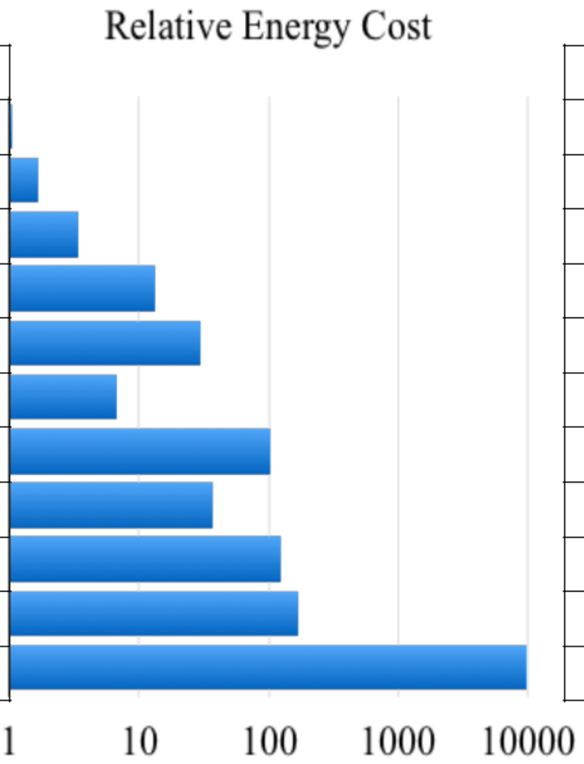
Reminder:

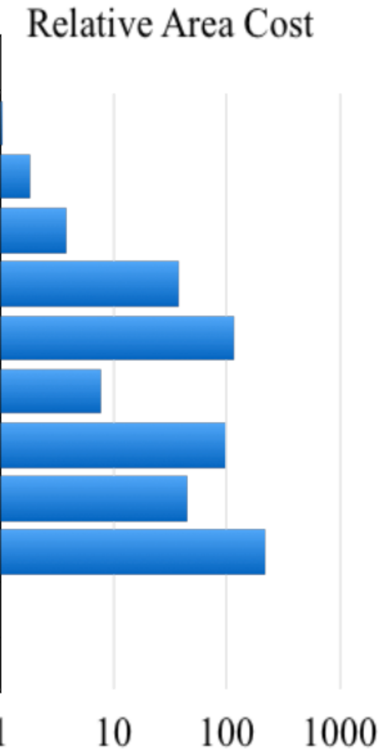
$$-1^{S} \times (1 + (M \times 2^{-23})) \times 2^{(E-127)}$$

Slide credit: Bill Dally

Energy and Area Cost of Compute

Operation:	Energy (pJ)	Area (μm^2)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A





Energy numbers are from Mark Horowitz "Computing's Energy Problem (and what we can do about it)", ISSCC 2014
Area numbers are from synthesized result using Design Compiler under TSMC 45nm tech node. FP units used DesignWare Library.

Ampere GPU SM (A100)

Each SM core has:

64 fp32 ALUs (mul-add)

32 int32 ALUs

4 “tensor cores”

Execute 8x4 x 4x8 matrix mul-add instr

A x B + D for matrices A,B,D

A, B stored as fp16, accumulation with fp32 D

There are 108 SM cores in the GA100 GPU:

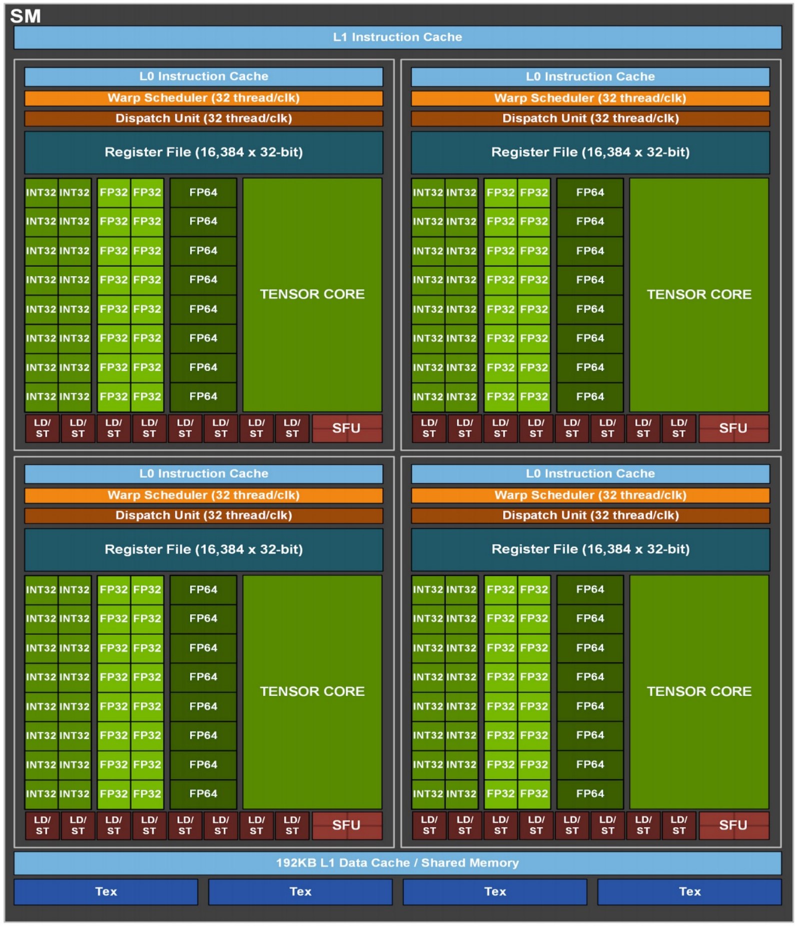
6,912 fp32 mul-add ALUs

432 tensor cores

1.4 GHz max clock

= 19.5 TFLOPs fp32

+ 312 TFLOPs (fp16/32 mixed) in tensor core


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

Single instruction to perform 8x4 x 4x8 FP16 + 8x8 TF32 ops

Nvidia H100 GPU (2022)

Fourth-generation Tensor Core

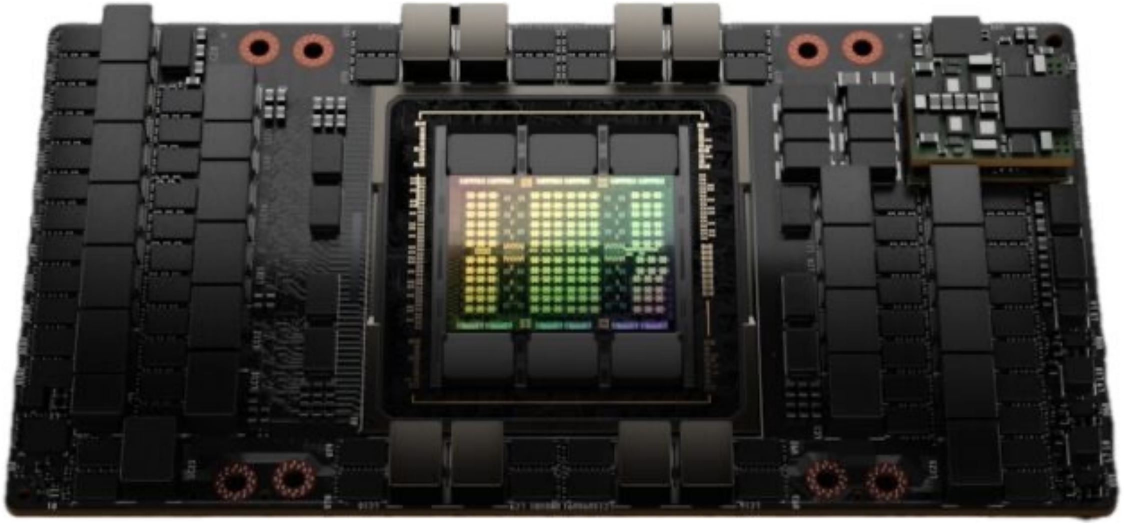
Tensor Memory Accelerator (TMA) unit

CUDA cluster capability

HBM3 with up to 80 GB

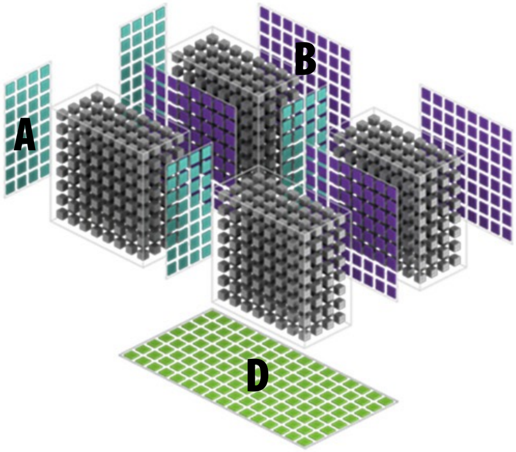
TSMC 4nm

80 Billion transistors

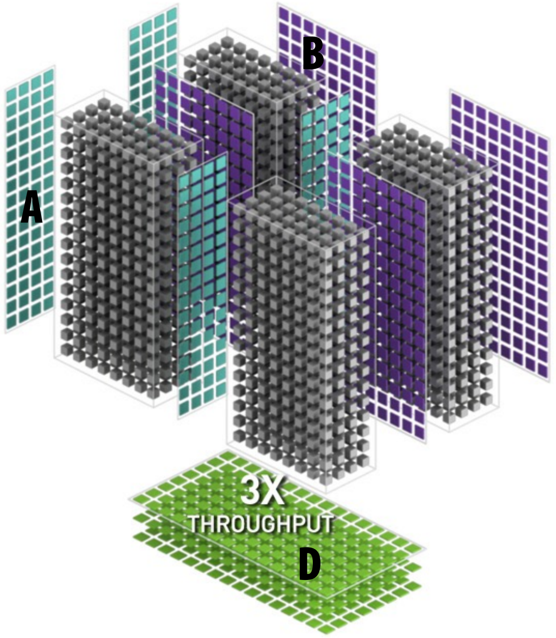


Tensor cores

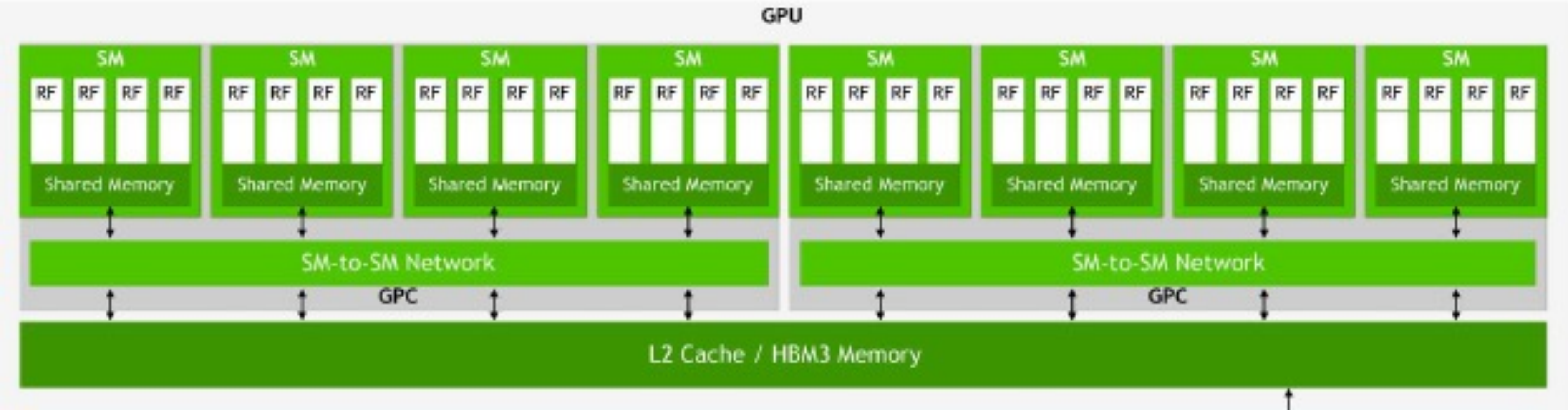
A100 FP16



H100 FP16



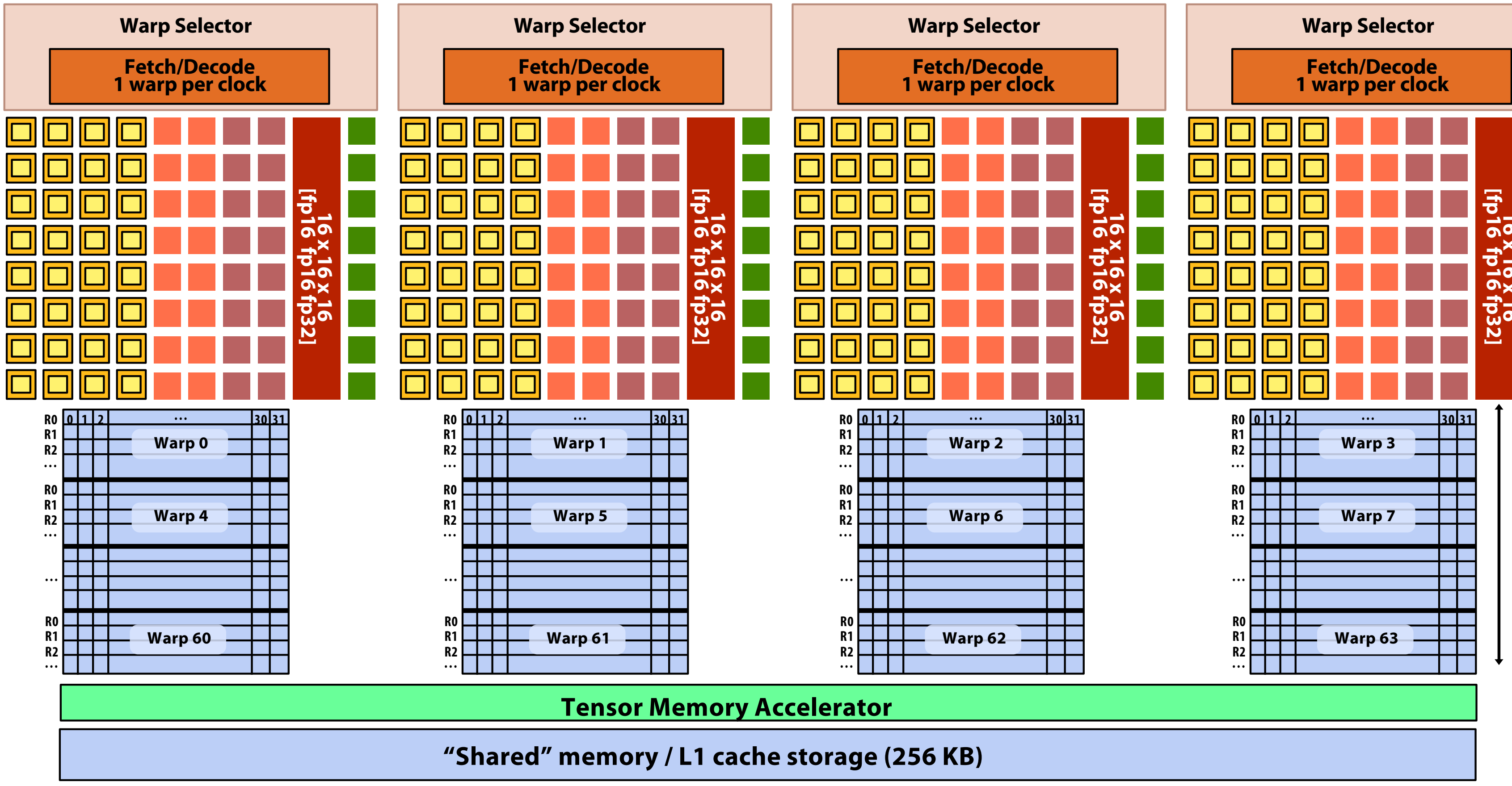
H100 CUDA, Compute and Memory Hierarchies



CUDA Hierarchy	Compute Hierarchy	Memory Hierarchy
Grid	GPU	80 GB HBM/ 50 MB L2
Cluster	CPC	256 KB shared memory per SM
Thread Block	SM	256 KB shared memory
Threads	SIMD Lanes	1 KB RF per thread, 64KB per SM partition

- Thread block cluster is a collective of up to 16 thread blocks
- Each thread block is guaranteed to execute on a separate SM and to run at the same time

H100 GPU Streaming Multi-processor (SM)



64 KB registers per sub-core

256 KB registers in total per SM

Registers divided among (up to) 64 "warps" per SM

🟨 = SIMD fp32 functional unit, control shared across 16 units (32 x MUL-ADD per clock *)

🟧 = SIMD int functional unit, control shared across 16 units (16 x MUL/ADD per clock **)

🟫 = SIMD fp64 functional unit, control shared across 8 units (16 x MUL/ADD per clock **)

🟥 = Tensor core unit

🟩 = Load/store unit

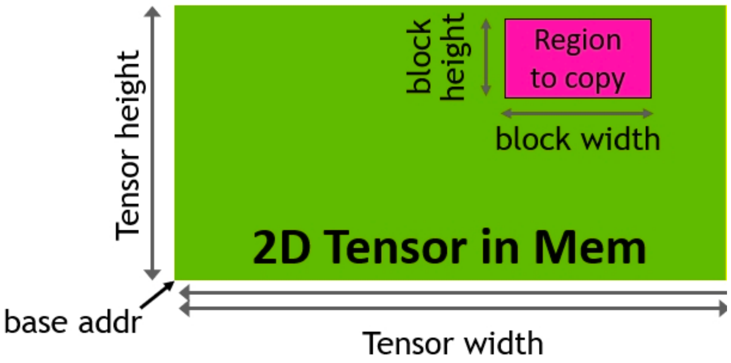
* one 32-wide SIMD operation every clock

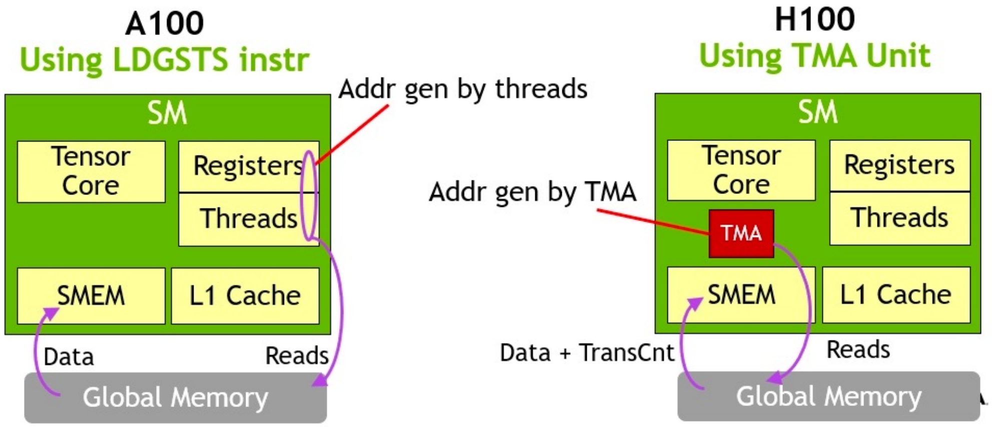
** one 32-wide SIMD operation every 2 clocks

Stanford CS149, Fall 2025

Tensor Memory Accelerator

Copy Descriptor





Special purpose instructions for efficient data movement

Asynchronously load/store a region of a tensor from global to shared memory

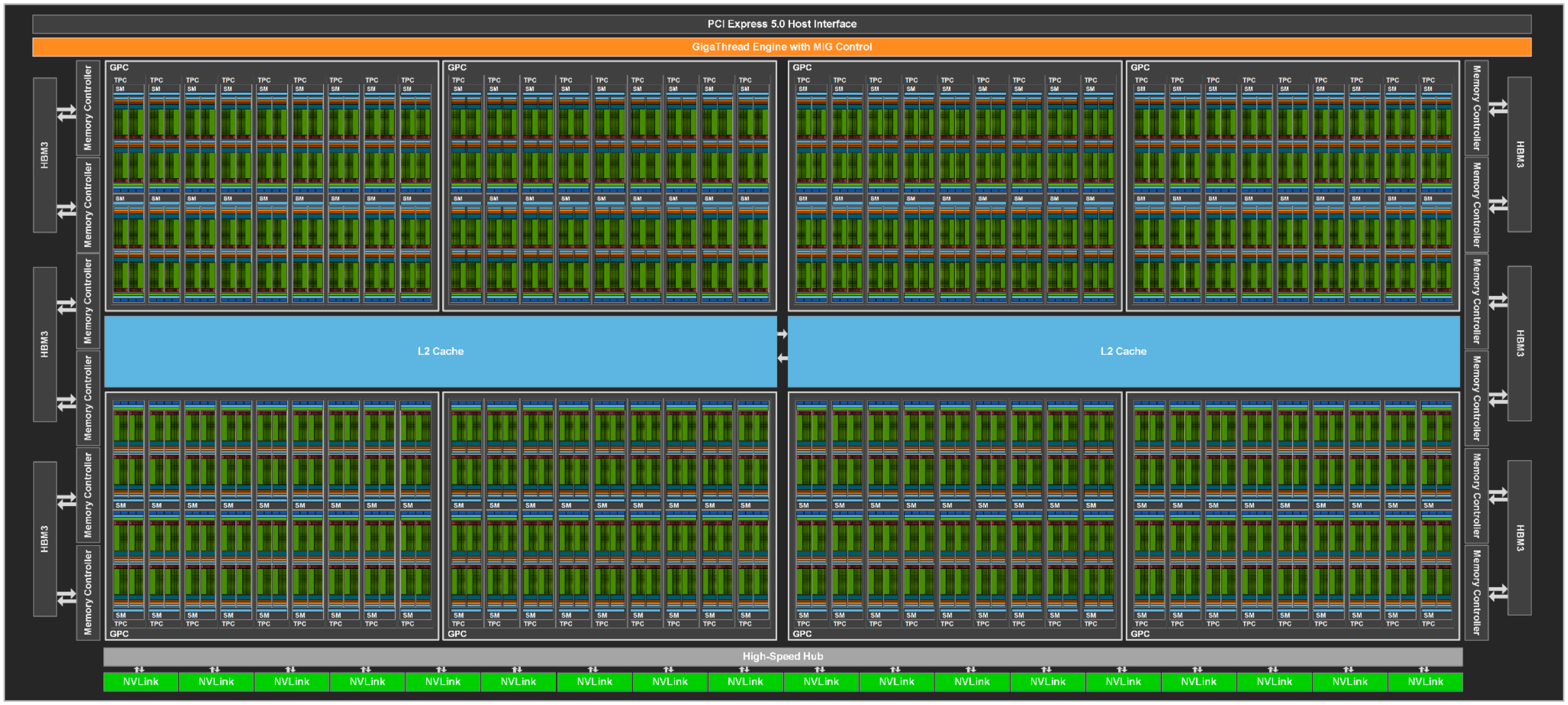
Copy descriptor describes region

Single thread issue TMA operation `cuda:memcpy_async`

Signal barrier when copy is complete

Hardware address generation and data movement

The Whole H100

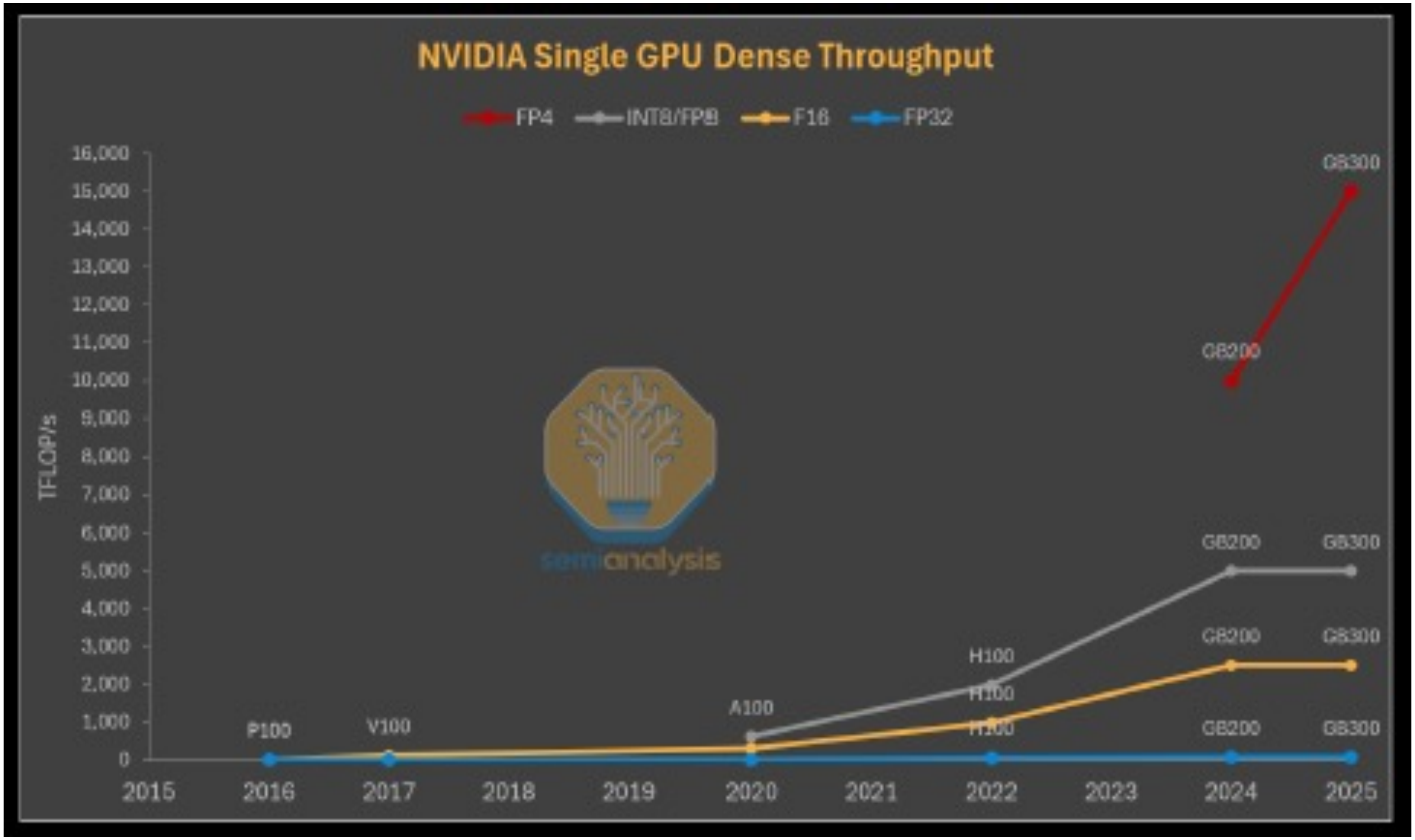

PCI Express 5.0 Host Interface
GigaThread Engine with MIG Control
GPC
TPC
SM
L2 Cache
Memory Controller
HBM3
High-Speed Hub
NVLink

144 SMs

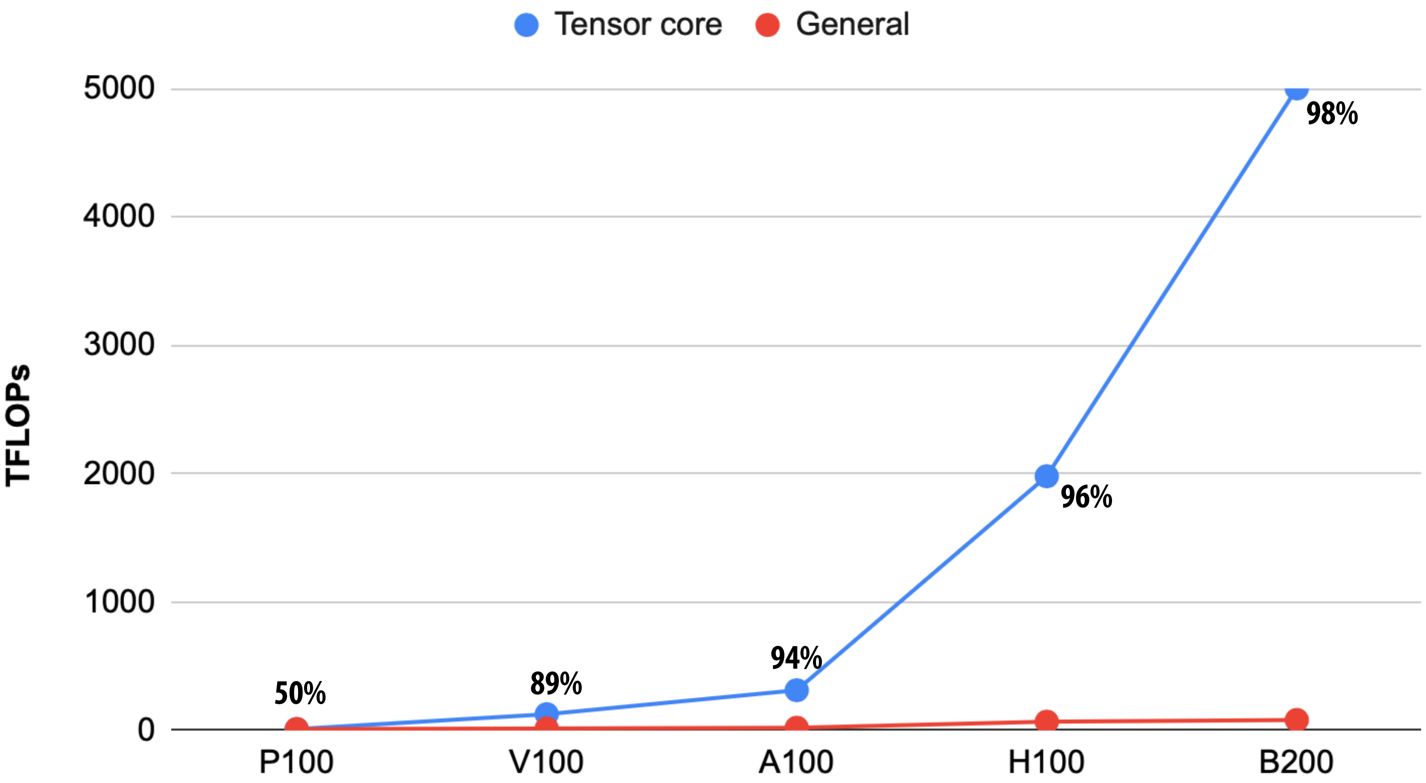
Tensor cores (systolic array MMA): 989 TFLOPS (fp16)

SIMD: 134 TFLOPS (fp16), 67 TFLOPS (fp32)

GPU TFLOPS Over Time

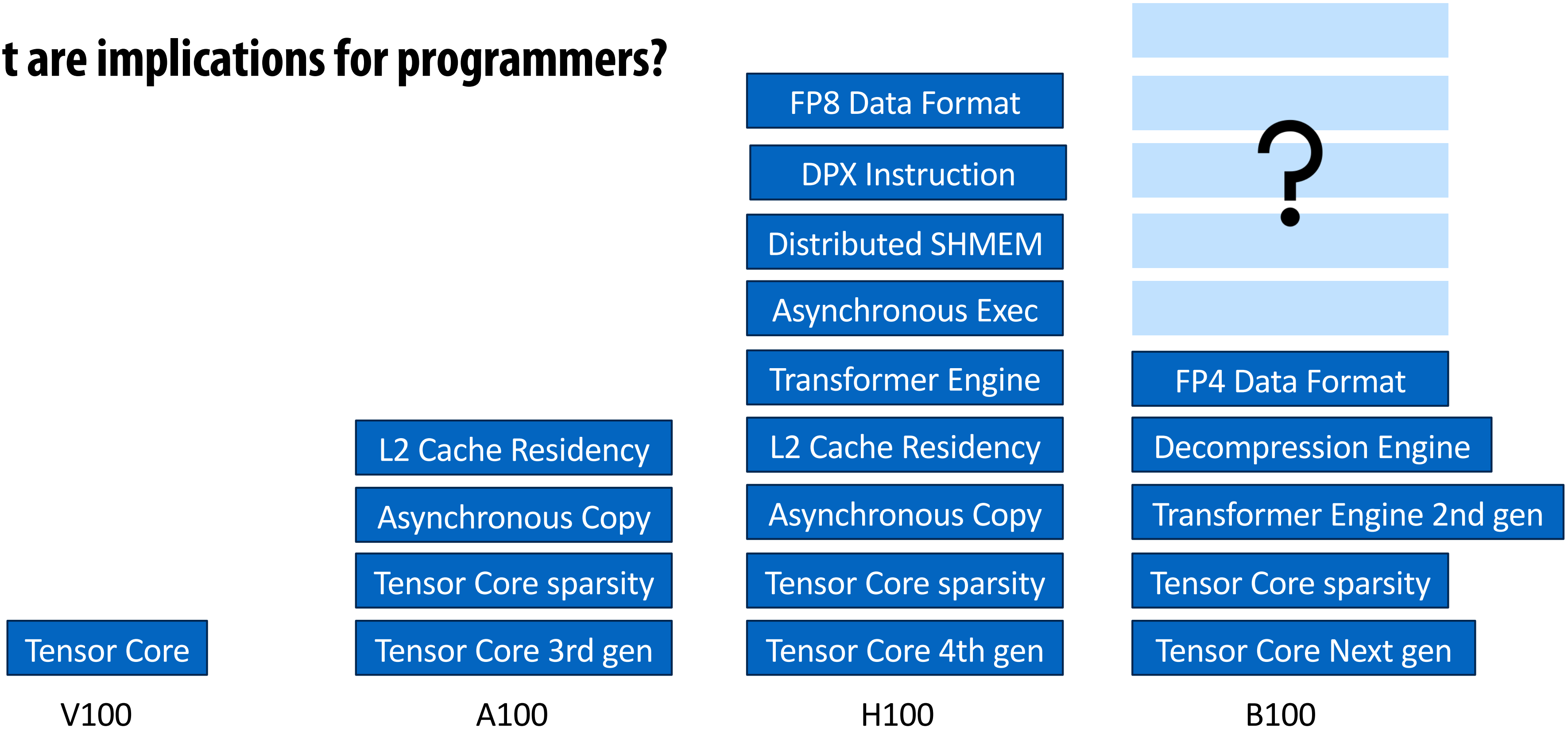


All the TFLOPS are in the Tensor Cores



Nvidia Chips Becoming More Specialized

What are implications for programmers?



V100

Tensor Core

A100

L2 Cache Residency

Asynchronous Copy

Tensor Core sparsity

Tensor Core 3rd gen

H100

FP8 Data Format

DPX Instruction

Distributed SHMEM

Asynchronous Exec

Transformer Engine

L2 Cache Residency

Asynchronous Copy

Tensor Core sparsity

Tensor Core 4th gen

B100

?

FP4 Data Format

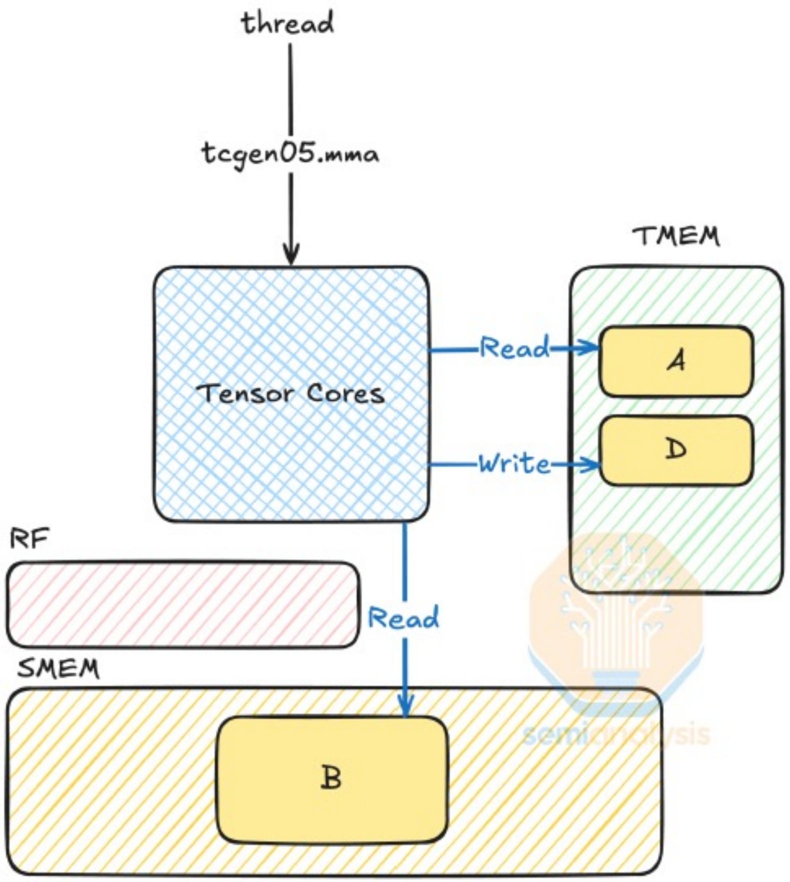
Decompression Engine

Transformer Engine 2nd gen

Tensor Core sparsity

Tensor Core Next gen

Tensor Cores in B100



Register bandwidth limits for tensor cores in B100

Tensor data in SMEM and TMEM

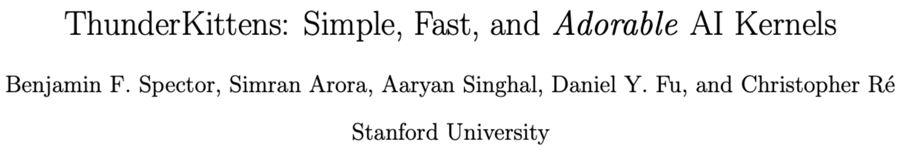
Single threads execute MMA ⇒ No more warps!

Programming Tensor Cores

- **Allocate TMEM and descriptors**
  - tcgen05.alloc
- **Prefetch/stream tiles with TMA (async)**
  - cp.async.bulk.tensor, coordinate with mbarrier
- **Launch async MMAs**
  - tcgen05.mma batch with tcgen05.commit
- **Order & retire**
  - tcgen05.fence

Not your father's CUDA

DSLs for GPU AI Kernels

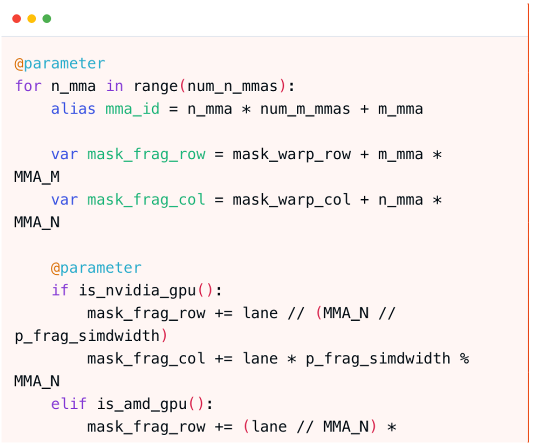


ThunderKittens: Simple, Fast, and *Adorable* AI Kernels

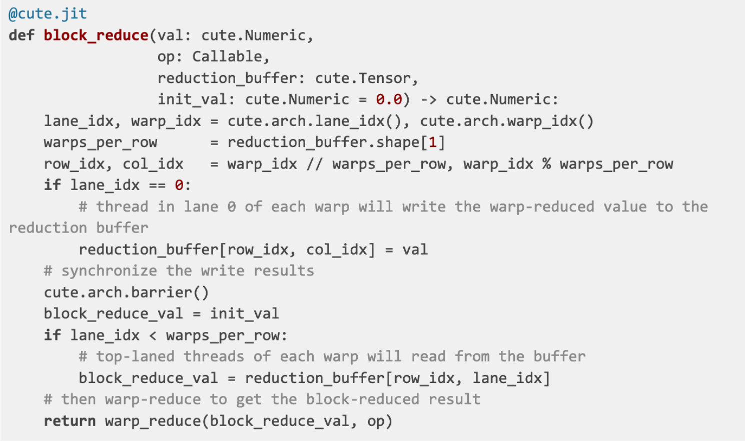
Benjamin F. Spector, Simran Arora, Aaryan Singhal, Daniel Y. Fu, and Christopher Ré

Stanford University

Mojo🔥



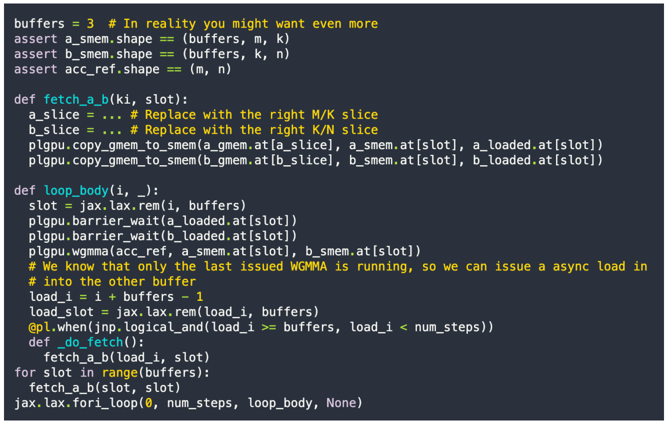
```
@parameter
for n_mma in range(num_n_mmas):
    alias mma_id = n_mma * num_m_mmas + m_mma

    var mask_frag_row = mask_warp_row + m_mma *
MMA_M
    var mask_frag_col = mask_warp_col + n_mma *
MMA_N

    @parameter
    if is_nvidia_gpu():
        mask_frag_row += lane // (MMA_N //
p_frag_simdwidth)
        mask_frag_col += lane * p_frag_simdwidth %
MMA_N
    elif is_amd_gpu():
        mask_frag_row += (lane // MMA_N) *
```



```
@cute.jit
def block_reduce(val: cute.Numeric,
                 op: Callable,
                 reduction_buffer: cute.Tensor,
                 init_val: cute.Numeric = 0.0) -> cute.Numeric:
    lane_idx, warp_idx = cute.arch.lane_idx(), cute.arch.warp_idx()
    warps_per_row      = reduction_buffer.shape[1]
    row_idx, col_idx   = warp_idx // warps_per_row, warp_idx % warps_per_row
    if lane_idx == 0:
        # thread in lane 0 of each warp will write the warp-reduced value to the
reduction buffer
        reduction_buffer[row_idx, col_idx] = val
    # synchronize the write results
    cute.arch.barrier()
    block_reduce_val = init_val
    if lane_idx < warps_per_row:
        # top-laned threads of each warp will read from the buffer
        block_reduce_val = reduction_buffer[row_idx, lane_idx]
    # then warp-reduce to get the block-reduced result
    return warp_reduce(block_reduce_val, op)
```

Cute-DSL
(CUTLASS in Python)

Mosaic GPU



```
buffers = 3  # In reality you might want even more
assert a_smem.shape == (buffers, m, k)
assert b_smem.shape == (buffers, k, n)
assert acc_ref.shape == (m, n)

def fetch_a_b(ki, slot):
  a_slice = ... # Replace with the right M/K slice
  b_slice = ... # Replace with the right K/N slice
  plgpu.copy_gmem_to_smem(a_gmem.at[a_slice], a_smem.at[slot], a_loaded.at[slot])
  plgpu.copy_gmem_to_smem(b_gmem.at[b_slice], b_smem.at[slot], b_loaded.at[slot])

def loop_body(i, _):
  slot = jax.lax.rem(i, buffers)
  plgpu.barrier_wait(a_loaded.at[slot])
  plgpu.barrier_wait(b_loaded.at[slot])
  plgpu.wgmma(acc_ref, a_smem.at[slot], b_smem.at[slot])
  # We know that only the last issued WGMMA is running, so we can issue a async load in
  # into the other buffer
  load_i = i + buffers - 1
  load_slot = jax.lax.rem(load_i, buffers)
  @pl.when(jnp.logical_and(load_i >= buffers, load_i < num_steps))
  def _do_fetch():
    fetch_a_b(load_i, slot)
for slot in range(buffers):
  fetch_a_b(slot, slot)
jax.lax.fori_loop(0, num_steps, loop_body, None)
```

How Ideal are GPUs

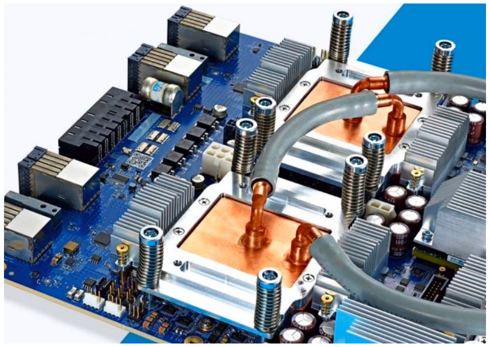
Feature	Why?	Nvidia GPU
Tiled tensors (e.g. 16 x 16, 32 x 32)	Max TFLOPS on GEMM Low instr. overhead	✅
Asynchronous compute	Overlap compute and memory access	✅ mma_async
Asynchronous memory access	Overlap compute and memory access	✅ TMA+TMEM
Asynchronous chip-to-chip communication	Overlap compute, memory and communication	
Compute unit to compute unit comm.	Fusion and pipelining Streaming Dataflow	? TB Cluster

AI Is Redefining Computing

And everyone is building silicon for it!

AI is the driving force behind new architectures, compilers, and system design

Hardware acceleration of AI inference/training

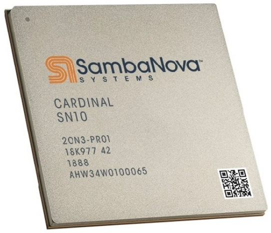


Google TPU3

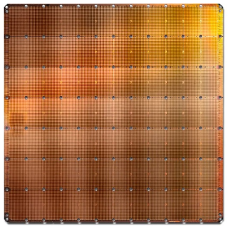
AWS Trainium 2

Apple Neural Engine

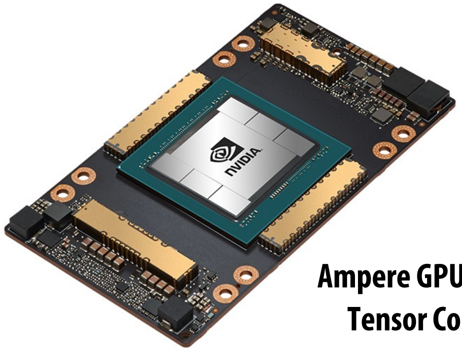
Intel Deep Learning Inference Accelerator



SambaNova Cardinal SN10



Cerebras Wafer Scale Engine



Ampere GPU with Tensor Cores

Google’s TPU (v1)

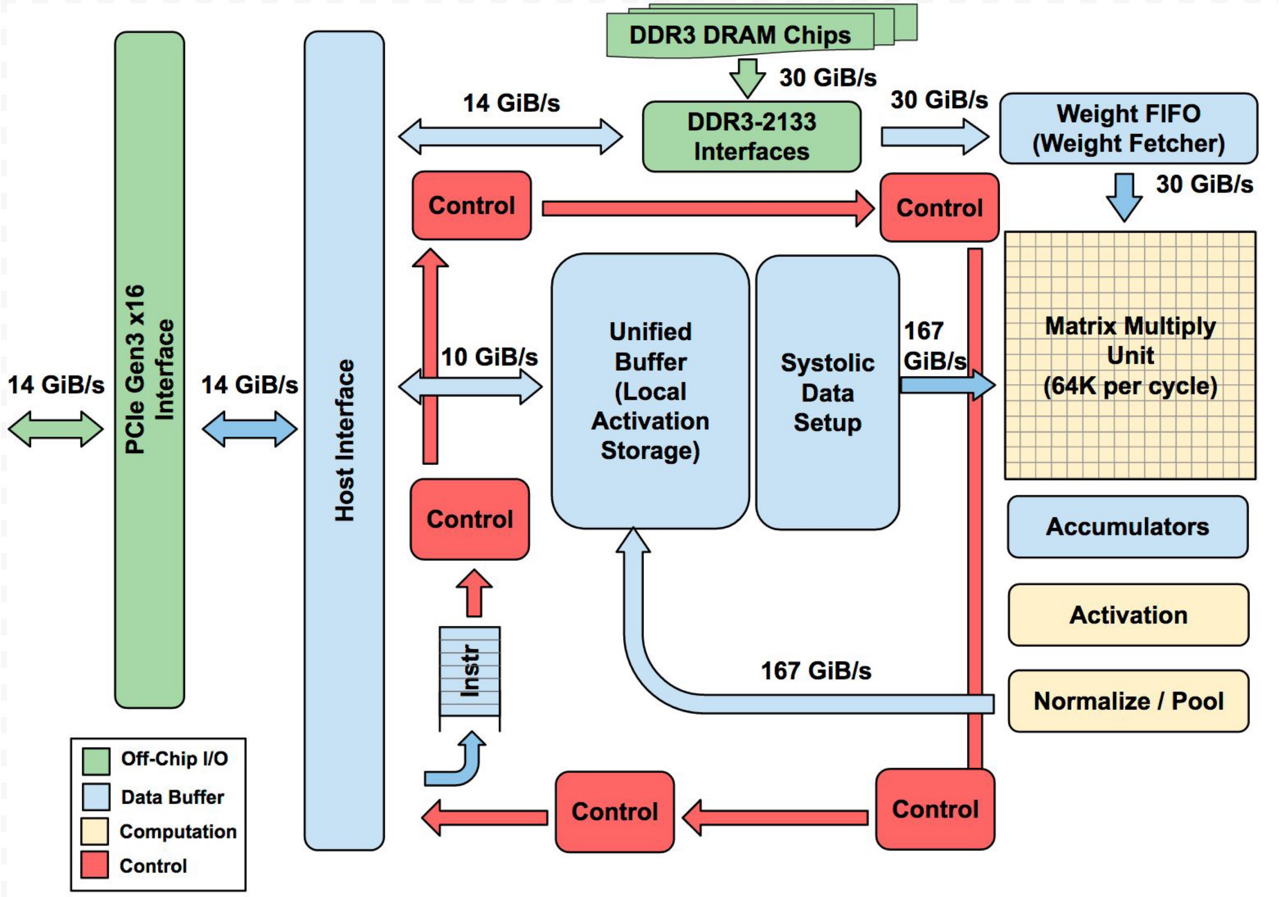
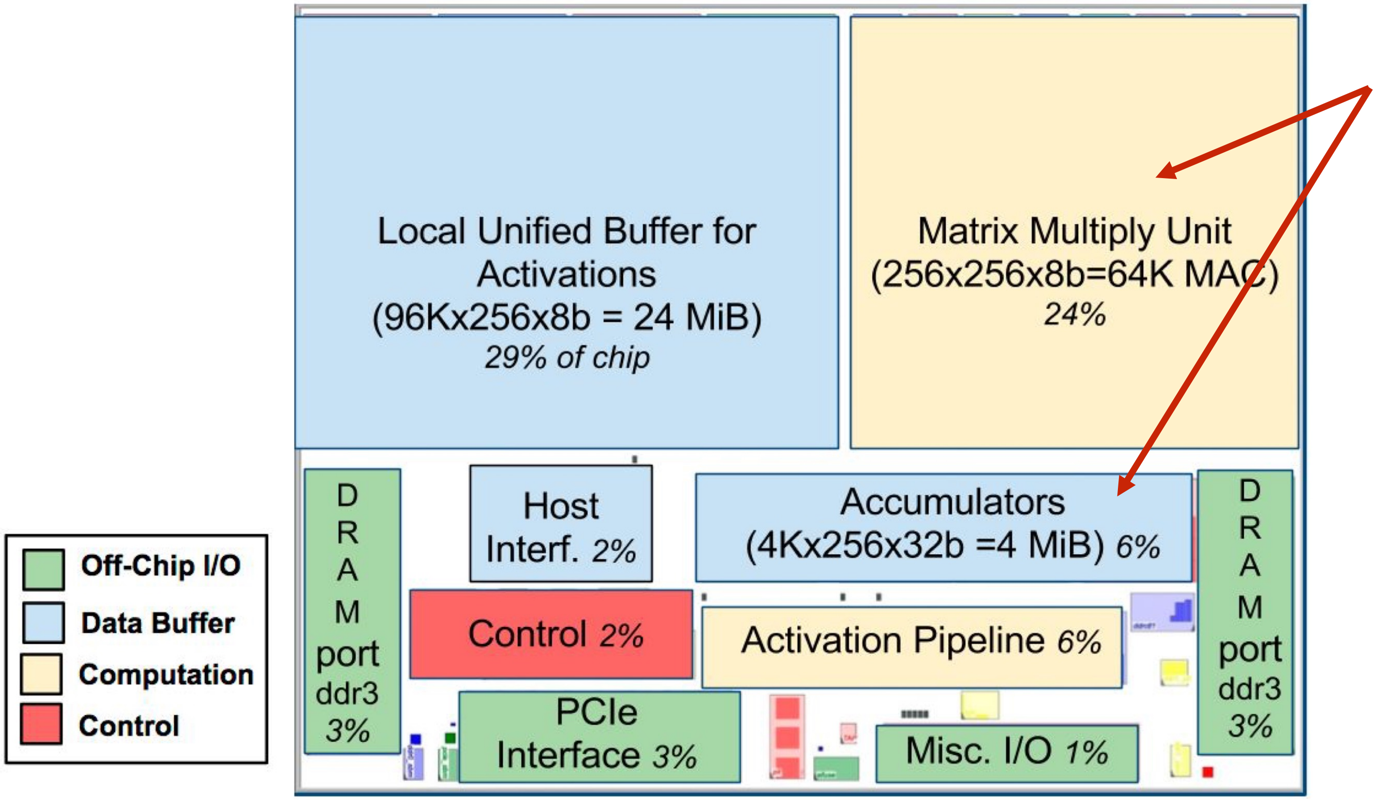


Figure credit: Jouppi et al. 2017

TPU area proportionality



Arithmetic units ~ 30% of chip

Note low area footprint of control

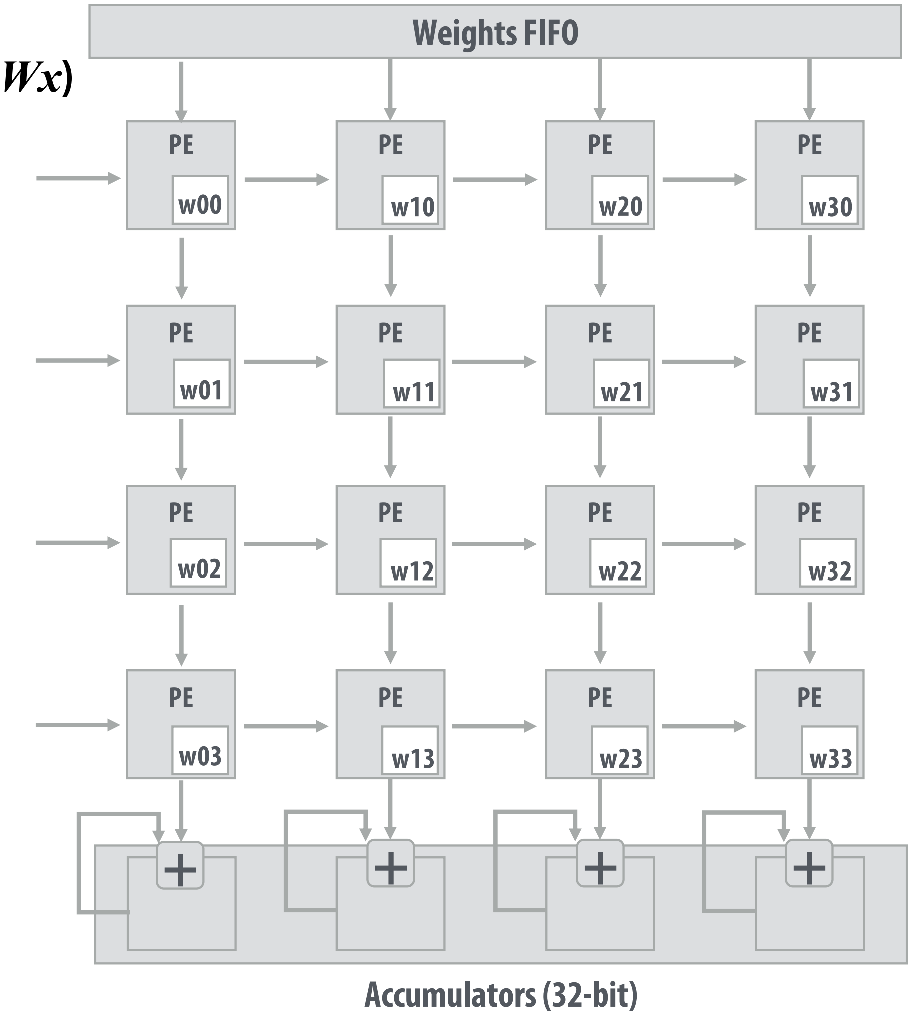
Key instructions:

- read host memory
- write host memory
- read weights
- matrix_multiply / convolve
- activate

Figure credit: Jouppi et al. 2017

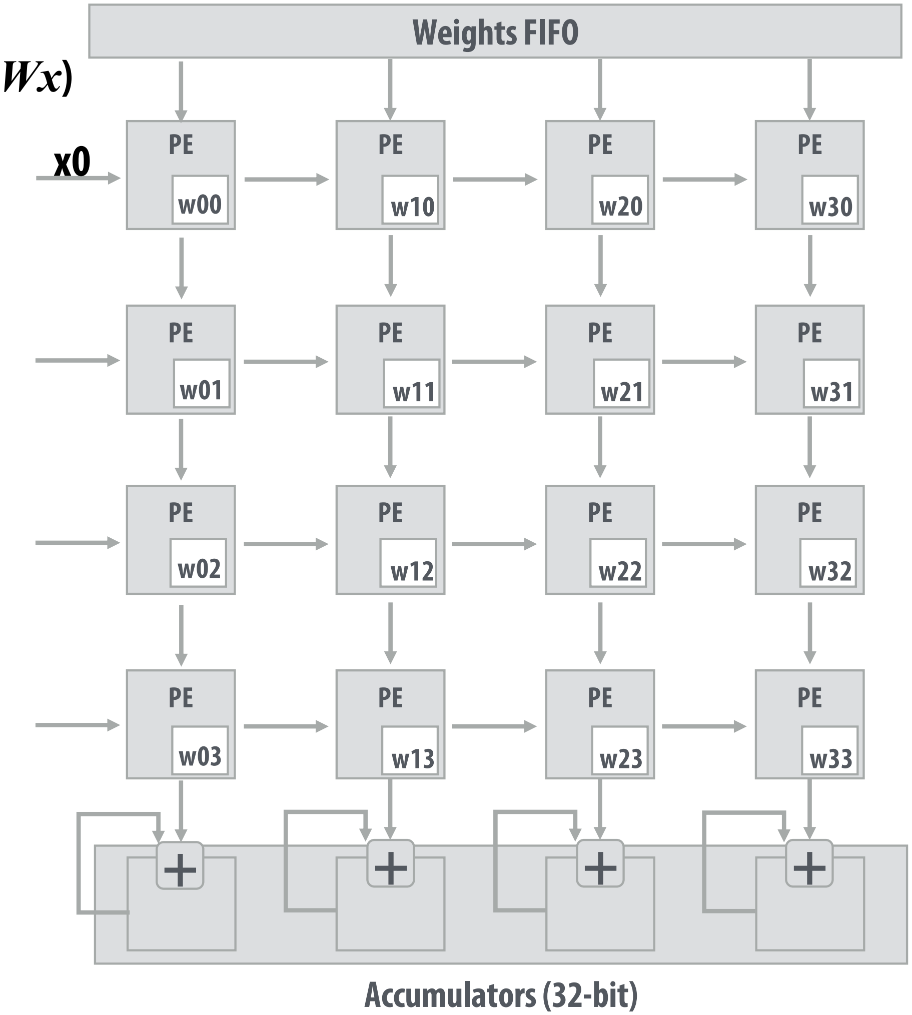
Systolic array

(matrix vector multiplication example: $y=Wx$)



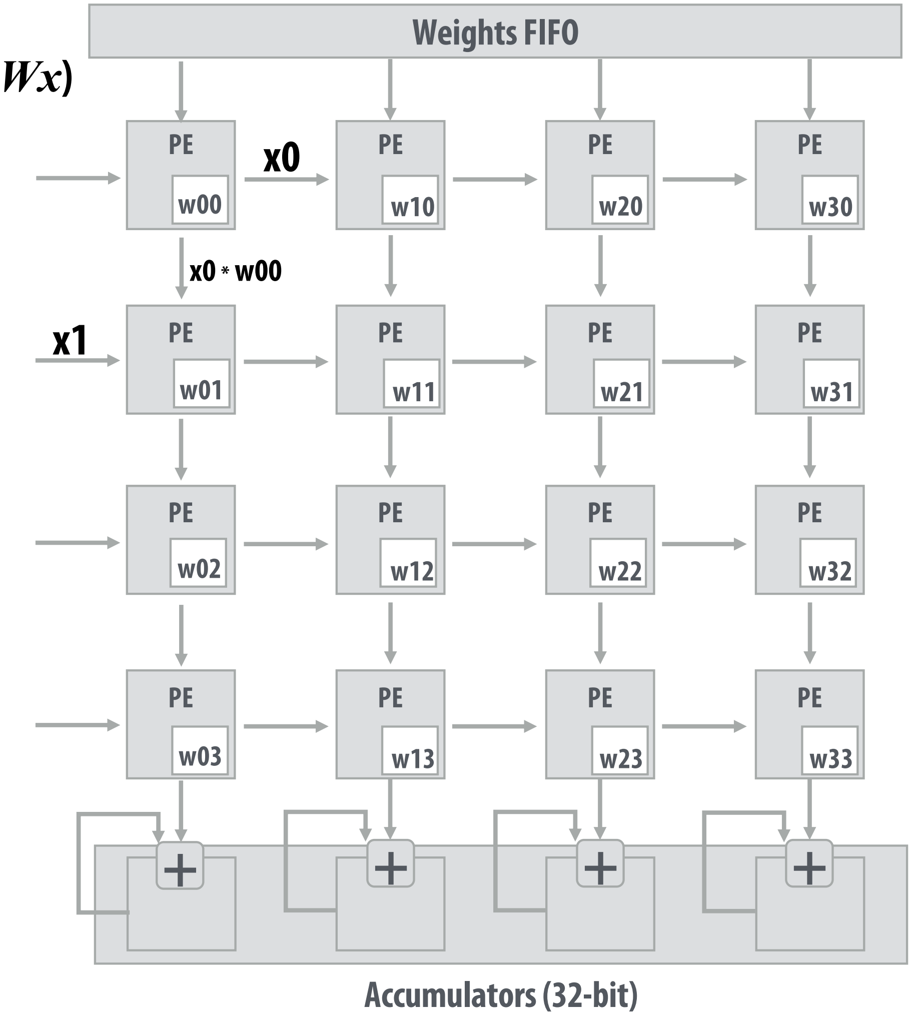
Systolic array

(matrix vector multiplication example: $y=Wx$)



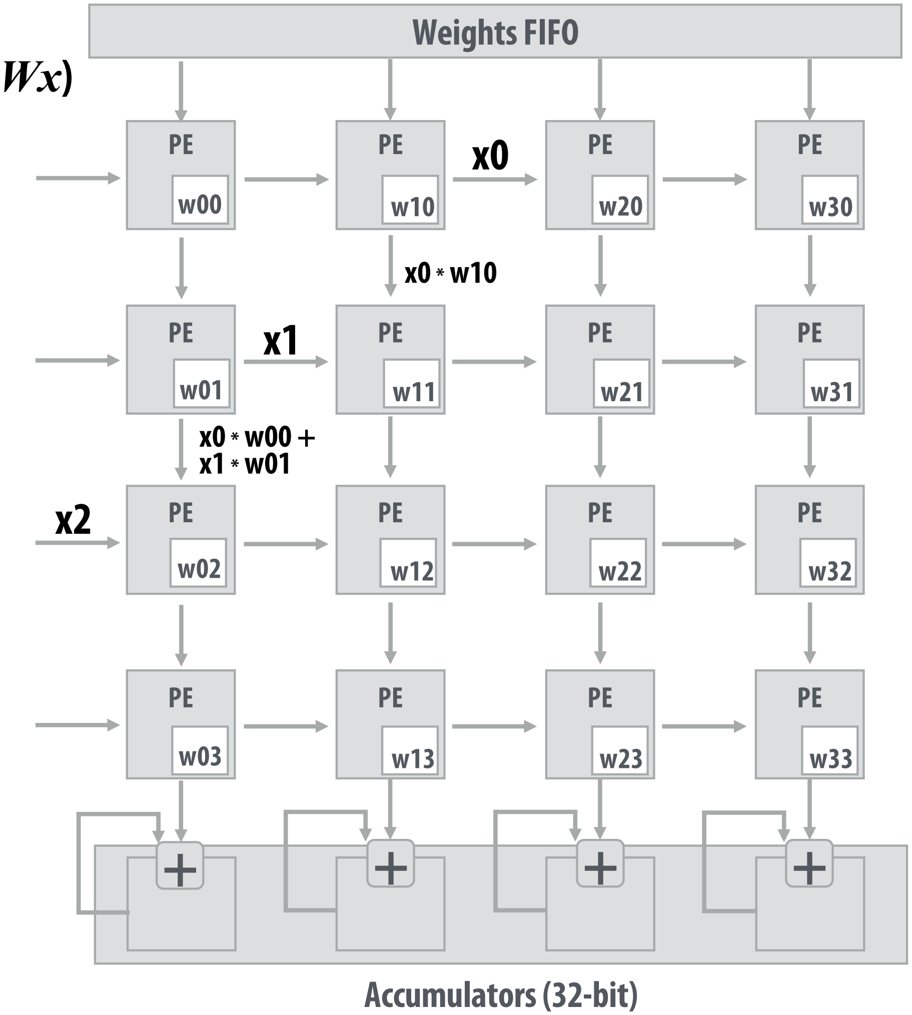
Systolic array

(matrix vector multiplication example: $y=Wx$)



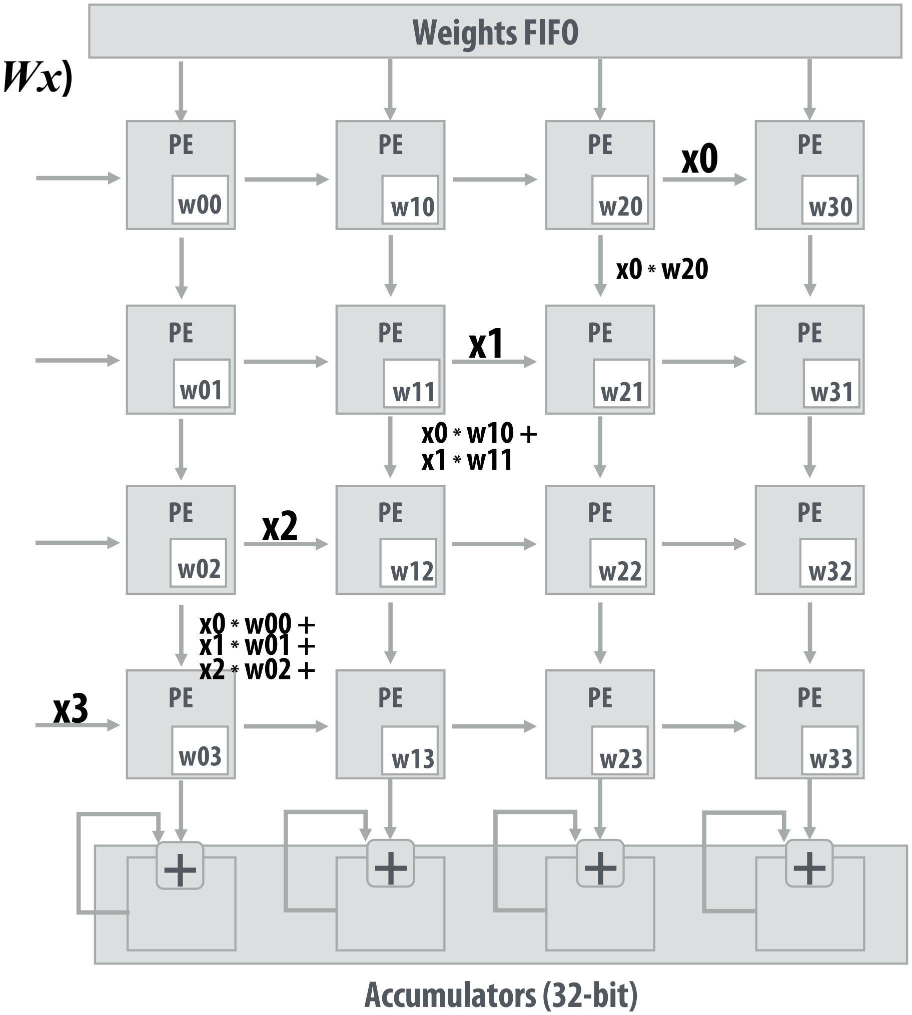
Systolic array

(matrix vector multiplication example: *y*=*Wx*)



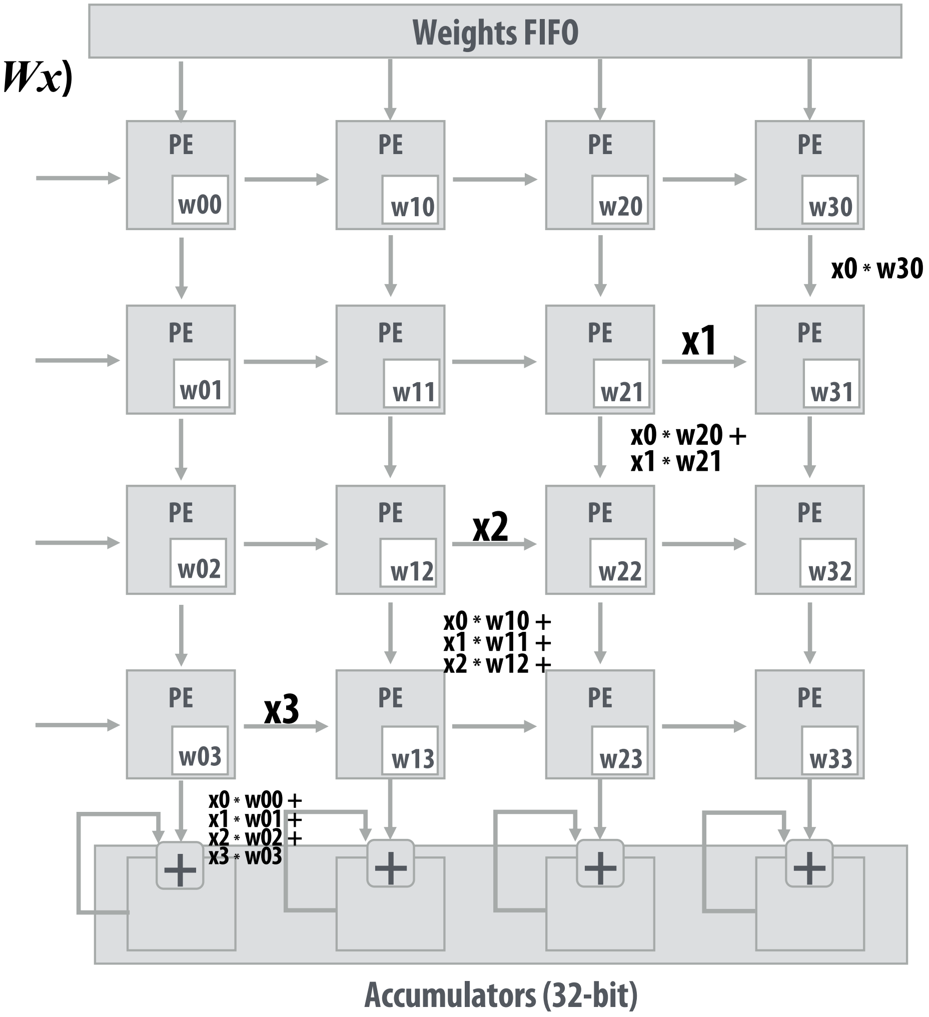
Systolic array

(matrix vector multiplication example: $y=Wx$)



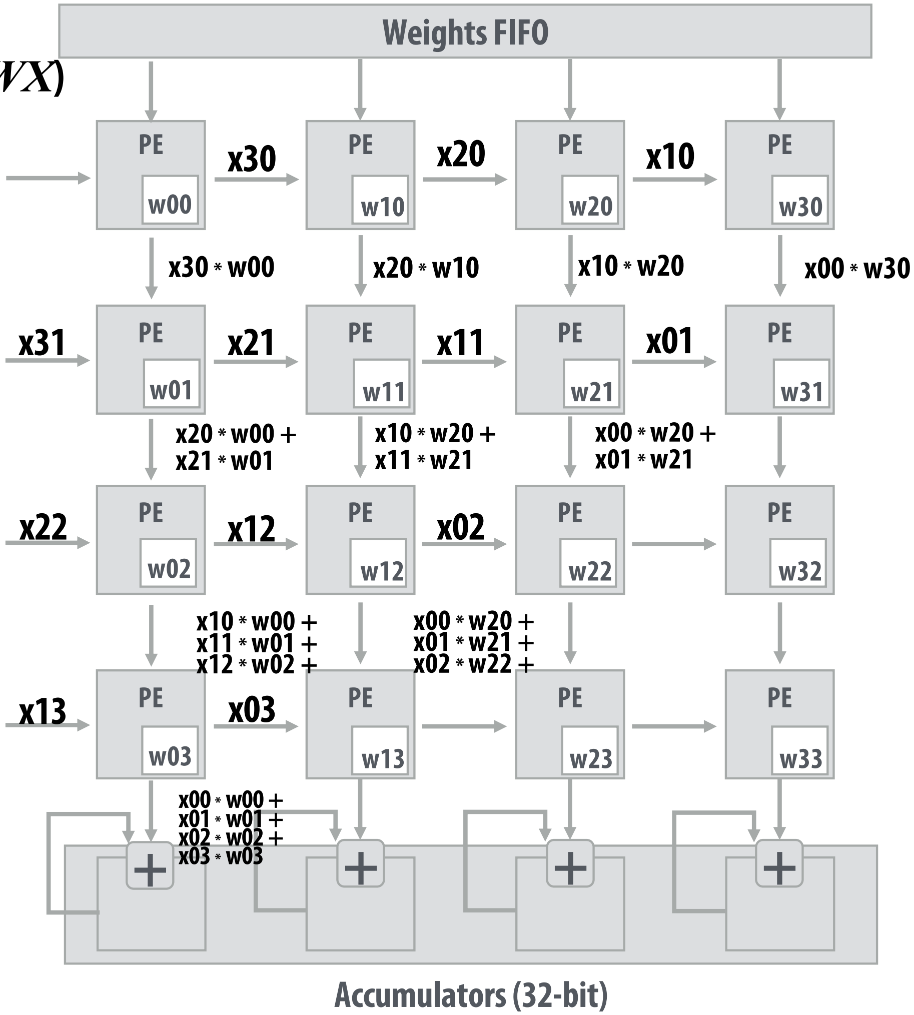
Systolic array

(matrix vector multiplication example: $y=Wx$)



Systolic array

(matrix matrix multiplication example: $Y=WX$)



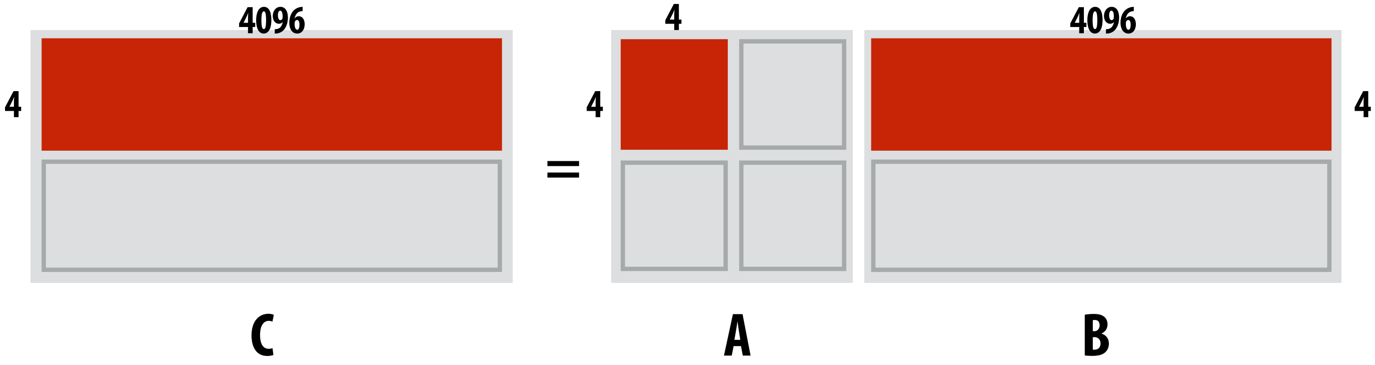
Notice: need multiple 4x32bit accumulators to hold output columns

SIMD vs. Systolic Array

Feature	SIMD	Systolic Array
Dataflow	Control-driven (instructions)	Data-driven (wavefront)
Locality (data reuse)	Limited	Temporal and spatial
Communication	Global (register/memory)	Local (neighbor PEs)
Control	Centralized	Distributed
Efficiency (perf/mm^2, perf/Watt)	Medium	Very high

Building larger matrix-matrix multiplies

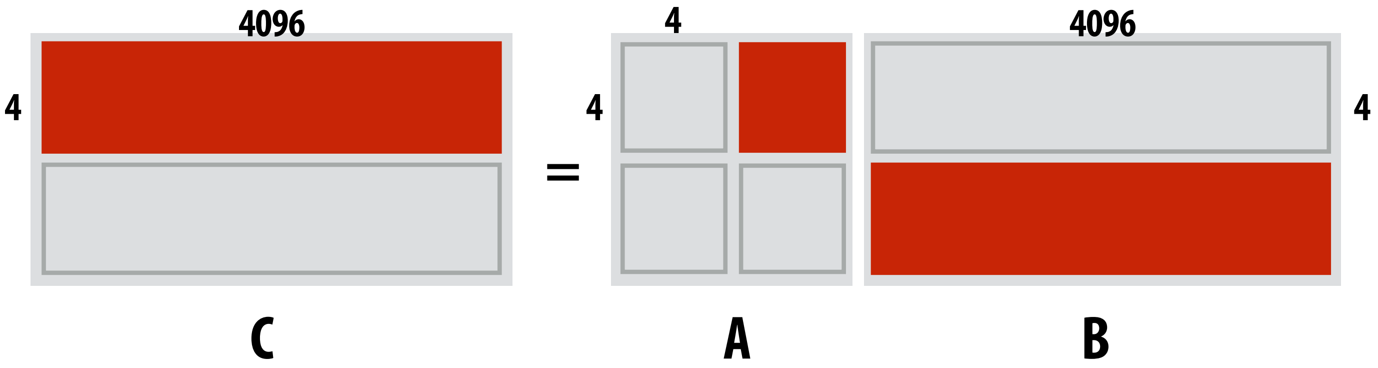
Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

Building larger matrix-matrix multiplies

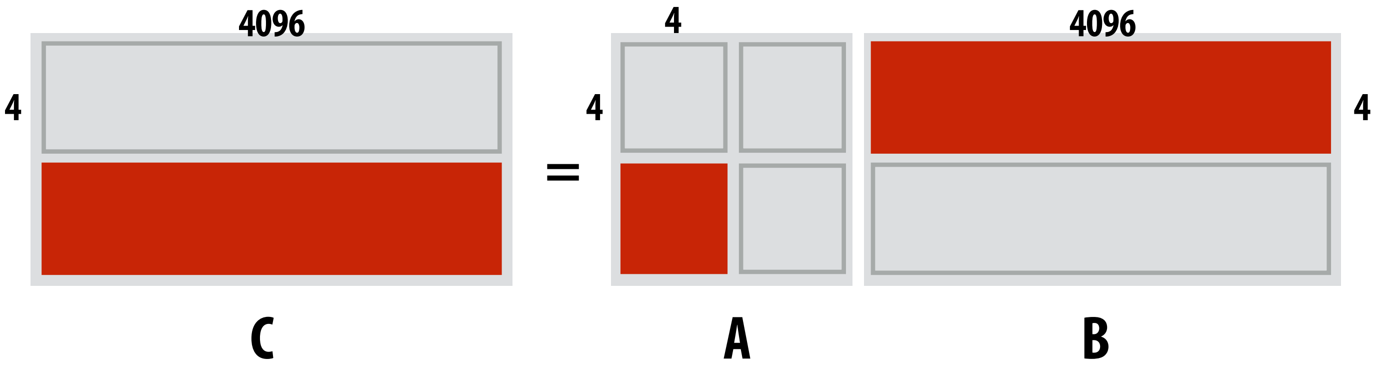
Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

Building larger matrix-matrix multiplies

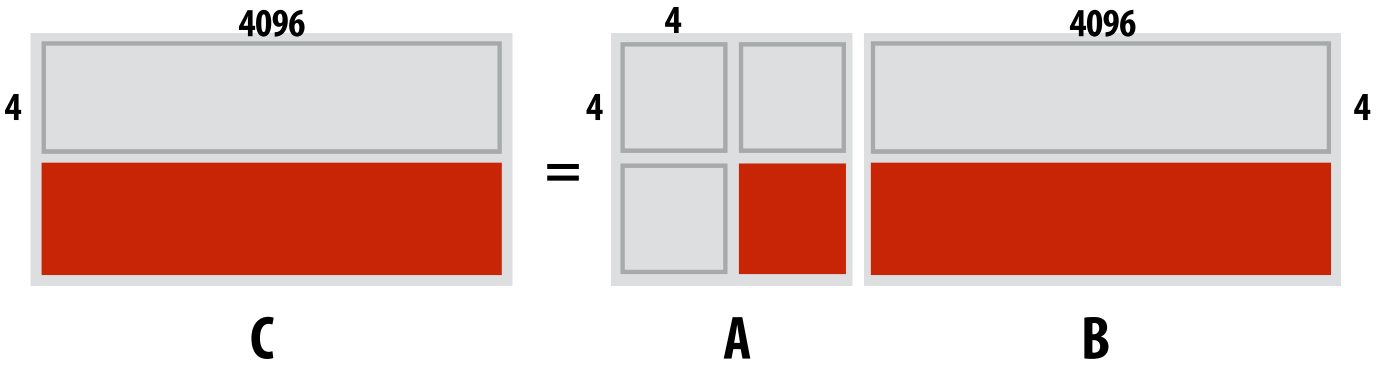
Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

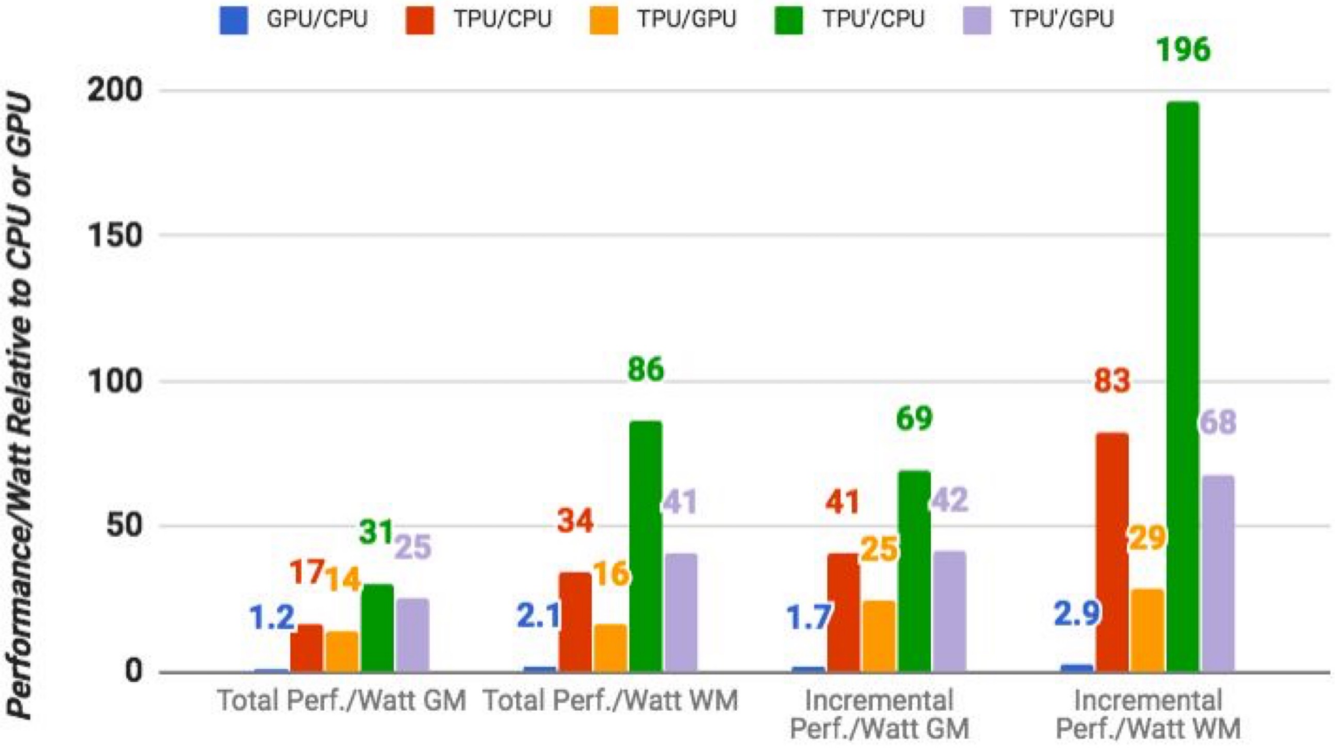
Building larger matrix-matrix multiplies

Example: A = 8x8, B= 8x4096, C=8x4096



Assume 4096 accumulators

TPU Performance/Watt

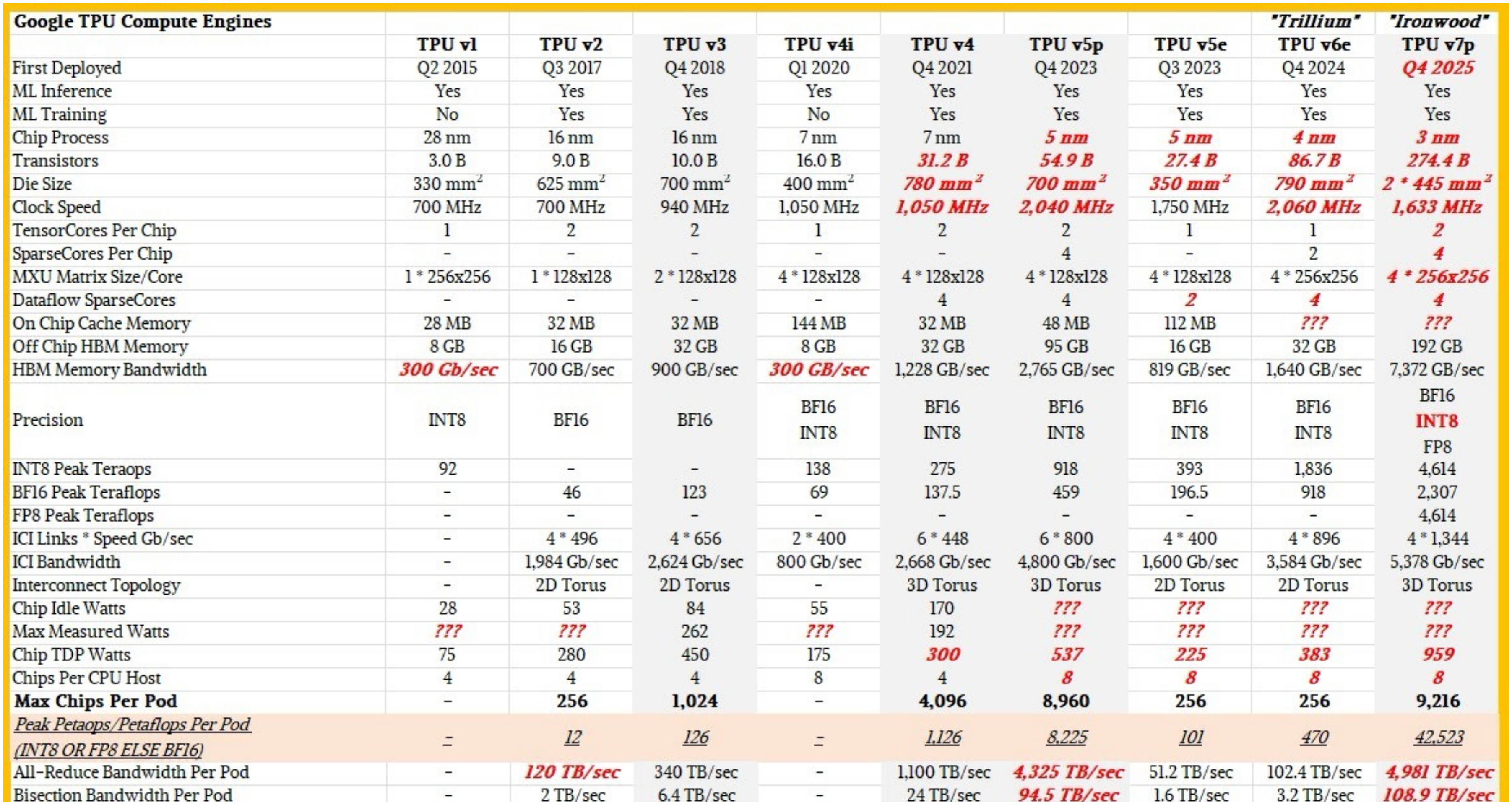


GM = geometric mean over all apps
WM = weighted mean over all apps

total = cost of host machine + CPU
incremental = only cost of TPU

Figure credit: Jouppi et al. 2017

Evolution of Google TPUs



Google TPU Compute Engines								"Trillium"	"Ironwood"
	TPU v1	**TPU v2**	**TPU v3**	**TPU v4i**	**TPU v4**	**TPU v5p**	**TPU v5e**	**TPU v6e**	**TPU v7p**
First Deployed	Q2 2015	Q3 2017	Q4 2018	Q1 2020	Q4 2021	Q4 2023	Q3 2023	Q4 2024	***Q4 2025***
ML Inference	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ML Training	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Chip Process	28 nm	16 nm	16 nm	7 nm	7 nm	***5 nm***	***5 nm***	***4 nm***	***3 nm***
Transistors	3.0 B	9.0 B	10.0 B	16.0 B	***31.2 B***	***54.9 B***	***27.4 B***	***86.7 B***	***274.4 B***
Die Size	330 mm^2	625 mm^2	700 mm^2	400 mm^2	***780 mm^2***	***700 mm^2***	***350 mm^2***	***790 mm^2***	***2 * 445 mm^2***
Clock Speed	700 MHz	700 MHz	940 MHz	1,050 MHz	***1,050 MHz***	***2,040 MHz***	1,750 MHz	***2,060 MHz***	***1,633 MHz***
TensorCores Per Chip	1	2	2	1	2	2	1	1	***2***
SparseCores Per Chip	-	-	-	-	-	4	-	2	***4***
MXU Matrix Size/Core	1 * 256x256	1 * 128x128	2 * 128x128	4 * 128x128	4 * 128x128	4 * 128x128	4 * 128x128	4 * 256x256	***4 * 256x256***
Dataflow SparseCores	-	-	-	-	4	4	***2***	***4***	***4***
On Chip Cache Memory	28 MB	32 MB	32 MB	144 MB	32 MB	48 MB	112 MB	***???***	***???***
Off Chip HBM Memory	8 GB	16 GB	32 GB	8 GB	32 GB	95 GB	16 GB	32 GB	192 GB
HBM Memory Bandwidth	***300 Gb/sec***	700 GB/sec	900 GB/sec	***300 GB/sec***	1,228 GB/sec	2,765 GB/sec	819 GB/sec	1,640 GB/sec	7,372 GB/sec
Precision	INT8	BF16	BF16	BF16 INT8	BF16 INT8	BF16 INT8	BF16 INT8	BF16 INT8	BF16 **INT8** FP8
INT8 Peak Teraops	92	-	-	138	275	918	393	1,836	4,614
BF16 Peak Teraflops	-	46	123	69	137.5	459	196.5	918	2,307
FP8 Peak Teraflops	-	-	-	-	-	-	-	-	4,614
ICI Links * Speed Gb/sec	-	4 * 496	4 * 656	2 * 400	6 * 448	6 * 800	4 * 400	4 * 896	4 * 1,344
ICI Bandwidth	-	1,984 Gb/sec	2,624 Gb/sec	800 Gb/sec	2,668 Gb/sec	4,800 Gb/sec	1,600 Gb/sec	3,584 Gb/sec	5,378 Gb/sec
Interconnect Topology	-	2D Torus	2D Torus	-	3D Torus	3D Torus	2D Torus	2D Torus	3D Torus
Chip Idle Watts	28	53	84	55	170	***???***	***???***	***???***	***???***
Max Measured Watts	***???***	***???***	262	***???***	192	***???***	***???***	***???***	***???***
Chip TDP Watts	75	280	450	175	***300***	***537***	***225***	***383***	***959***
Chips Per CPU Host	4	4	4	8	4	***8***	***8***	***8***	***8***
Max Chips Per Pod	-	**256**	**1,024**	-	**4,096**	**8,960**	**256**	**256**	**9,216**
Peak Petaops/Petaflops Per Pod (INT8 OR FP8 ELSE BF16)	*-*	*12*	*126*	*-*	*1,126*	*8,225*	*101*	*470*	*42,523*
All-Reduce Bandwidth Per Pod	-	***120 TB/sec***	340 TB/sec	-	1,100 TB/sec	***4,325 TB/sec***	51.2 TB/sec	102.4 TB/sec	***4,981 TB/sec***
Bisection Bandwidth Per Pod	-	2 TB/sec	6.4 TB/sec	-	24 TB/sec	***94.5 TB/sec***	1.6 TB/sec	3.2 TB/sec	***108.9 TB/sec***

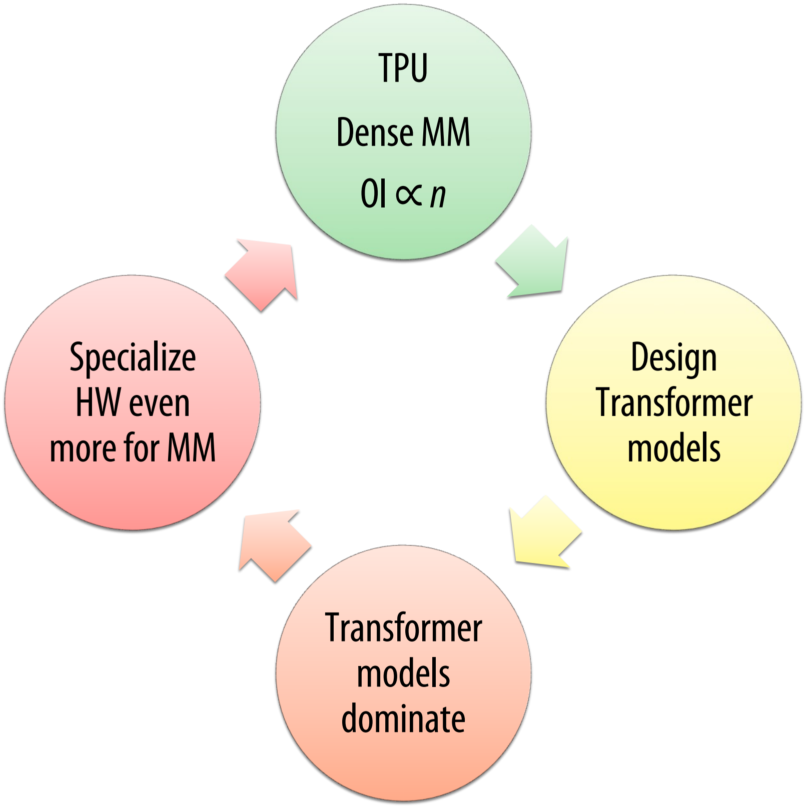
Source: The Next Platform

Hardware Lottery

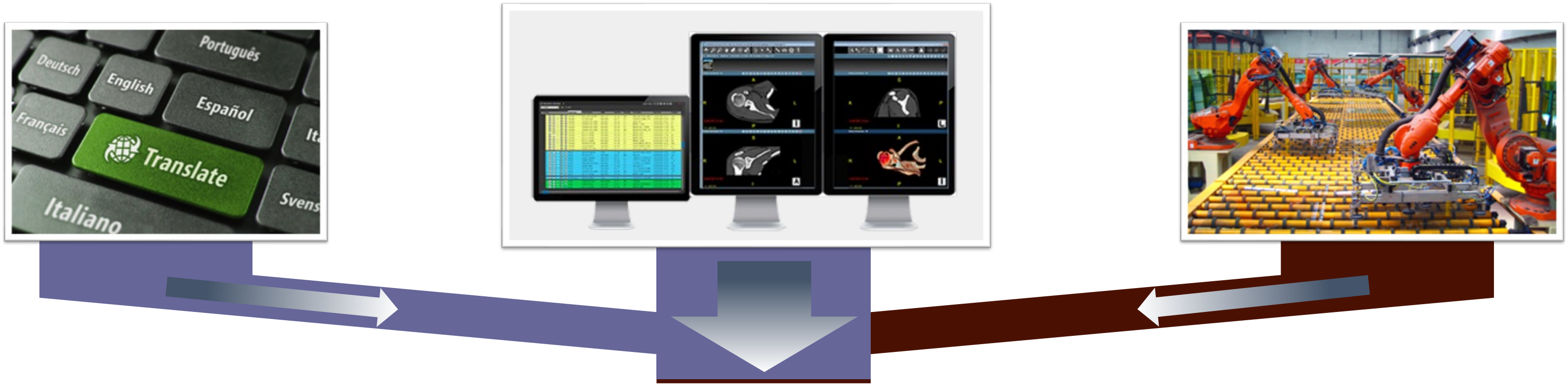
COMMUNICATIONS OF THE ACM
CACM.ACM.ORG
12/2021 VOL.64 NO.12
THE HARDWARE LOTTERY
$$$
Datasheets for Datasets • Q&A with Scott Aaronson
Digital Agriculture • Speculative Taint Tracking
Association for Computing Machinery
acm

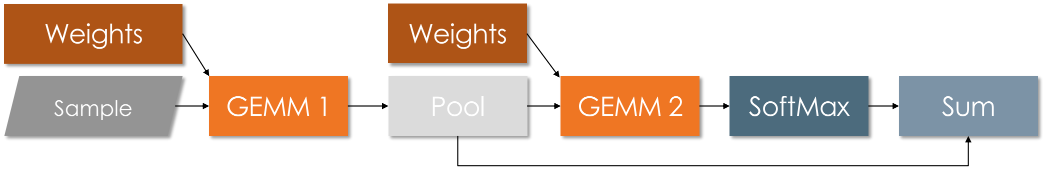
When a research idea wins because it is suited to the available software and hardware and not because the idea is universally superior to alternative research directions.

Sara Hooker

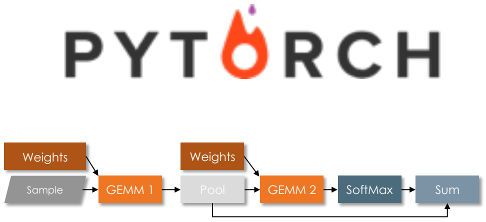

TPU
Dense MM
OI ∝ n
Design Transformer models
Transformer models dominate
Specialize HW even more for MM

Recall: AI Models are Dataflow Graphs

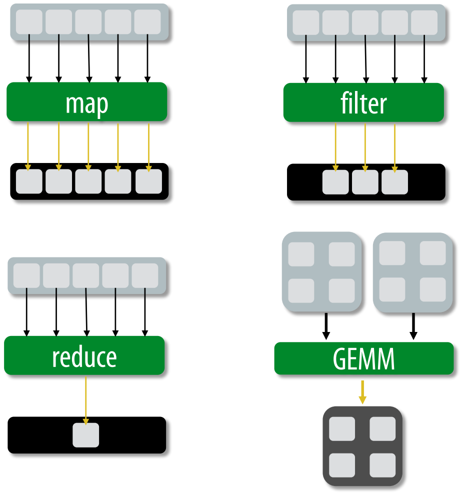




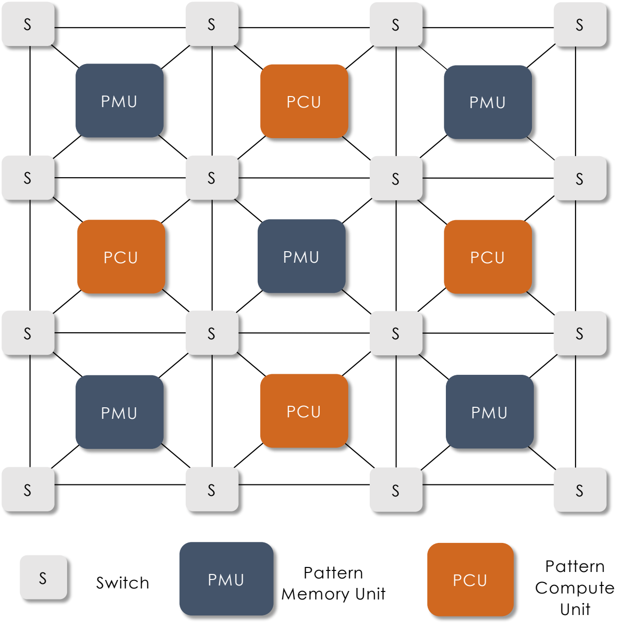
AI Models ⇒ Dataflow Architecture



AI Models



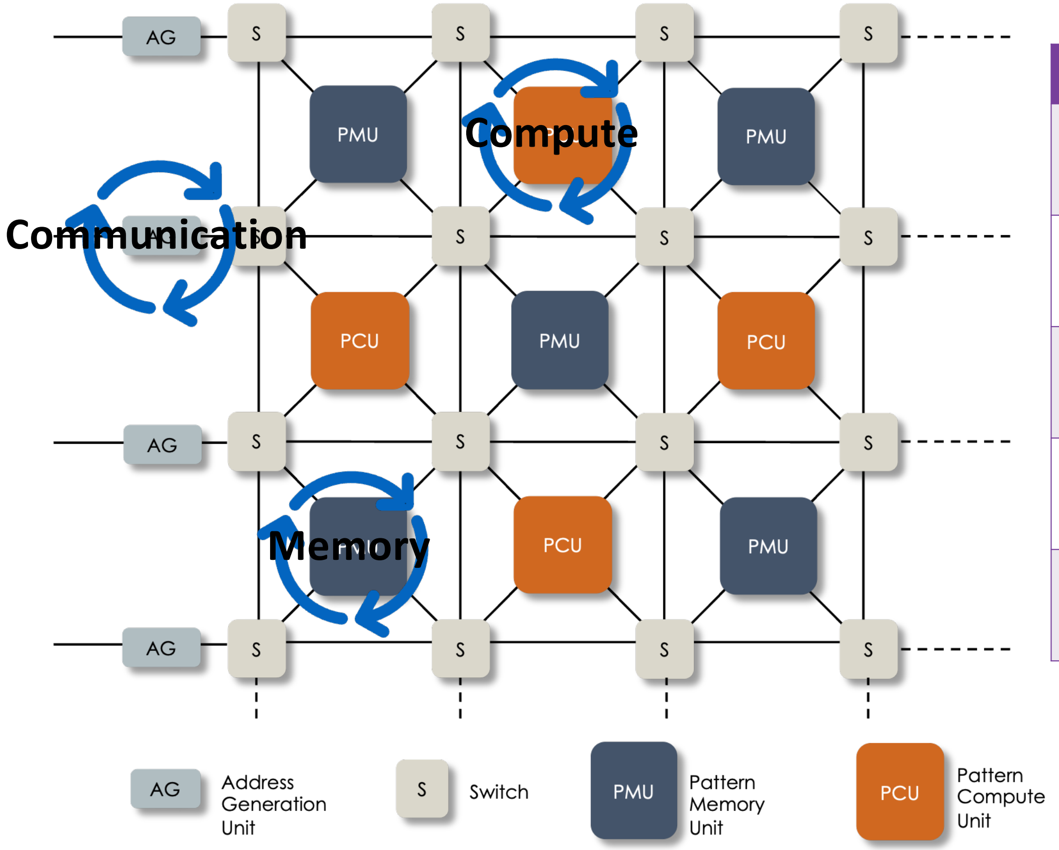
Dataflow graph:
GEMM + Parallel Patterns



Plasticine
Reconfigurable Dataflow Architecture

Prabhakar, Zhang, et. al. ISCA 2017

Reconfigurable Dataflow Architecture vs Ideal Accelerator



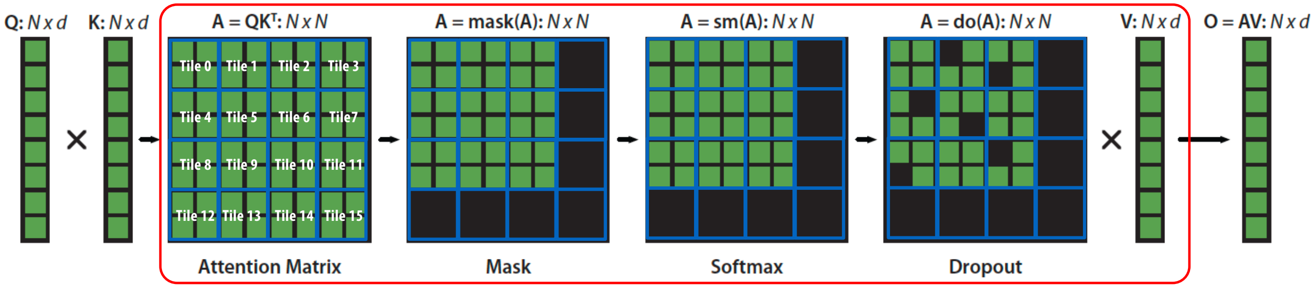
Feature	Why?
Tiled tensors (e.g. 16 x 16, 32 x 32)	Max TFLOPS on GEMM Low instr. overhead
Asynchronous compute	Overlap compute and memory access
Asynchronous memory access	Overlap compute and memory access
Asynchronous chip-to-chip communication	Overlap compute, memory and communication
Compute unit to compute unit comm.	Fusion and pipelining Streaming Dataflow

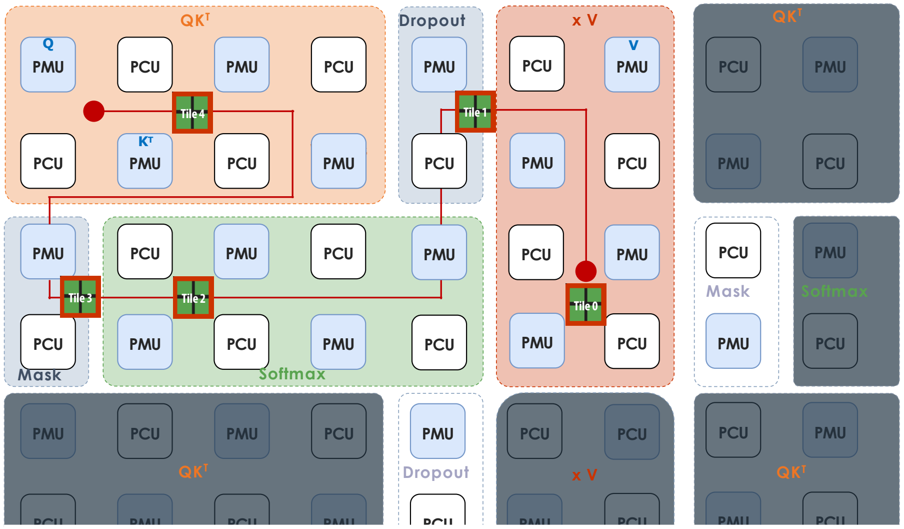
No instructions ⇒ No instruction fetch/decode overhead

Extreme asynchrony: no sequential instruction execution

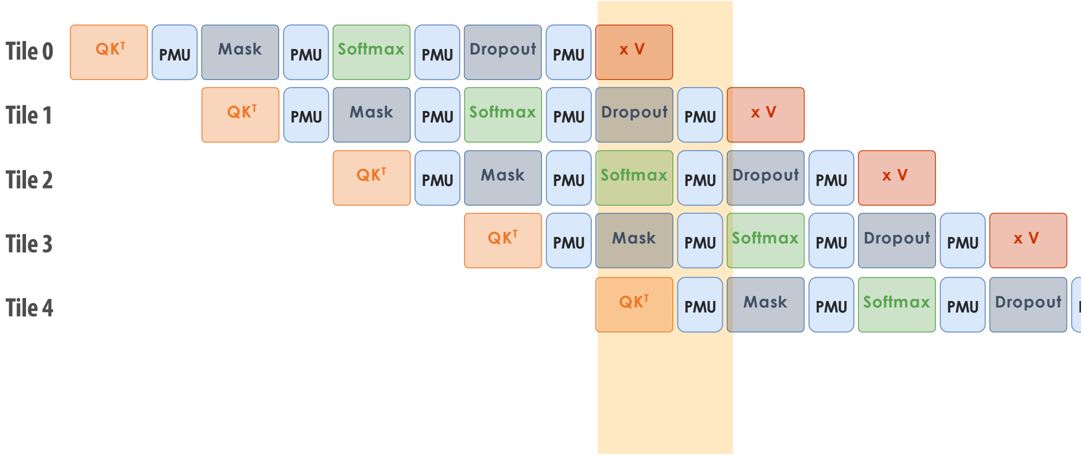
Dataflow Kernel Fusion

FlashAttention





Dataflow execution



MetaPipeline

Summary: specialized hardware for AI model processing

Specialized hardware for executing key DNN computations efficiently

Feature many arithmetic units

Customized/configurable datapaths to directly move intermediate data values between processing units (schedule computation by laying it out spatially on the chip) at multiple granularities

–

Large amounts of on-chip storage for fast access to intermediates