Parallel Computing & Parallel Hardware

Thoai Nam

High Performance Computing Lab (HPC Lab)

Faculty of Computer Science and Engineering

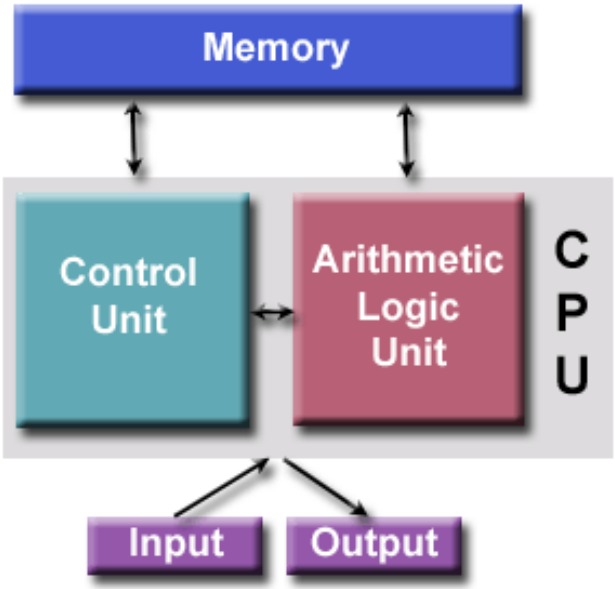
HCMC University of Technology

Outline

- Concepts and Terminology
  - A review on von Neumann Arch
  - Instructions & Clock cycle
- Multicore Processor, Parallelism & Performance
- Parallelism Levels
  - Instruction-Level, Data-Level, Thread-Level Parallelism
- From Data-Parallelism to Vector Processor, Graphic Processing Unit (GPU)
- Domain Specific Architectures

von Neumann Architecture

- Known as "*stored-program computer*" - both **program instructions** and **data** are kept in electronic memory.
- Four main components:
  - Control unit fetches instructions/data from memory, decodes the instructions and then sequentially coordinates operations to accomplish the programmed task.
  - Arithmetic Unit performs basic arithmetic operations.
  - Input/Output is the interface to the human operator.



https://hpc.llnl.gov/training/tutorials/

von Neumann Architecture

How computer executes a program?

```cpp
#include <iostream>
using namespace std;

int main()
{
    int firstNumber, secondNumber, sumOfTwoNumbers;

    cout << "Enter two integers: ";
    cin >> firstNumber >> secondNumber;

    // sum of two numbers in stored in variable sumOfTwoNumbers
    sumOfTwoNumbers = firstNumber + secondNumber;

    // Prints sum
    cout << firstNumber << " + " <<  secondNumber << " = " << sumOfTwoNumbers;

    return 0;
}
```

prog_instruction 1: **Start** the program

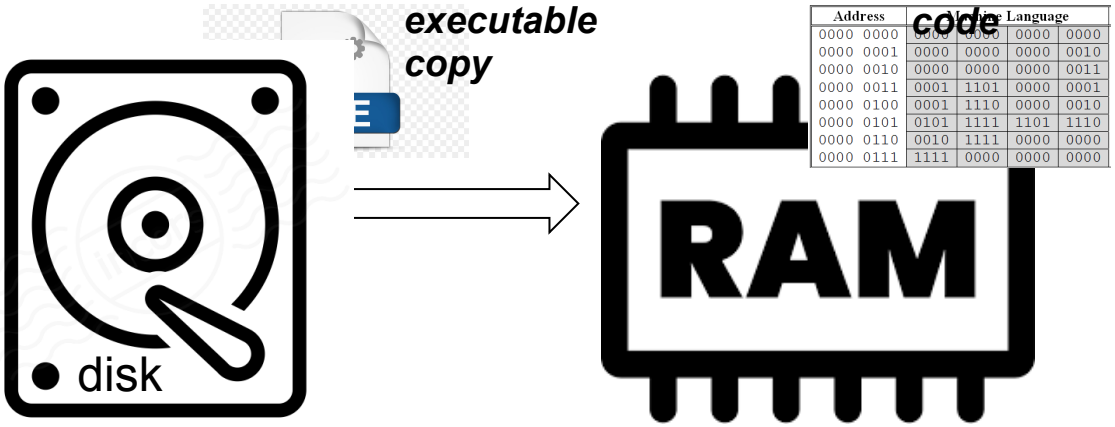
prog_instruction 2: declare **Variables**

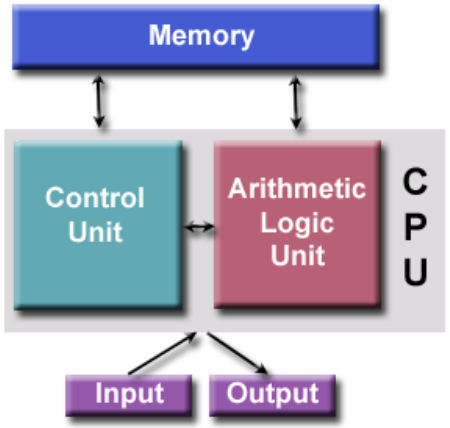
prog_instruction 3: **IO** display

…

prog_instruction 6: **End** the program

compile the program

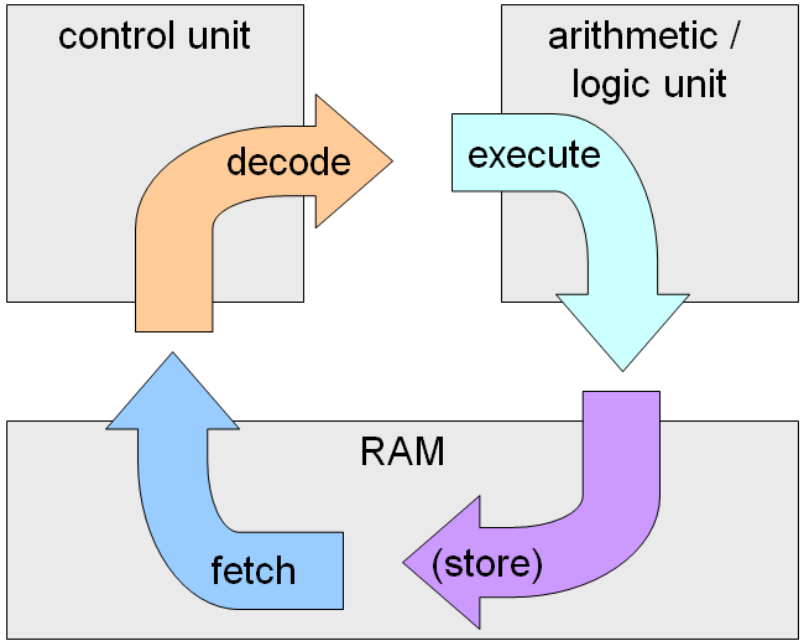




https://hpc.llnl.gov/training/tutorials/

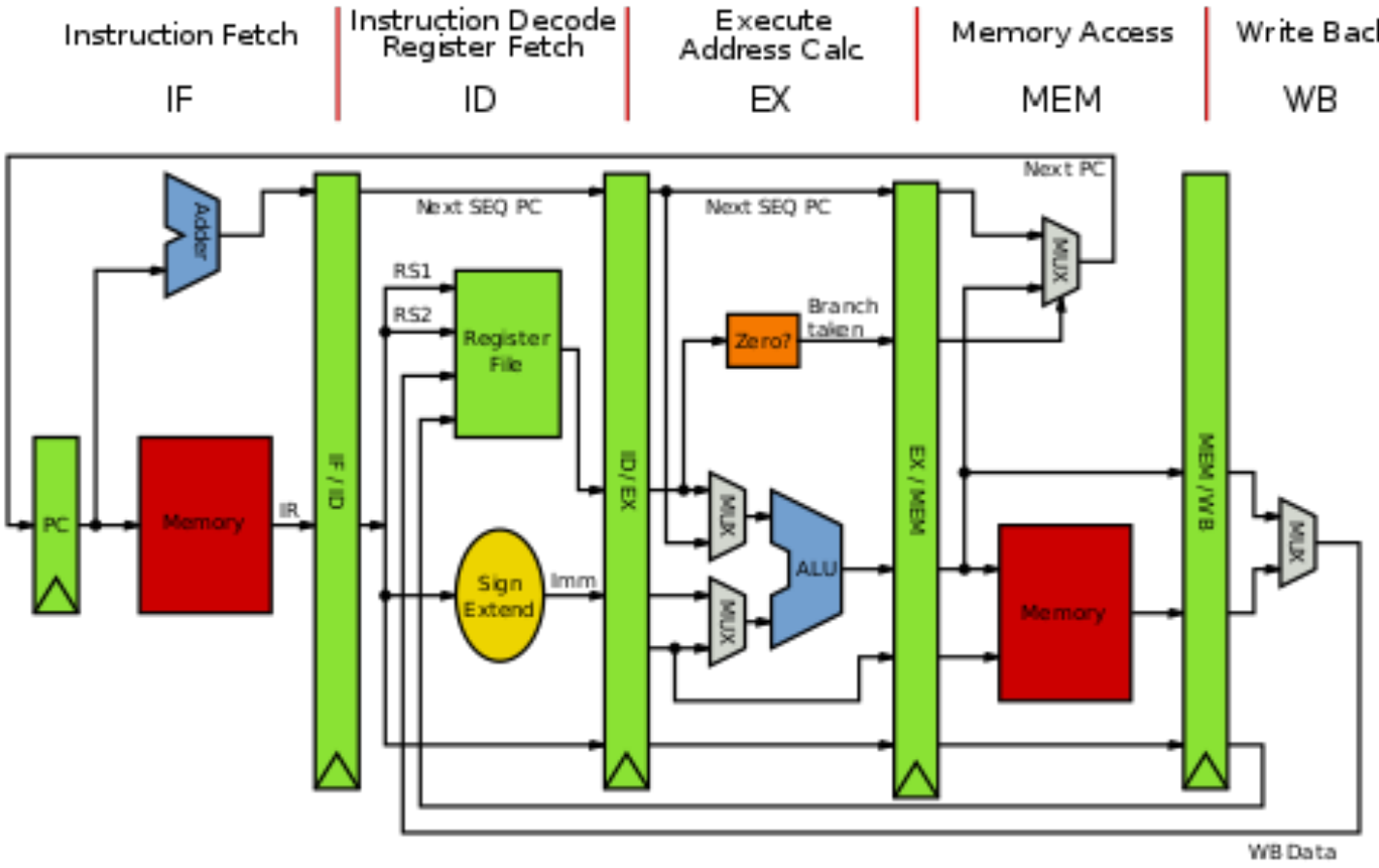
Instructions & Clock Cycle

- A program instruction could have *many* machine instructions
- A machine instruction is performed by a process including 4 fundamental steps:
  - Fetch the instruction
  - Decode it
  - Execute
  - (Store the result)



https://computersciencewiki.org/index.php/The_machine_instruction_cycle

MIPS architecture

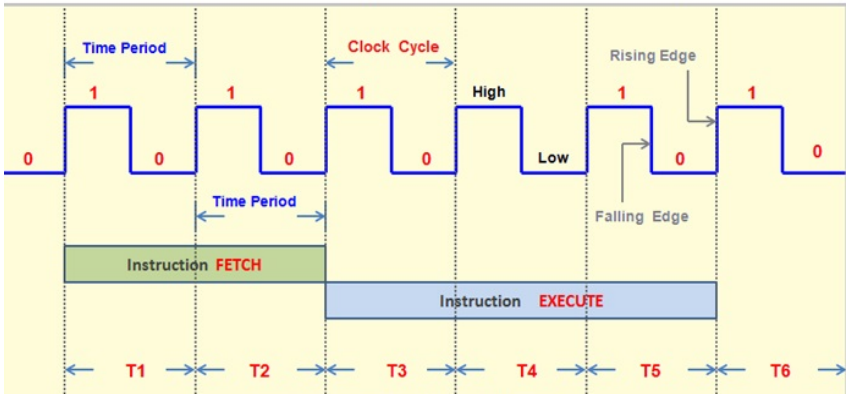


Instructions & Clock Cycle (1)

- **CPU Clock**: all computers are constructed using a clock running at a constant rate[1]
- These discrete time events are called *clock periods*, *clocks*, *cycles*, or *clock cycles.*

- The ***clock cycle time*** is the amount of time for one clock period to elapse (e.g. 1 ns).
- The ***clock rate*** is the inverse of the ***clock cycle time***.
- For example, if a CPU has 1 ns clock cycle time, its rate:

$$\frac{1}{1 * 10^{-9}(sec)} = 1\ GHz$$



https://computersciencewiki.org/index.php/The_machine_instruction_cycle

[1] Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2011.

Instructions & Clock Cycle (2)

How many cycles are required for a program?

- Different instructions take different amounts of time on different machines

Different Numbers of Cycles for Different Instructions

- For examples:
  - *Division* takes more time than *Addition*
  - *Floating point* operations take longer than *Integer* ones
  - Accessing *Memory* takes more time than accessing *Registers*

The Processor Performance

- CPU time for a program can then be expressed 2 ways:
  - $CPU\ Time = CPU\ clock\ cycles\ for\ a\ program\ *\ Clock\ cycle\ time$

  or
  - $CPU\ Time = \frac{CPU\ clock\ cycles\ for\ a\ program}{Clock\ rate}$
- In addition to the *number of clock cycles* needed to execute a program, we can also count the number of instructions executed – *instruction path length* or *instruction count* (IC).
- If we know number of clock cycles & instruction count, we can calculate average number of clock *cycles per instruction* (CPI).
  - $CPI = \frac{CPU\ clock\ cycles\ for\ a\ program}{Instruction\ count}$

The Processor Performance

- By transposing the instruction count in the preceding formula, clock cycles can be defined as IC/CPI. This allows us to use CPI in the execution time formula:

$$CPU\ Time = IC * CPI * Clock\ cycle\ time$$

- Expanding the first formula into the units of measurement shows how the pieces fit together:

$$\frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ cycle} = \frac{Seconds}{Program} = CPU\ Time$$

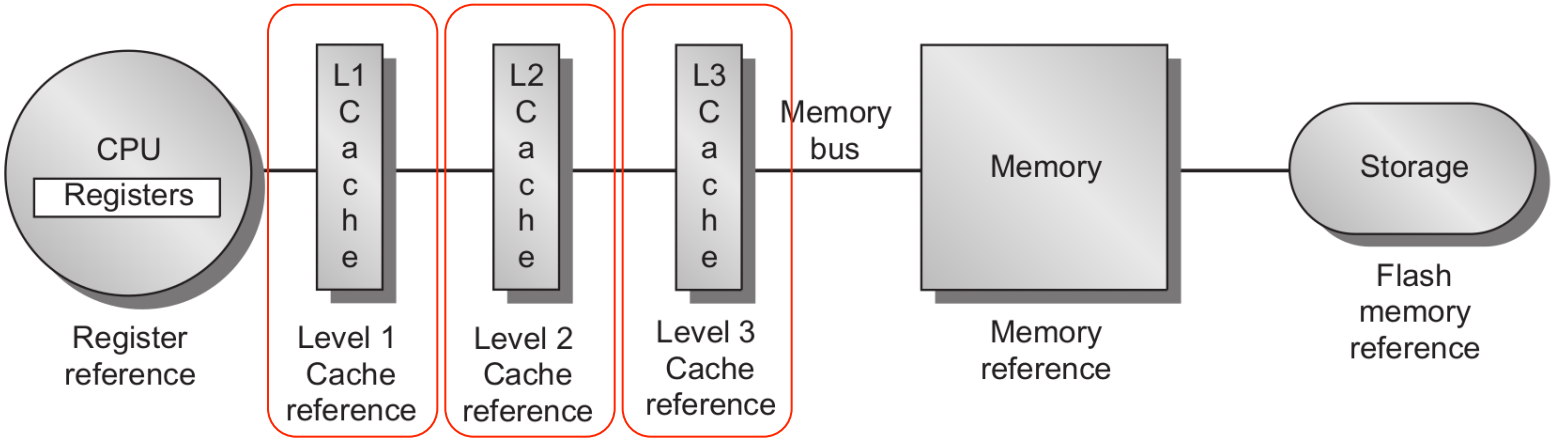
- Processor performance depends on what???

clock cycle, *CPI,* *IC*

Hardware technology	Organization & Instruction set architecture	Instruction set architecture & compiler

Memory

Regarding Memory Access Instructions…, how to improve the performance?



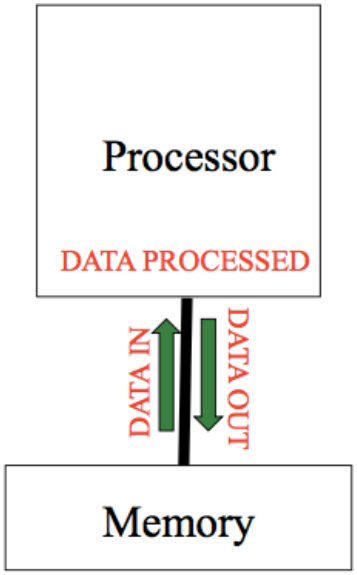
Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4–16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS

Memory hierarchy for a laptop or a desktop.

Source: Hennessy, John L., and David A. Patterson. Computer architecture: a quantitative approach. Elsevier, 2017.

4 key performance factors

- *Parallel processing*: amount of data processed at one time
- *Clock frequency (clock rate)*: processing speed on each data element
- *Memory bandwidth*: amount of data transferred at one time
- *Memory latency*: time for each data element to be transferred



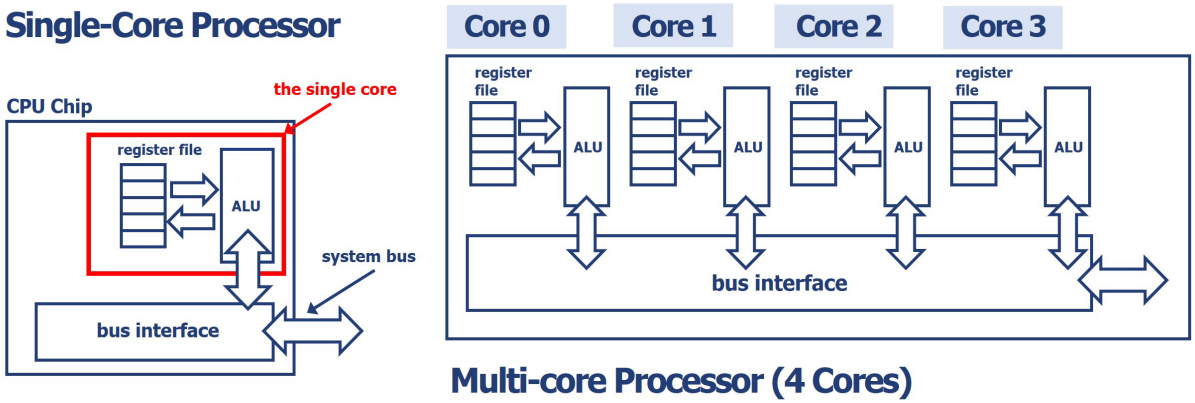
- Different computational problems are sensitive to these in different ways from one another
- Different architectures address these factors in different ways

CPUs: 4 key performance factors

- Parallel processing
  - Before each CPU only had a single core. Now CPUs have multiple cores, where each can process multiple instructions per cycle
- Clock frequency
  - CPUs aim to maximize clock frequency, but this has now hit a limit due to power restrictions
- Memory bandwidth
  - CPUs use regular DDR memory, which has limited bandwidth
- Memory latency
  - Latency from DDR is high, but CPUs strive to hide the latency through:
    - » Large on-chip low-latency caches to stage data
    - » Multithreading
    - » Out-of-order execution

Multicore Processing & Hardware

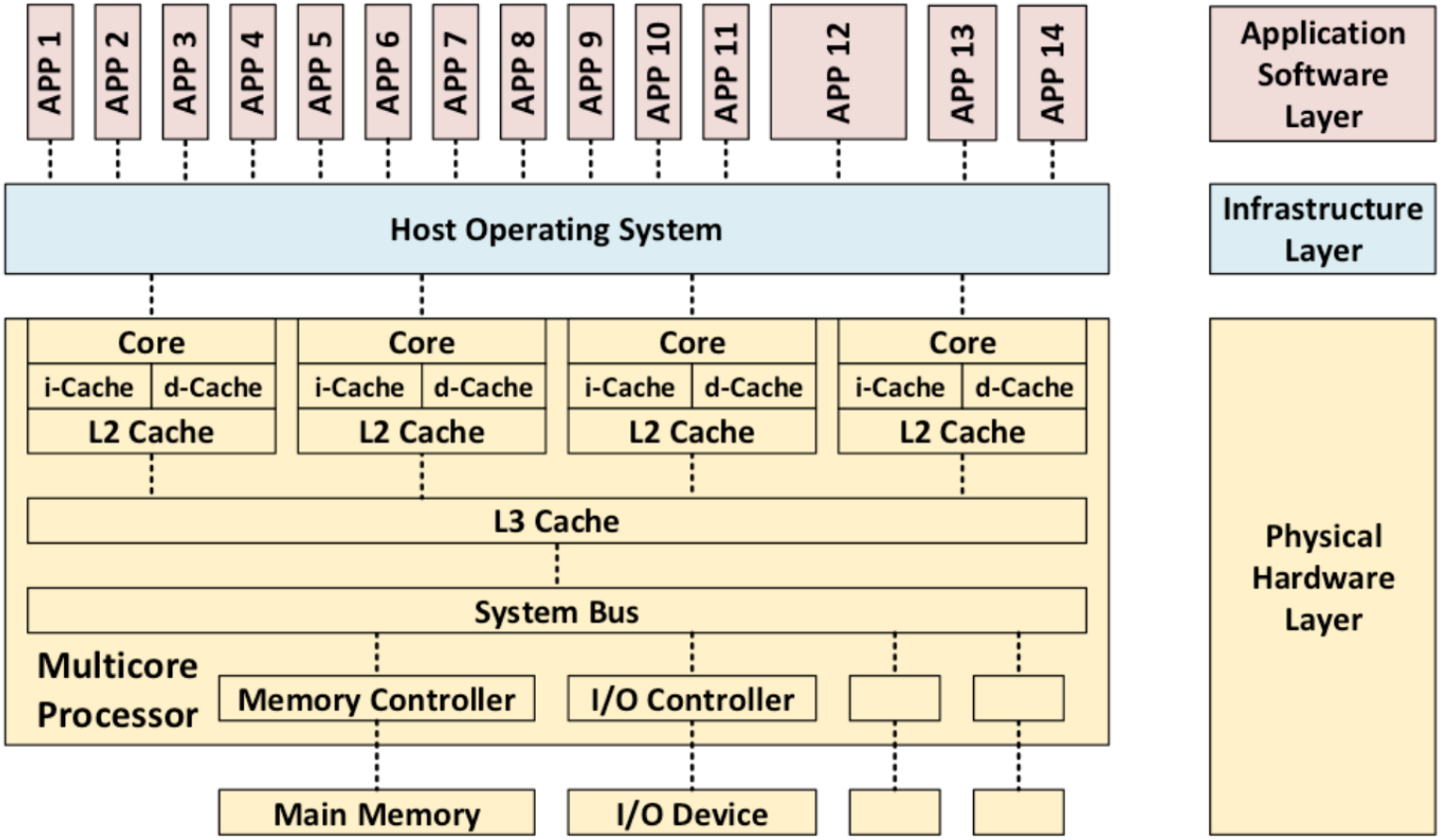
- Instead of having only 1 single processing unit/CPU, there could be multiple processing units, more commonly known as ***cores***
- There are many different multicore processor architectures, which vary in terms of:
  - # cores
  - Core types
  - # level of caches
  - How cores are interconnected



https://www.lynx.com/embedded-systems-learning-center/cast-32a-significance-and-implications

Multicore Processing & Hardware

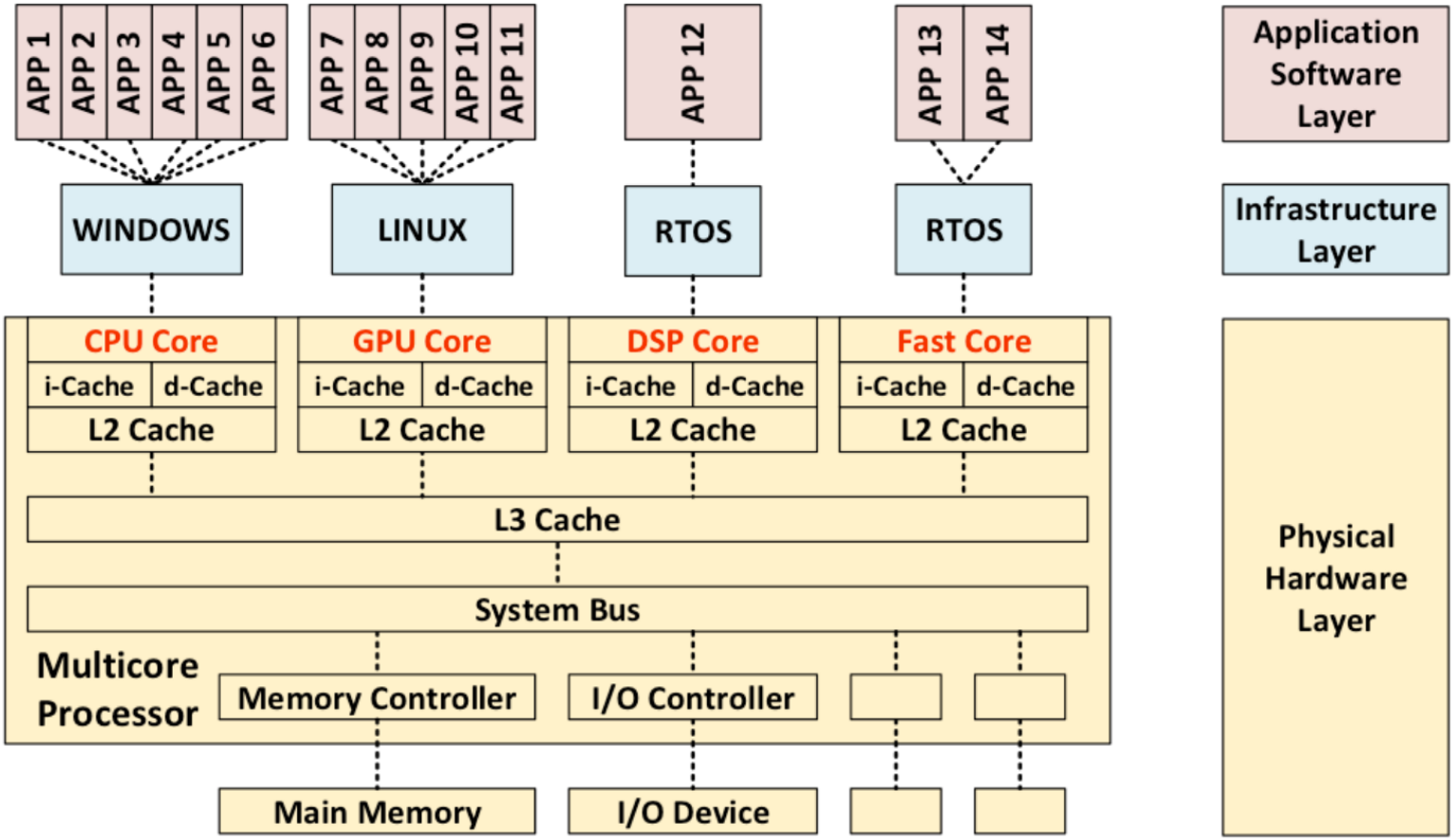
Homogeneous Multicore Processor



https://insights.sei.cmu.edu/sei_blog/2017/08/multicore-processing.html

Multicore Processing & Hardware

Heterogeneous Multicore Processor

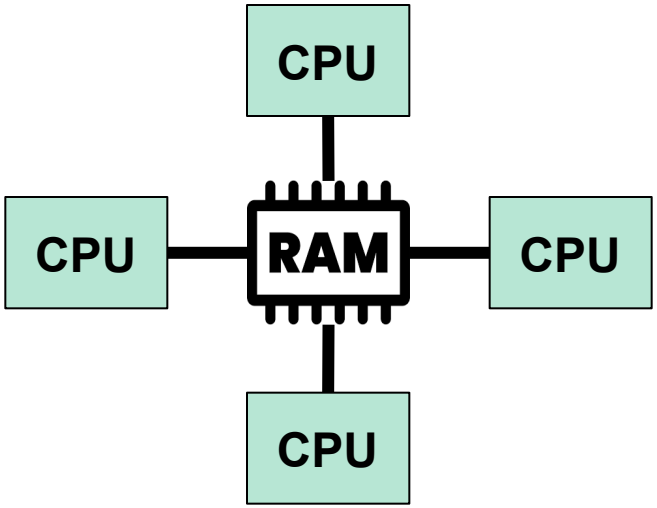


https://insights.sei.cmu.edu/sei_blog/2017/08/multicore-processing.html

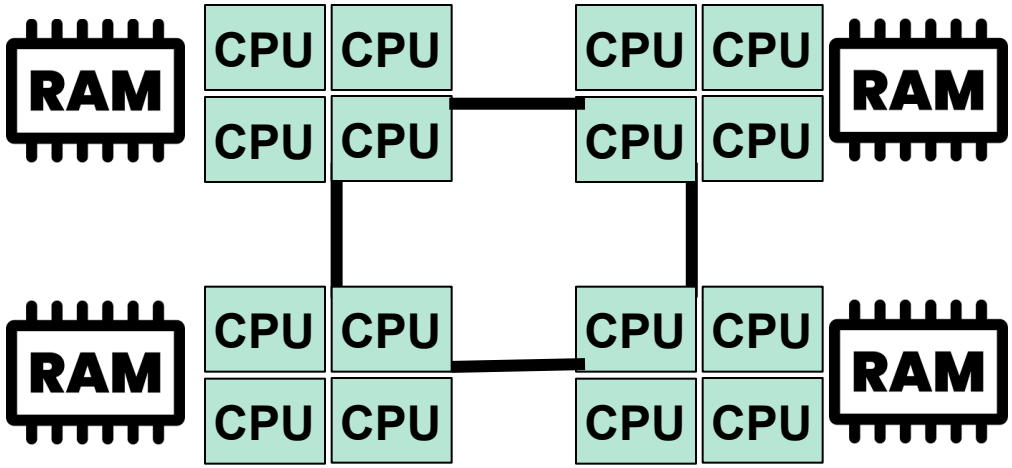
Parallel Computer???

… Multiple Processors (CPU) in a computer???

- *Then, how are they connected?*
- *How about memories?*



Uniform Memory Access (UMA)



Non-Uniform Memory Access (NUMA)

Simplified by Memory Architectures

Parallelism levels

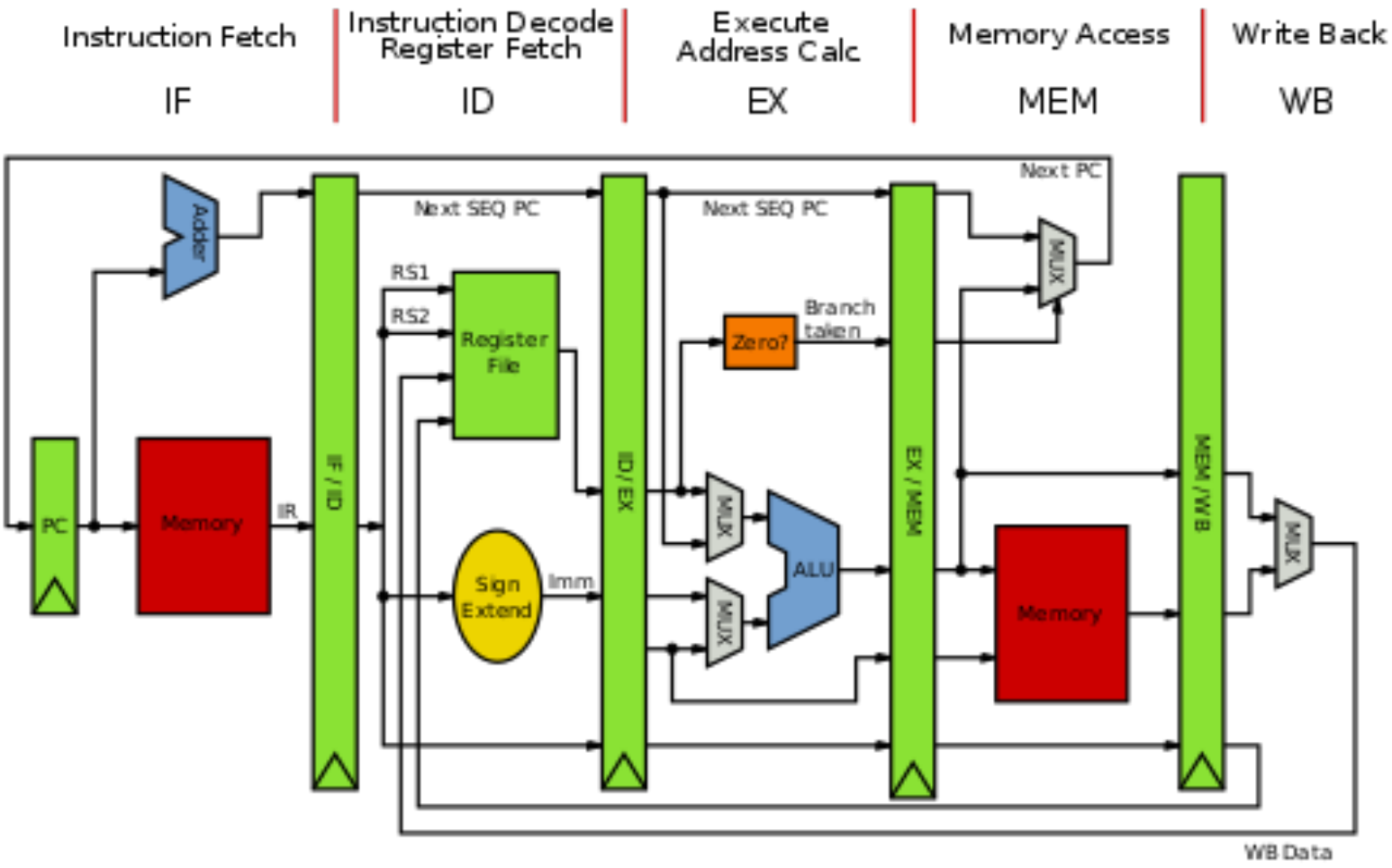
- Instruction-Level Parallelism (ILP)
- Data-Level Parallelism (DLP)
- Thread-Level Parallelism (TLP)

Instruction-Level Parallelism (ILP)

Instruction-Level Parallelism (ILP)

- ILP is a measure of how many of the instructions in a computer program can be executed simultaneously
- Micro-architectural techniques
  - Instruction pipelining where the execution of multiple instructions can be partially overlapped
  - Superscalar execution where multiple execution units are used to execute multiple instructions in parallel
    - Ex: 1 CPU (IBM Power PC970) = 4 ALU + 2 FPU + 2 SIMD units
  - Out-of-order execution where instructions execute in any order that does not violate data dependencies: both pipelining and superscalar
  - Register renaming, which is a technique that abstracts logical registers from physical registers, used to avoid unnecessary serialization of program operations imposed by the reuse of registers by those operations, used to enable out-of-order execution
  - Speculative execution which allows the execution of complete instructions or parts of instructions before being certain whether this execution should take place
  - Branch prediction which is used to avoid stalling for control dependencies to be resolved

Pipeline in MIPS



Data-Level Parallelism (DLP)

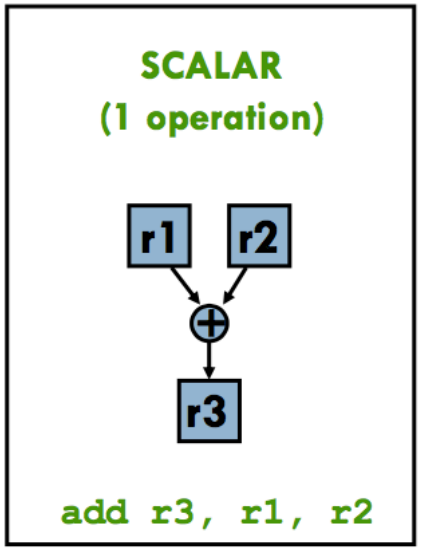
Data-Level Parallelism (DLP)

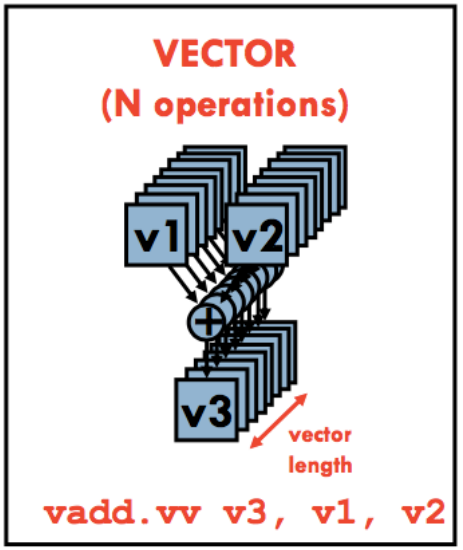
- Data Parallelism: parallelism arises from executing essentially the same code on a large number of objects
- DLP originally linked with SIMD machines
- DLP architecture
  - *Vector processor*
  - *SIMD extensions*
  - MPP
  - *Modern GPU: Nvidia, AMD, Qualcomm, etc.*
- Focus of throughput rather than latency

Vector processor

- Scalar processors operate on single numbers (scalars)
- Vector processors operate on linear sequences of numbers (vectors)
- Many scientific/commercial programs use vector

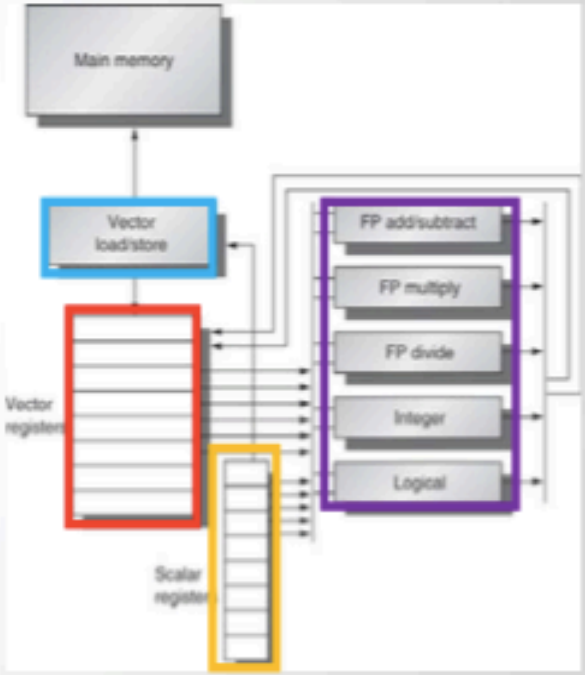
```
for (i=0; i< 1000; i++) // DAXPY
  Y[i] = a * X[i] + Y[i];
```





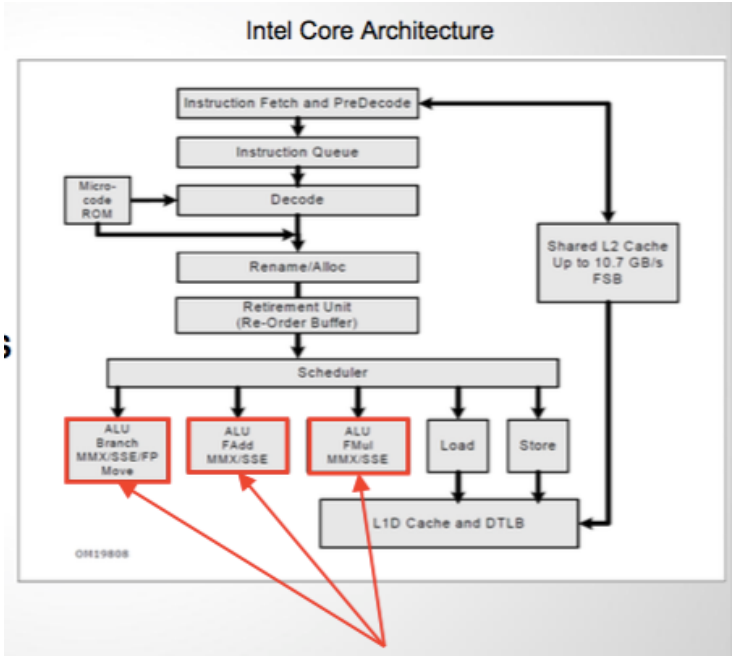
Components of Vector Processors

- *Vector Registers*
  - Typically 8-32 vector registers with 64 - 128 64-bit elements
  - Each contains a vector of double-precision numbers
  - Register size determines the maximum vector length
  - Each includes at least 2 read and 1 write ports
- *Vector Functional Units (FUs)*
  - Fully pipelined, new operation every cycle
  - Performs arithmetic and logic operations
  - Typically 4-8 different units
- *Vector Load-Store Units (LSUs)*
  - Moves vectors between memory and registers
- *Scalar Registers*
  - Single elements for interconnecting FUs, LSUs, and registers

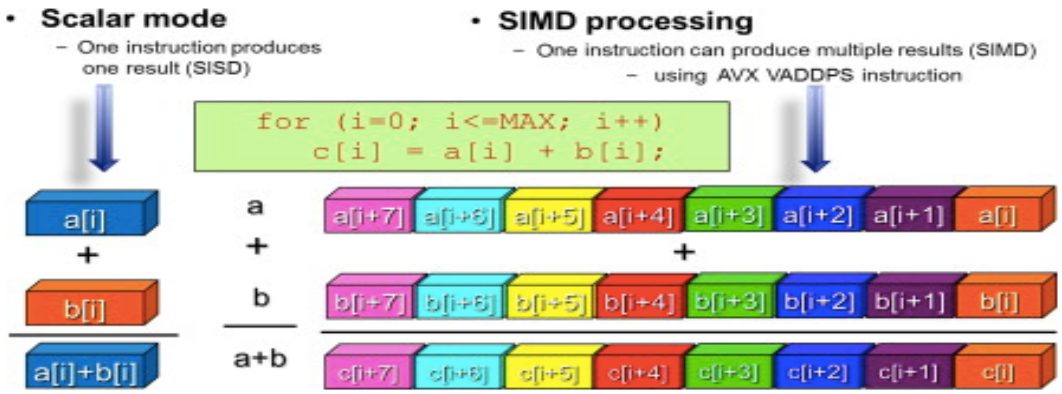


Intel SIMD Extensions (1)

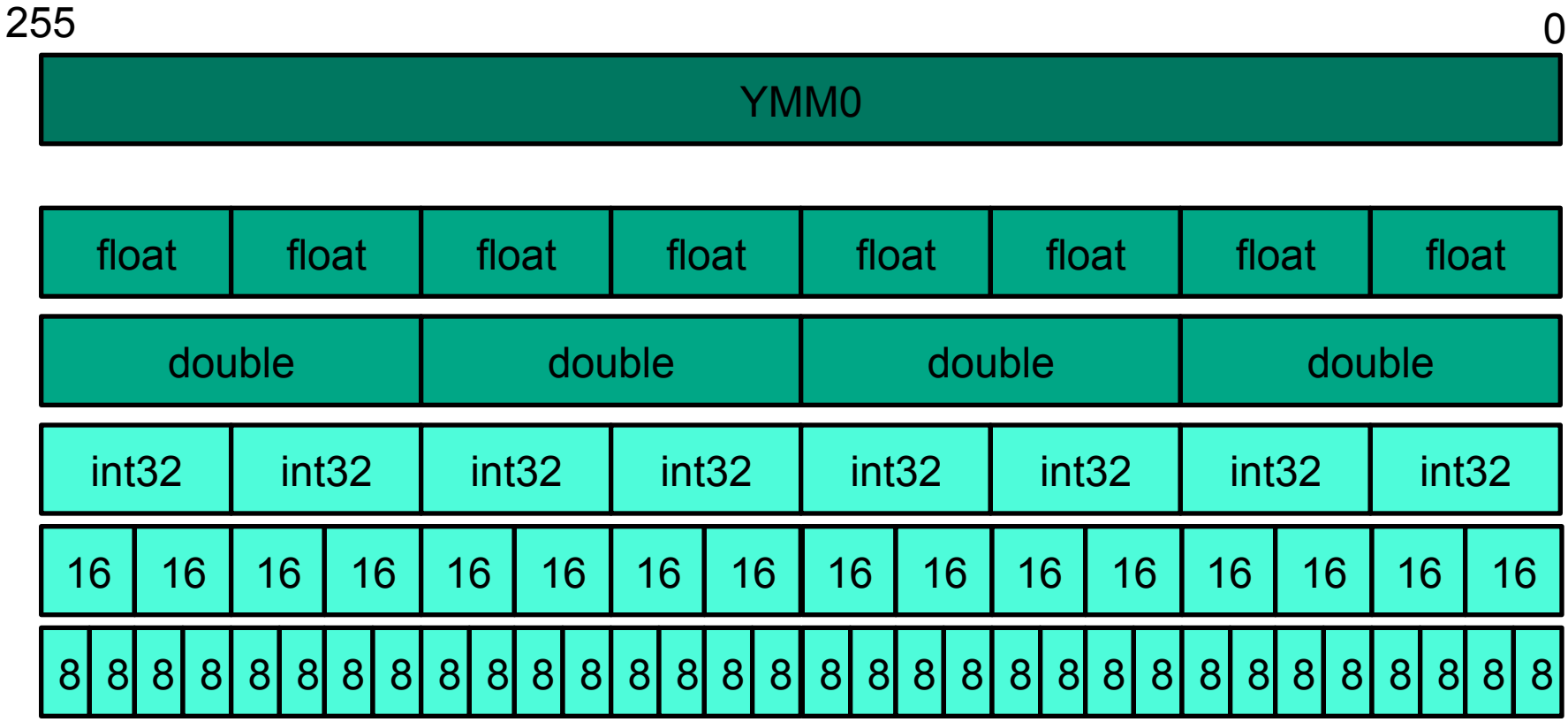
- SSE (Streaming SIMD Extensions)
- Introduced in phases/groups of functionality
  - SSE – SSE4 (1999 –2006)
    - » 128 bit width operations
  - AVX, FMA, AVX2, AVX-512 (2008 – 2015)
    - » 256 – 512 bit width operations



Combined SSE Functional Units



Intel SIMD Extensions (2)

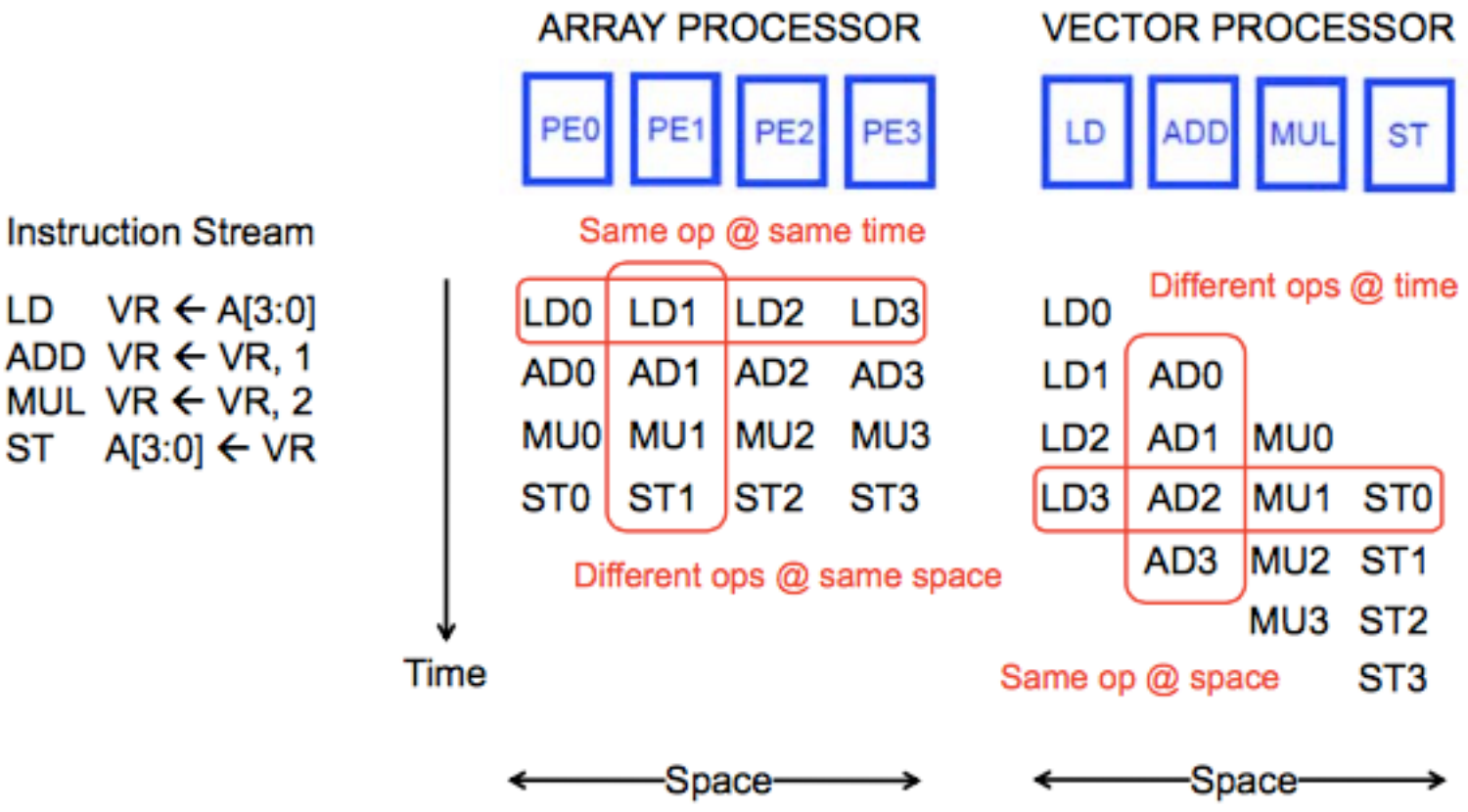


SIMD processing

- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements

- Time-space duality
  - Array processor: Instruction operates on multiple data elements at the same time
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps

Array vs. vector processors



Power problem

- The power used by a CPU core is proportional to

  Clock Frequency x Voltage^2

- In the past, computers got faster by increasing the frequency
  - Voltage was decreased to keep power reasonable.
- Now, voltage cannot be decreased any further
  - 1s and 0s in a system are represented by different voltages
  - Reducing overall voltage further would reduce this difference to a point where 0s and 1s cannot be properly distinguished

The problem with CPUs

- Instead, performance increases can be achieved through exploiting parallelism
- Need a chip which can perform many parallel operations every clock cycle
  - Many cores and/or many operations per core
- Want to keep power/core as low as possible
- Much of the power expended by CPU cores is on functionality not generally that useful for HPC
  - e.g. branch prediction

Accelerators (1)

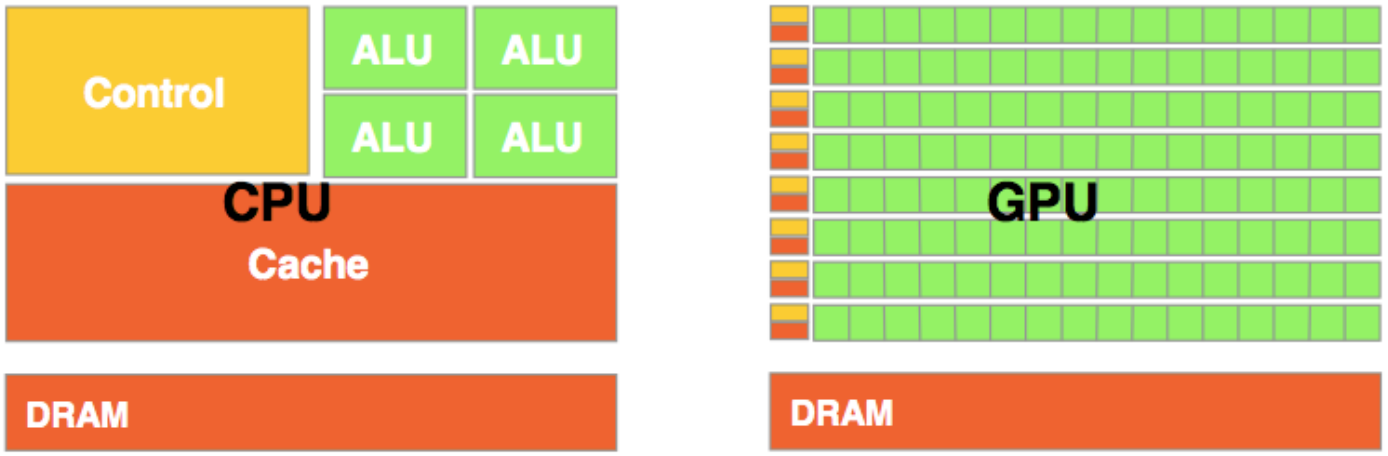
- So, for HPC, we want chips with simple, low power, number-crunching cores
- But we need our machine to do other things as well as the number crunching
  - Run an operating system, perform I/O, set up calculation etc.
- Solution: “Hybrid” system containing both CPU and “accelerator” chips

Accelerators (2)

- It costs a huge amount of money to design and fabricate new chips
  - Not feasible for relatively small HPC market
- Luckily, over the last few years, Graphics
- Processing Units (GPUs) have evolved for the highly lucrative gaming market
  - And largely possess the right characteristics for HPC – Many number-crunching cores
- GPU vendors NVIDIA and AMD have tailored existing GPU architectures to the HPC market
- GPUs now firmly established in HPC industry

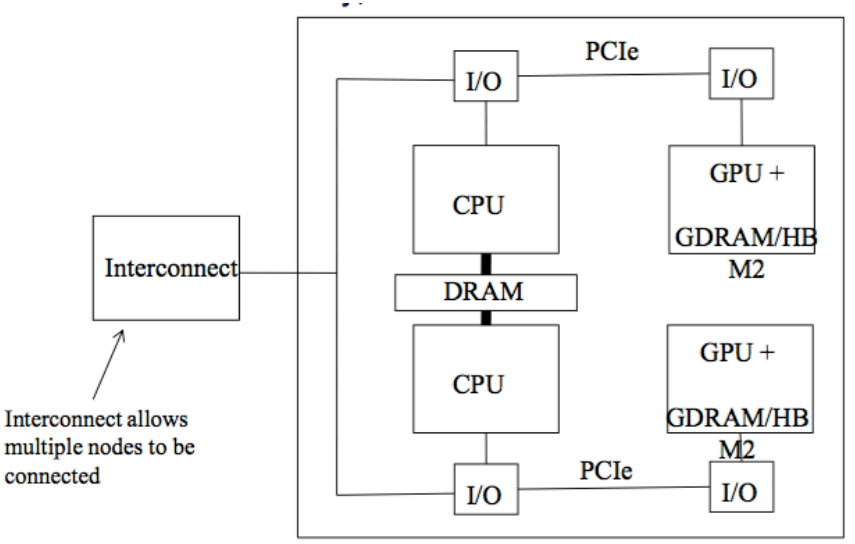
GPU

- An architecture for compute-intensive, highly data- parallel computation
  - Exactly what graphics rendering is about
  - Transistors devoted to data processing rather than caching and flow control



GPU: 4 key factors

- Parallel processing
  - GPUs have a much higher extent of parallelism than CPUs: many more cores (high-end GPUs have thousands of cores)
- Clock frequency
  - GPUs typically have lower clock-frequency than CPUs, and instead get performance through parallelism
- Memory bandwidth
  - GPUs use high bandwidth GDDR or HBM2 memory
- Memory latency
  - Memory latency from is similar to DDR
  - GPUs hide latency through very high levels of multithreading



Intel Xeon Phi

- More recently, Intel have released a different type of accelerator to compete with GPUs for scientific computing
  - Many Integrated Core (MIC) architecture
  - AKA Xeon Phi (codenames Larrabee, Knights Ferry, Knights Corner)
  - Used in conjunction with regular Xeon CPU
  - Intel prefer the term “coprocessor” to “accelerator”
- Essentially a many-core CPU
  - Typically 50-100 cores per chip
  - with wide vector units
  - So again uses concept of many simple low-power cores – Each performing multiple operations per cycle
- But latest “Knights Landing (KNL)” is not normally used as an accelerator
  - Instead a self-hosted CPU

Thread-Level Parallelism (TLP)

Flynn's Taxonomy

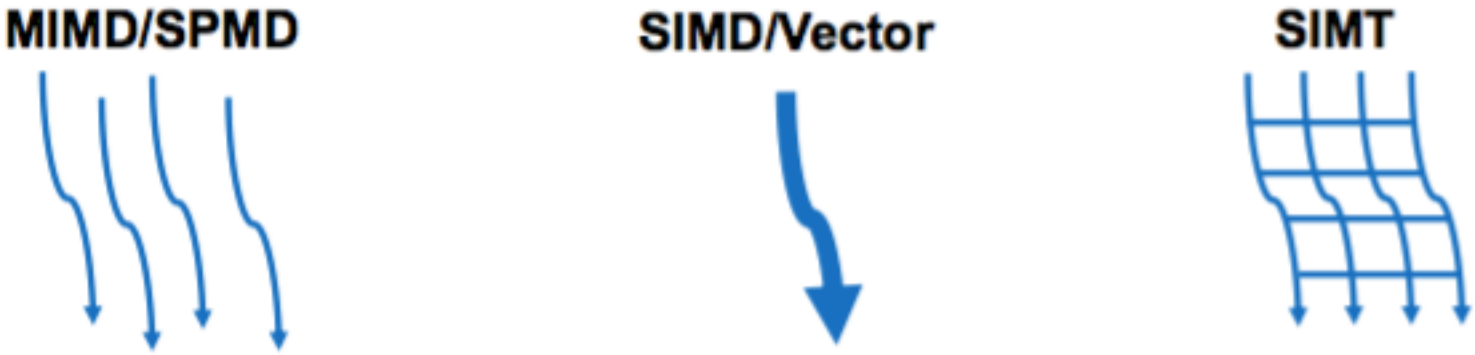
- Focus: Data parallel workloads
  - Independent, identical computation on multiple data inputs
- MIMD (Multiple Instruction, Multiple Data):
  - Split independent work over multiple processors
  - Subcategory: SPMD (Single Program, Multiple Data)
    - » Only if work is identical (same program)
- SIMD (Single Instruction, Multiple Data):
  - Split identical, independent work over multiple execution units
  - More efficient: eliminate redundant fetch/decode vs. SPMD/MIMD
  - Use single PC and single register file

SIMT

- SIMD's cousin: SIMT (Single Instruction, Multiple Thread)
  - Split identical, independent work over multiple lockstep threads
  - One PC for group of lockstep threads, but multiple register files
  - This is what GPUs do today
  - Work well for **streaming** applications

- People use SIMT and SIMD somewhat interchangeably
  - They do have differences though

Execution models



	MIMD/SPMD	SIMD/Vector	SIMT
Example	Multicore CPUs	x86 SSE/AVX	GPUs
Pros	More general: better support for TLP	Able to mix serial and parallel code	Easier to program, Scatter & Gather operations
Cons	Inefficient for data parallelism	Gather/Scatter implementations more complicated	Divergence kills performance

GPU & memory

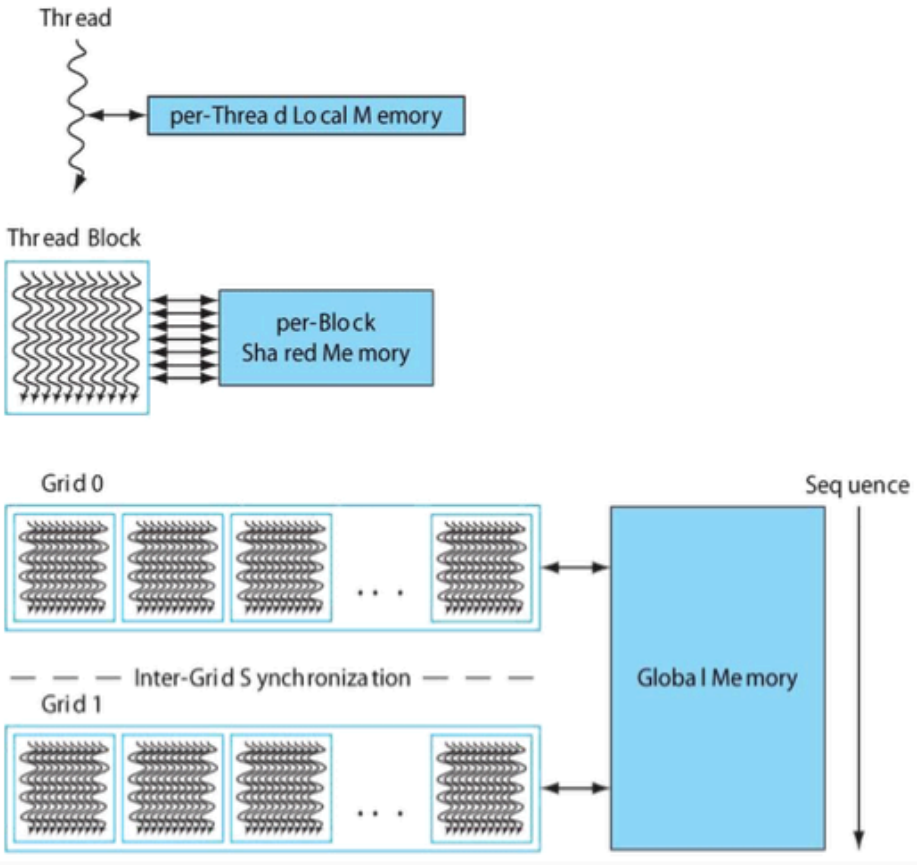
- GPUs optimized for streaming computations
  - Thus, we have a lot of streaming memory accesses
- DRAM: 100's of GPU cycles per memory access
  - How to hide this overhead & keep the GPU busy in the meantime?

- Traditional CPU approaches:
  - Caches ➔ Need spatial/temporal locality
    - » Streaming applications have little reuse
  - OOO (Out-of-Order)/Dynamic Scheduling ➔ Need ILP
    - » Too power hungry, diminishing returns for GPU applications
  - Multicore/Multithreading/SMT(Simultaneous Multithreading) ➔ need independent threads

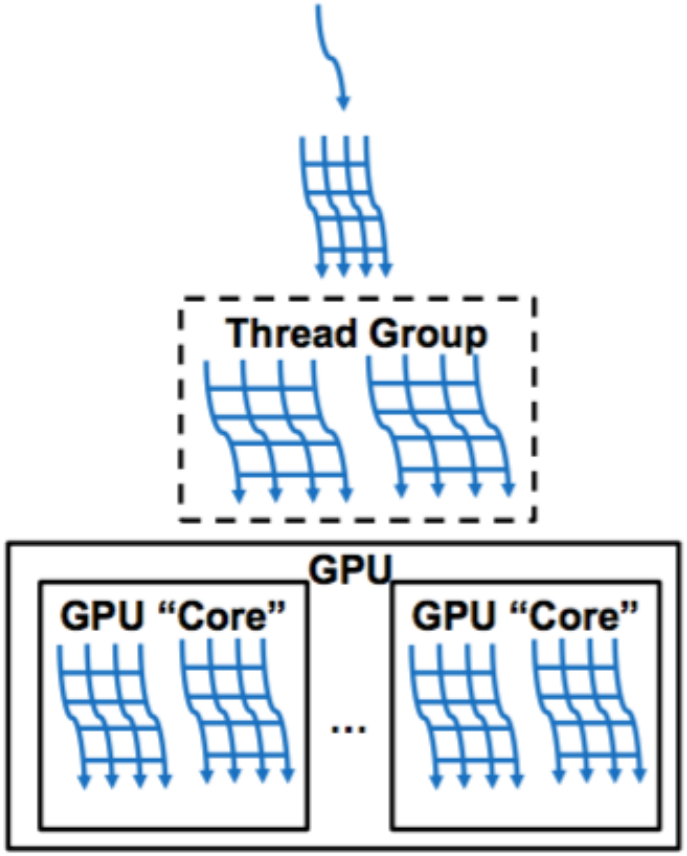
CUDA GPU thread model

Single-program multiple data (SPMD) model

- Each **thread** has local memory
- Parallel threads packed in **blocks**
  - Access to per-block shared memory
  - Can synchronize with barrier
- **Grids** include independent blocks
- *Vector analog:* Program a single lane; HW dynamically schedules

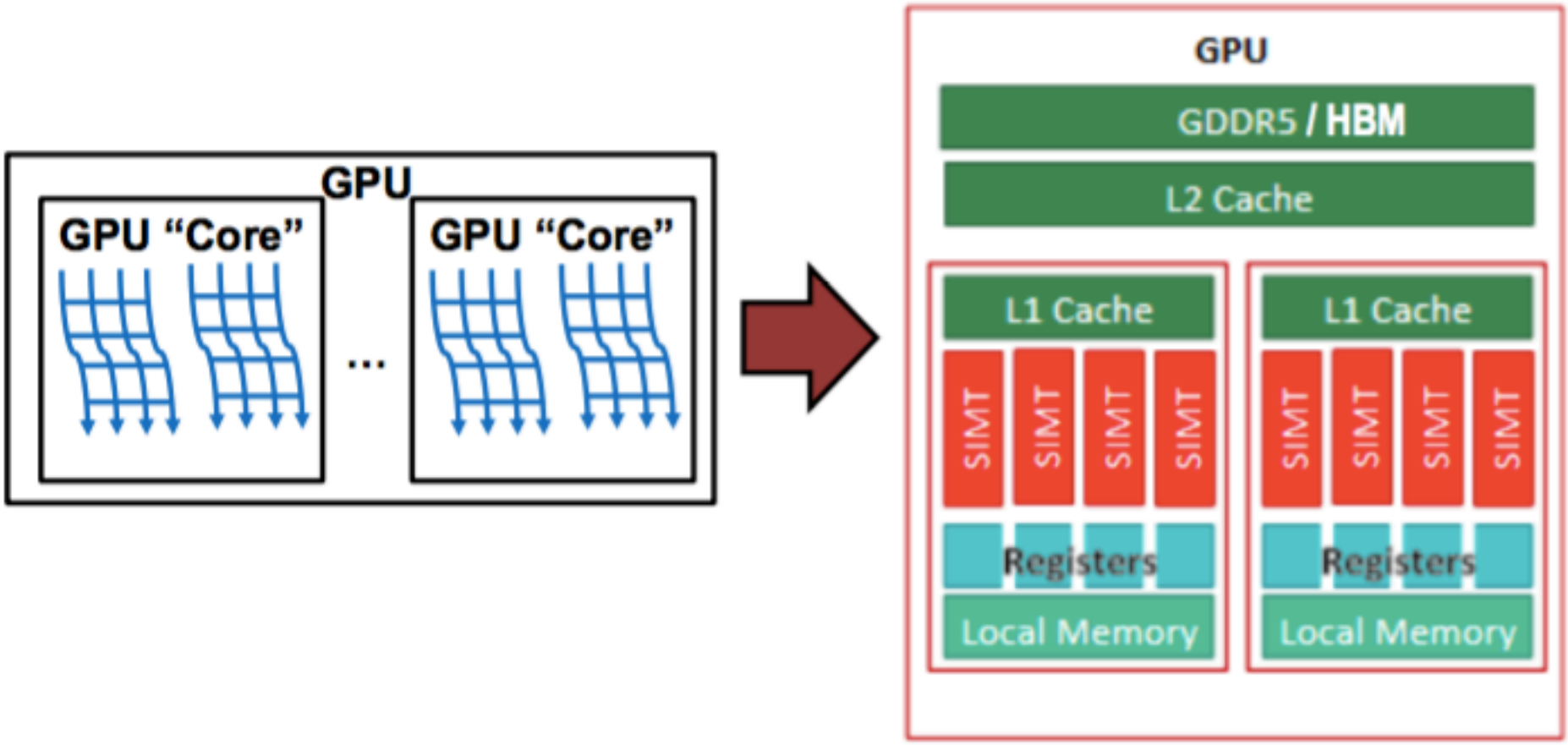


GPU component names



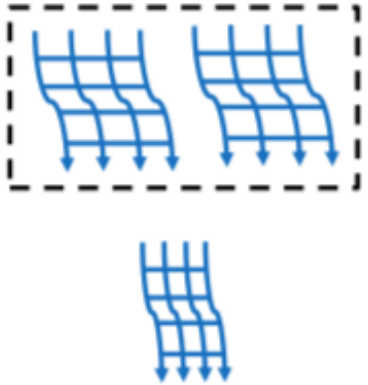
CUDA/HIP	OpenCL
Thread	Work-item
Warp	Wavefront
Thread Block/CTA	Workgroup
Grid (Kernel)	NDRange (Kernel)

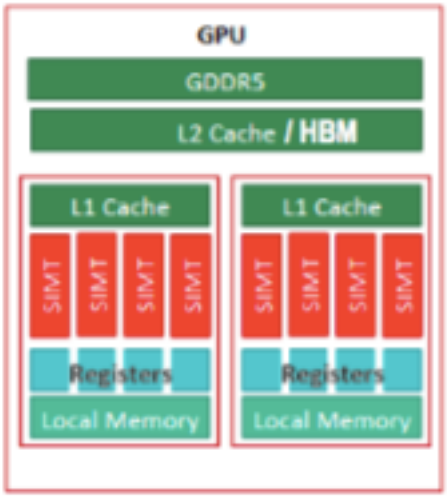
GPU hardware



Compute Unit (CU) – GPU "core"

- Job: run thread blocks/workgroups
  - Contains multiple SIMT units (4 in picture below)
  - Each cycle, schedule one SIMT unit
- SIMT unit: runs wavefronts/warps
  - Run the threads
  - AMD: size N (e.g., 10) wavefront instruction buffer
    - » 4 cycles to execute one wavefront
    - » Average: fetch and commit 1 wavefront/cycle





CPU vs. GPU

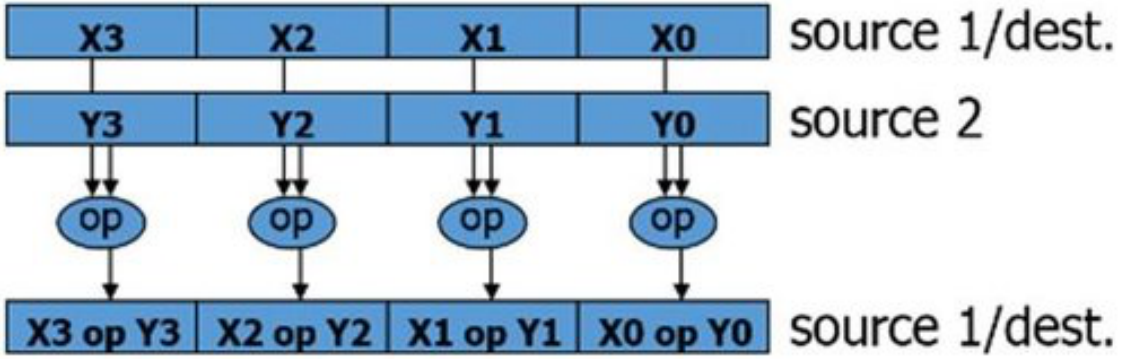
CPU

SIMD

1 instruction – multiple data

SSE2/3/4 – Neon – Altivec

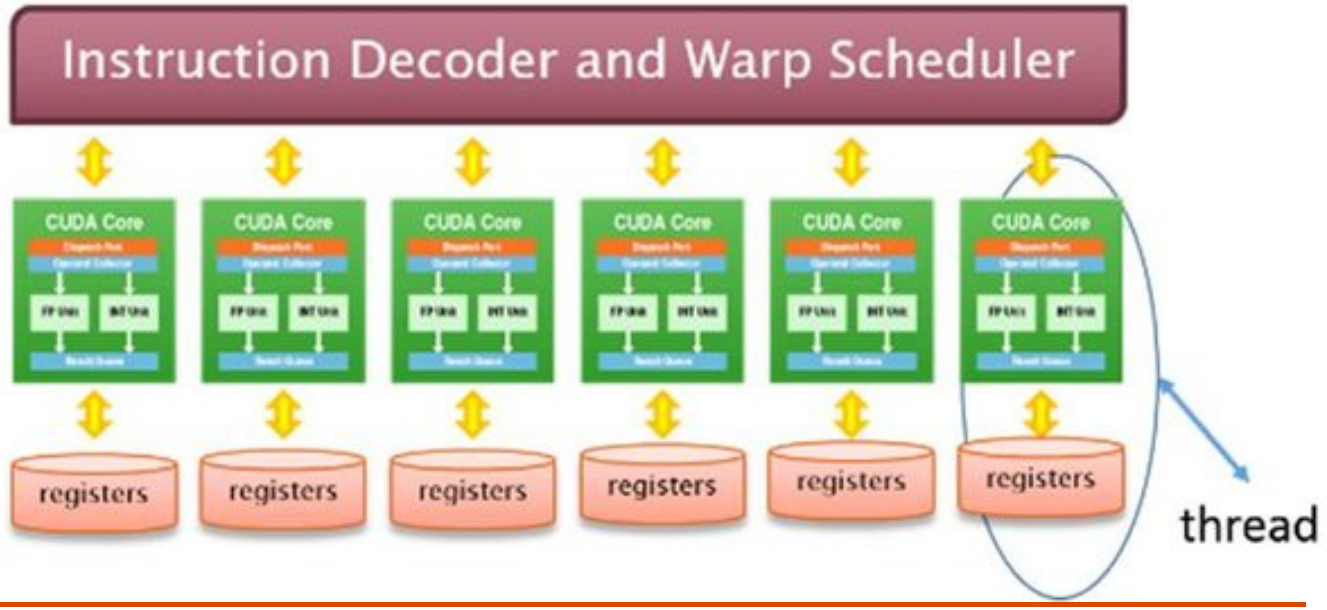
AVX – AVX2...



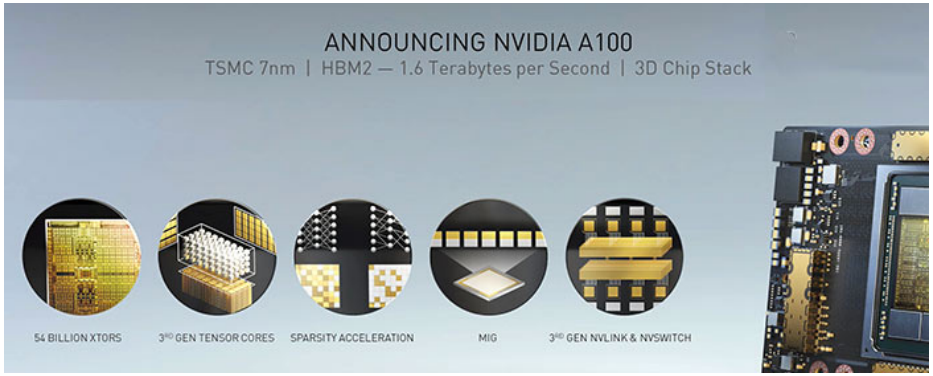
GPU

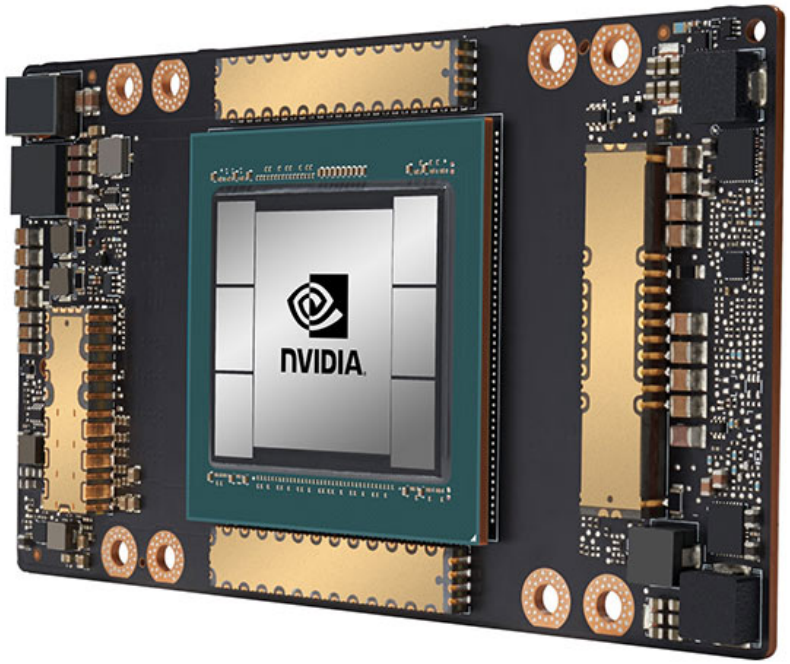
SIMT

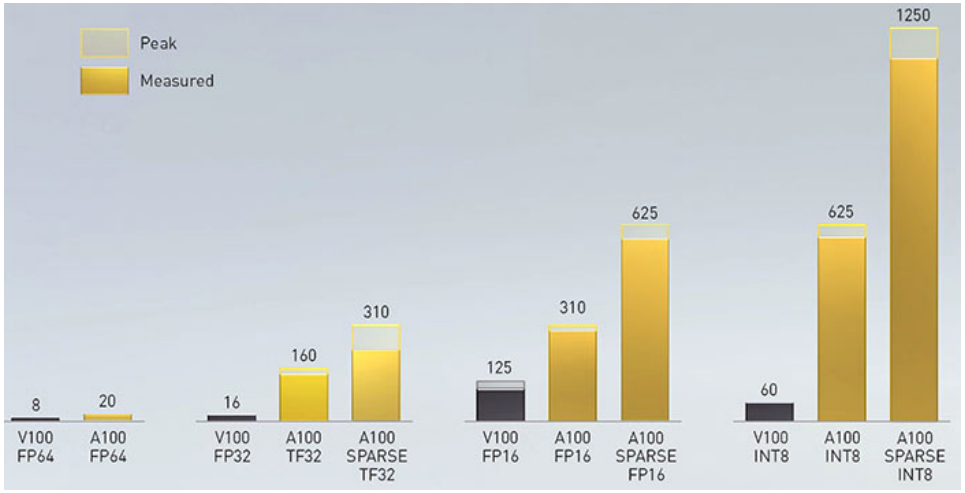
1 instruction – multiple threads



Nvidia A100 GPU



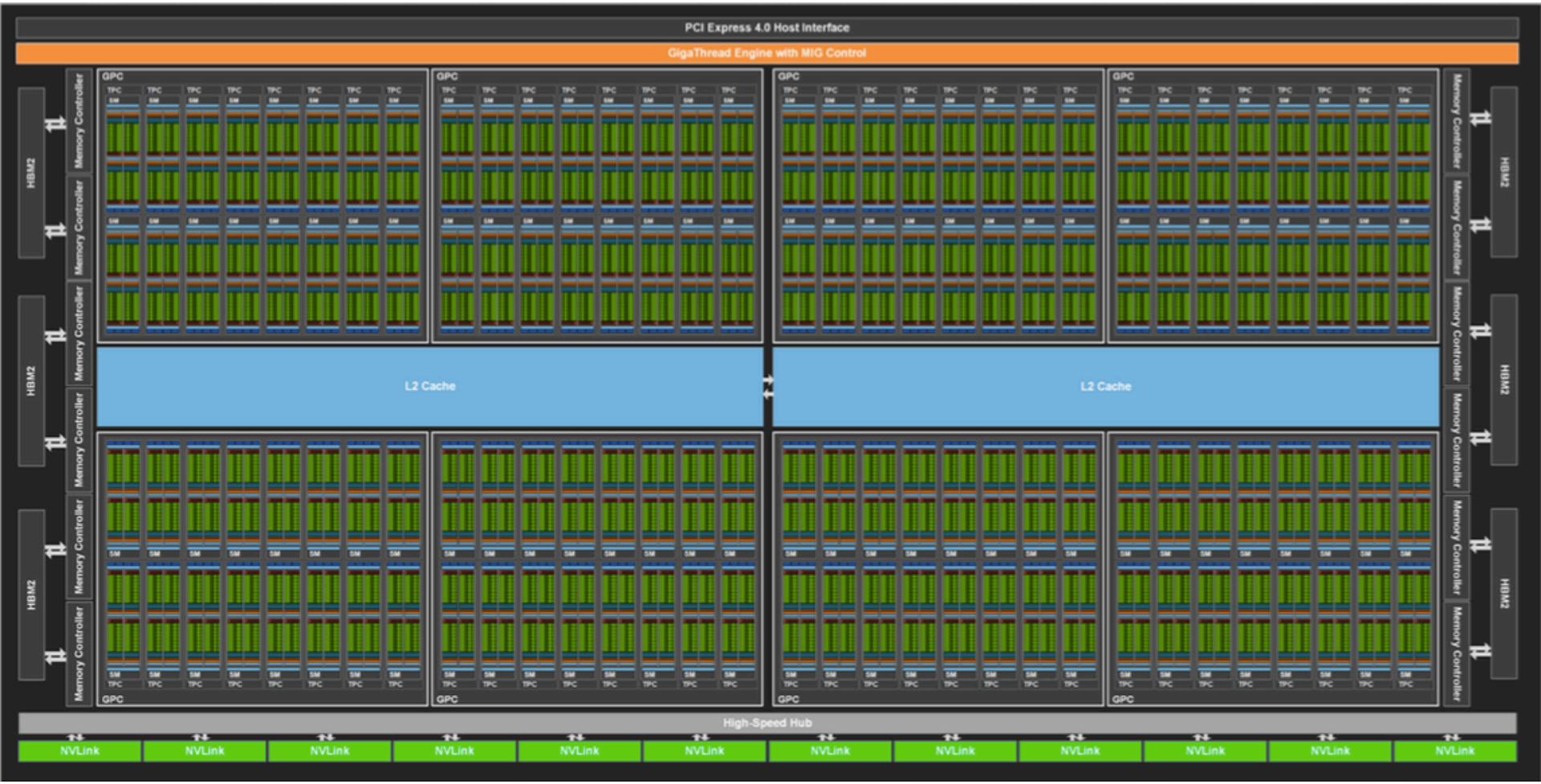




Nvidia GA100 GPU (1)

- Based on NVIDIA Ampere architecture
- Composed of multiple:
  - GPU Processing Clusters (GPCs)
  - Texture Processing Clusters (TPCs)
  - Streaming Multiprocessors (SMs)
  - HBM2 memory controllers
- The **full implementation** of the GA100 GPU includes:
  - 8 GPCs, 8 TPCs/GPC, 2 SMs/TPC, 16 SMs/GPC, 128 SMs per full GPU
  - 64 FP32 CUDA Cores/SM, 8192 FP32 CUDA Cores per full GPU
  - 4 Third-generation Tensor Cores/SM, 512 Third-generation Tensor Cores per full GPU
  - 6 HBM2 stacks, 12 512-bit Memory Controllers

Nvidia GA100 GPU (2)



Nvidia A100 Tensor Core GPU

- The NVIDIA A100 Tensor Core GPU implementation of the GA100 GPU includes:
  - 7 GPCs, 7 or 8 TPCs/GPC, 2 SMs/TPC, up to 16 SMs/GPC, 108 SMs
  - 64 FP32 CUDA Cores/SM, 6912 FP32 CUDA Cores per GPU
  - 4 Third-generation Tensor Cores/SM, 432 Third-generation Tensor Cores per GPU
  - 5 HBM2 stacks, 10 512-bit Memory Controllers
- Built on the TSMC 7nm N7 FinFET fabrication process
  - 12nm FFN process used in Tesla V100
- Multi-Instance GPU (MIG)
  - Delivering up to 7x more GPU Instances

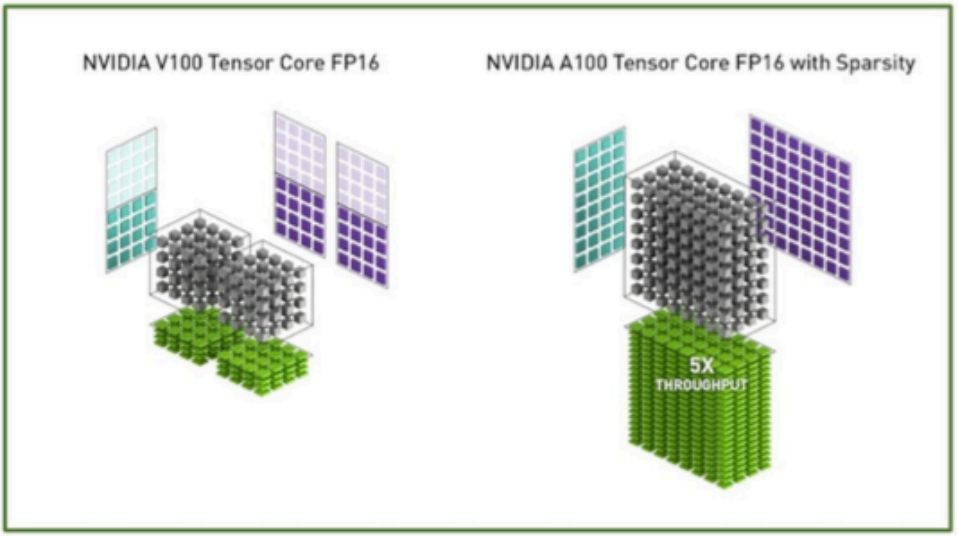
A100 GPU: Tensor cores

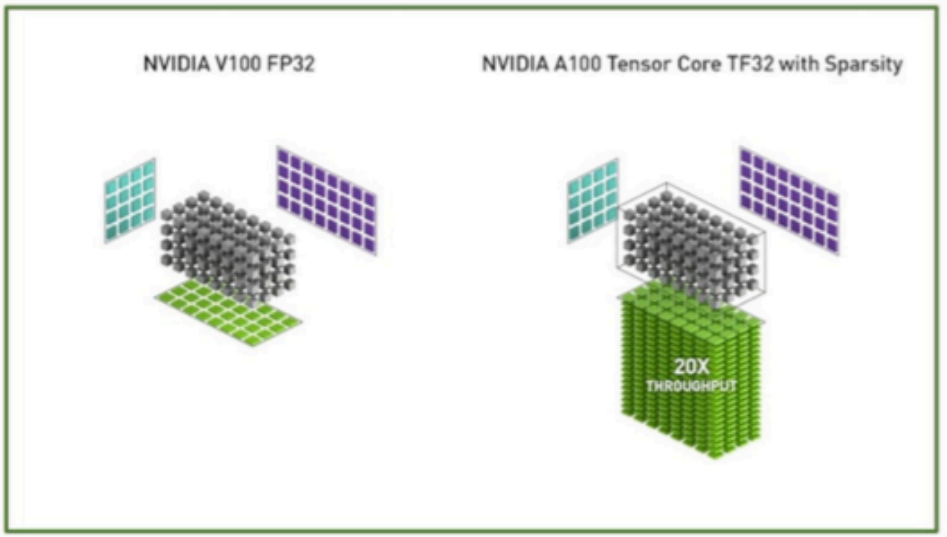
- 8 Tensor cores per SM
- Each Tensor Core performing 256 FP16/FP32 mixed-precision fused multiply-add (FMA) operations per clock
- A100 has 4 Tensor Cores per SM, which together deliver 1024 dense FP16/FP32 FMA operations per clock
  - 2x compared to Volta and Turing
- New Tensor Core sparsity feature exploits fine-grained structured sparsity in deep learning networks

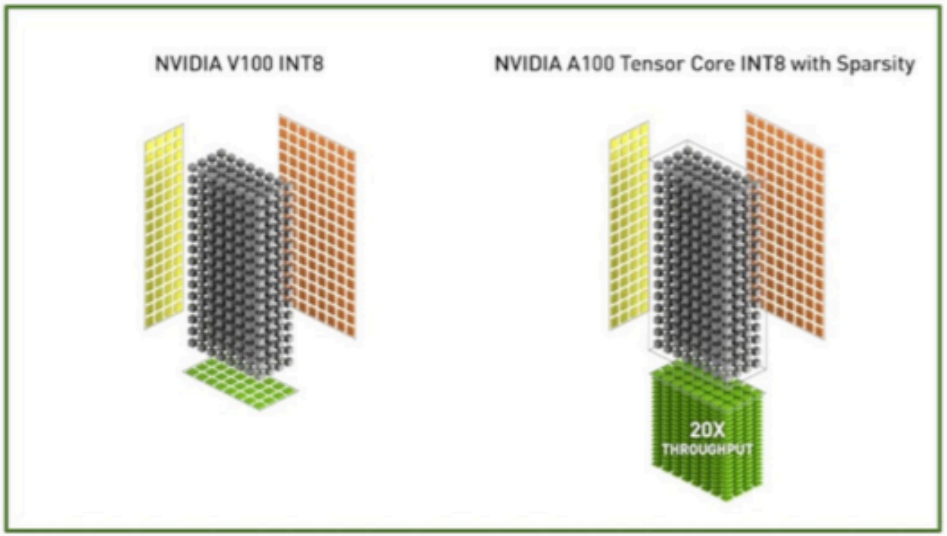
	V100	A100	A100 Sparsity[1]	A100 Speedup	A100 Speedup with Sparsity
A100 FP16 vs V100 FP16	31.4 TFLOPS	78 TFLOPS	NA	2.5x	NA
A100 FP16 TC vs V100 FP16 TC	125 TFLOPS	312 TFLOPS	624 TFLOPS	2.5x	5x
A100 BF16 TC vs V100 FP16 TC	125 TFLOPS	312 TFLOPS	624 TFLOPS	2.5x	5x
A100 FP32 vs V100 FP32	15.7 TFLOPS	19.5 TFLOPS	NA	1.25x	NA
A100 TF32 TC vs V100 FP32	15.7 TFLOPS	156 TFLOPS	312 TFLOPS	10x	20x
A100 FP64 vs V100 FP64	7.8 TFLOPS	9.7 TFLOPS	NA	1.25x	NA
A100 FP64 TC vs V100 FP64	7.8 TFLOPS	19.5 TFLOPS	NA	2.5x	NA
A100 INT8 TC vs V100 INT8	62 TOPS	624 TOPS	1248 TOPS	10x	20x
A100 INT4 TC	NA	1248 TOPS	2496 TOPS	NA	NA
A100 Binary TC	NA	4992 TOPS	NA	NA	NA

1 - Effective TOPS / TFLOPS using the new Sparsity Feature

A100 vs. V100 Tensor Core operations

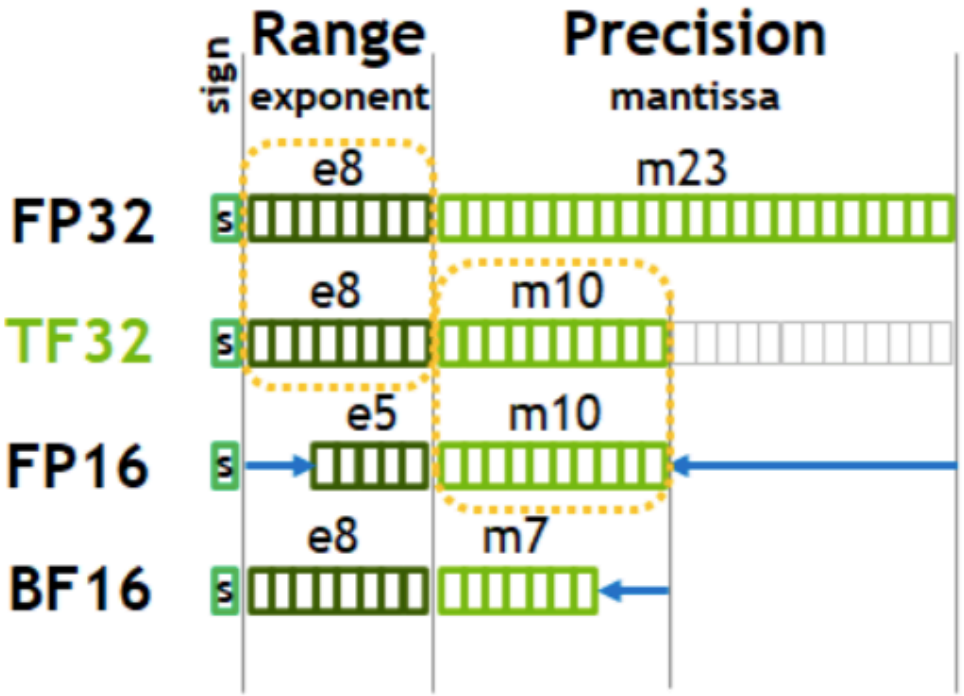


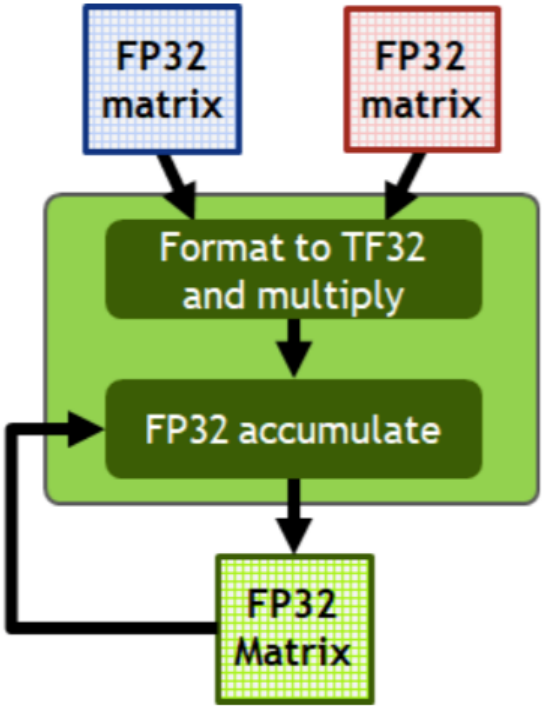




TensorFloat-32 & BF16 (1)

- FB32: 32-bit IEEE 754 single-precision floating-point format
- BF16 (Bfloat-16): Brain floating point
- TF32: Tensor Float-32
  - 10x FP32





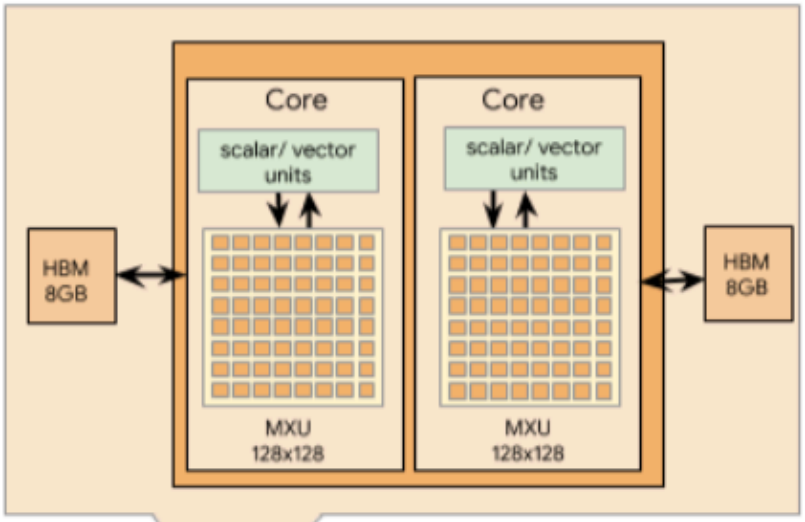
TensorFloat-32 & BF16 (2)

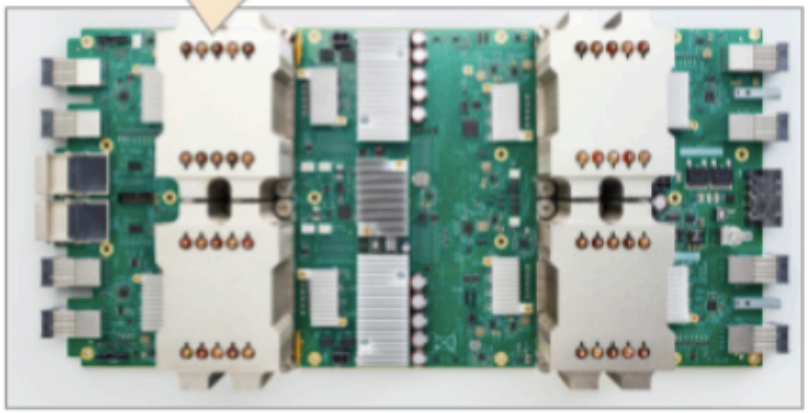
	INPUT OPERANDS	ACCUMULATOR	TOPS	X-factor vs. FFMA	SPARSE TOPS	SPARSE X-factor vs. FFMA
V100	FP32	FP32	15.7	1x	-	-
	FP16	FP32	125	8x	-	-
A100	FP32	FP32	19.5	1x	-	-
	TF32	FP32	156	8x	312	16x
	FP16	FP32	312	16x	624	32x
	BF16	FP32	312	16x	624	32x
	FP16	FP16	312	16x	624	32x
	INT8	INT32	624	32x	1248	64x
	INT4	INT32	1248	64x	2496	128x
	BINARY	INT32	4992	256x	-	-
	IEEE FP64		19.5	1x	-	-

Domain-Specific Architectures

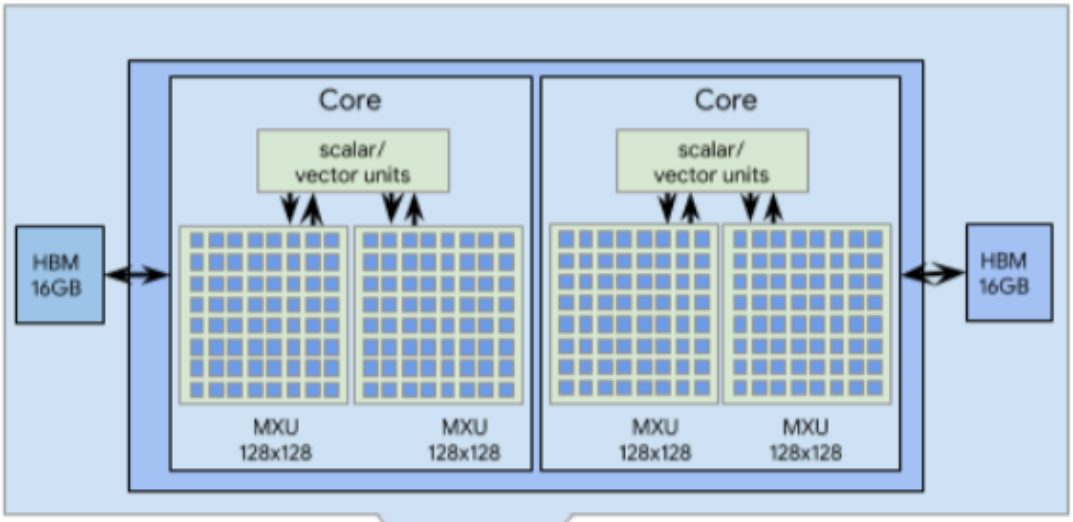
- Google TPU
- Microsoft Catapult
- Intel Crest
- Pixel Visual Core
- …

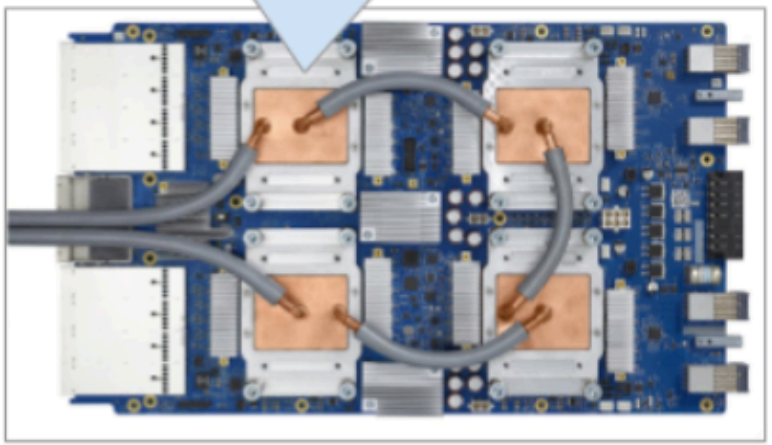
TPU (1)





TPU v2 - 4 chips, 2 cores per chip

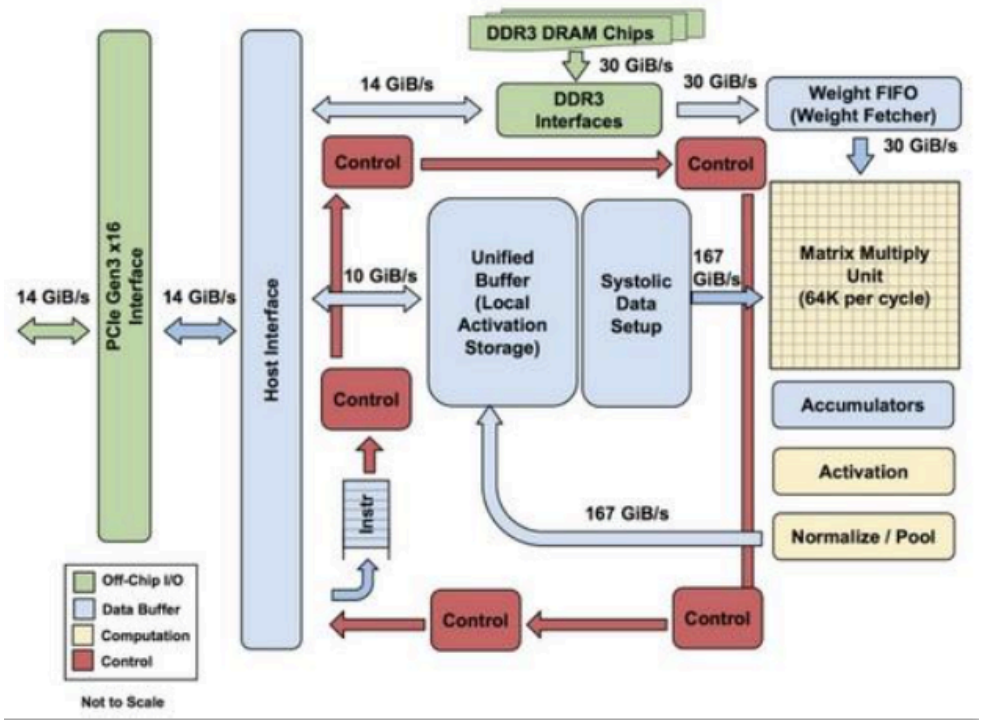




TPU v3 - 4 chips, 2 cores per chip

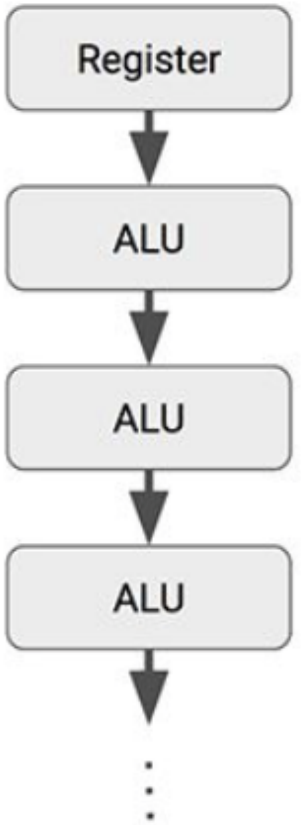
TPU (2)

- Large, on-chip DRAM required for accessing pairing weight values
- It is Possible to simultaneously store weights and load activations
  - TPU can do 64,000 of these accumulates per cycle
- First generation used 8-bit operands and quantization
  - Second generation uses 16-bit
- Matrix Multiplication Unit has 256 × 256 (65,536) ALUs



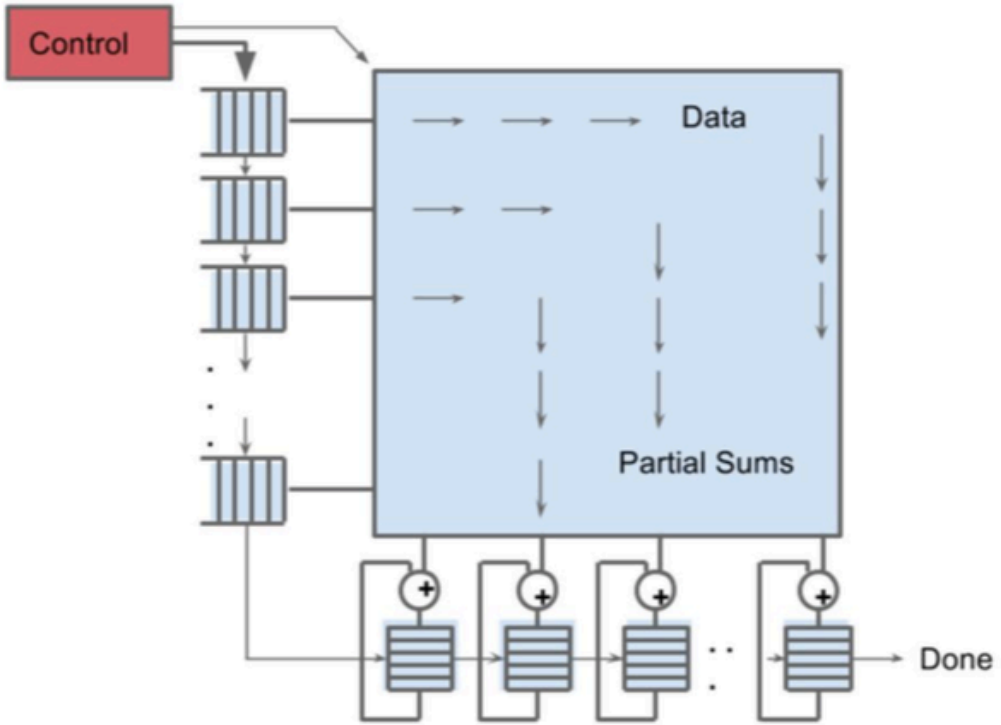
TPU: performance

- TPU consists of Matrix Multiplier Unit (MXU)
- MXU performs hundreds of thousands of operations per clock cycle
- Reads an input value only once
- Inputs are used many times without storing back to register
- Wires connect adjacent ALUs
- Multiplication and addition are performed in specific order
- Short and energy efficient
- Design is known as systolic array



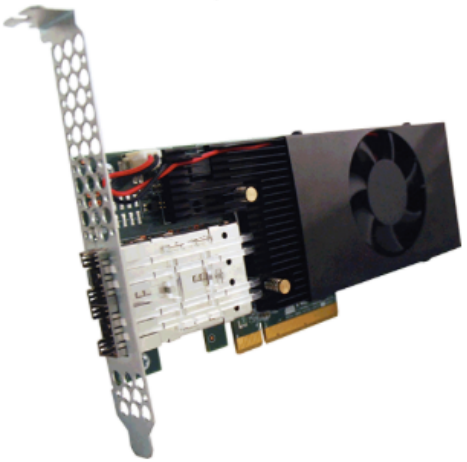
Matrix Multiplication Unit

- Contains 256 x 256 = 65,536 ALUs
- TPU runs at 700 MHz
- Able to compute 46 x 1012 multiply-and-add operations per second
- Equivalent to 92 TeraOps per second in matrix unit



Microsoft's Catapult v2

- Reconfigurable cloud architecture developed by Microsoft
- Acceleration board
  - FPGA Stratix V D5 with 172K ALMs



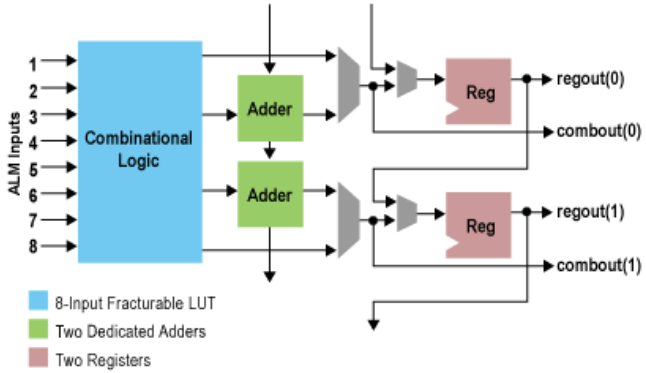
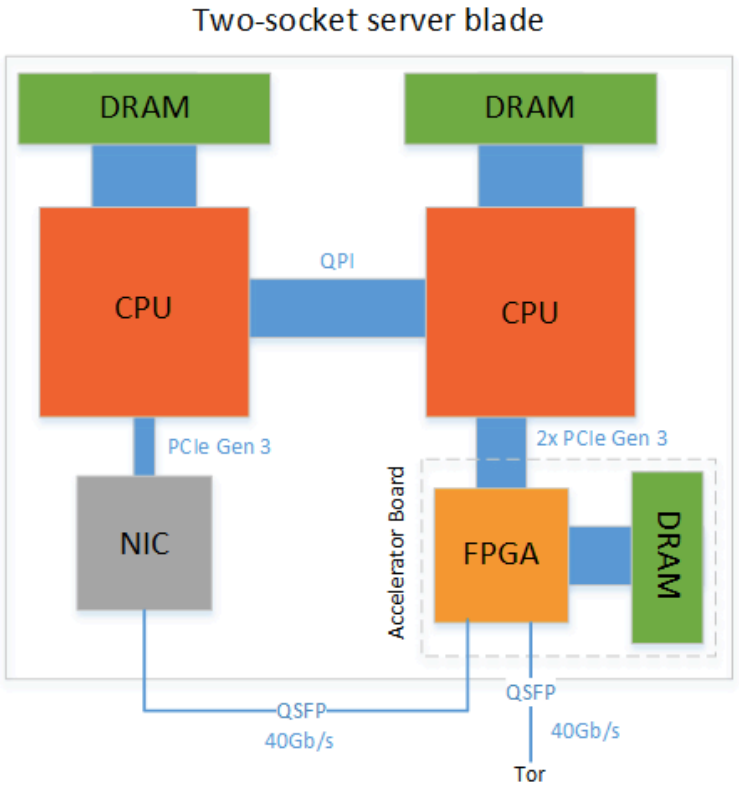
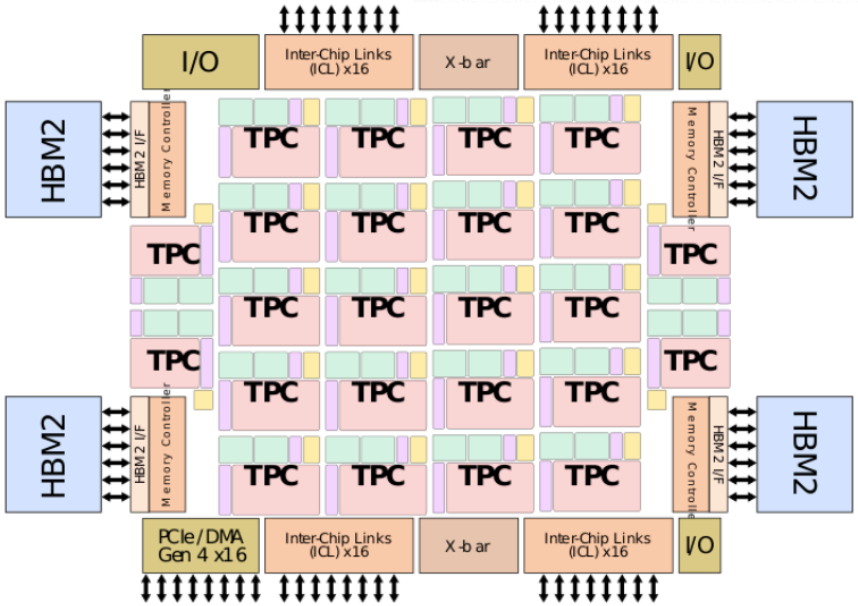


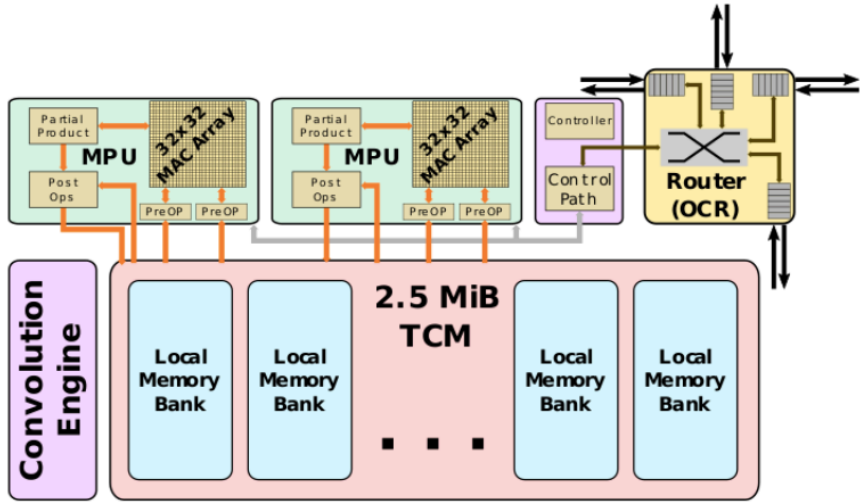
Figure 1. Stratix IV FPGA ALM



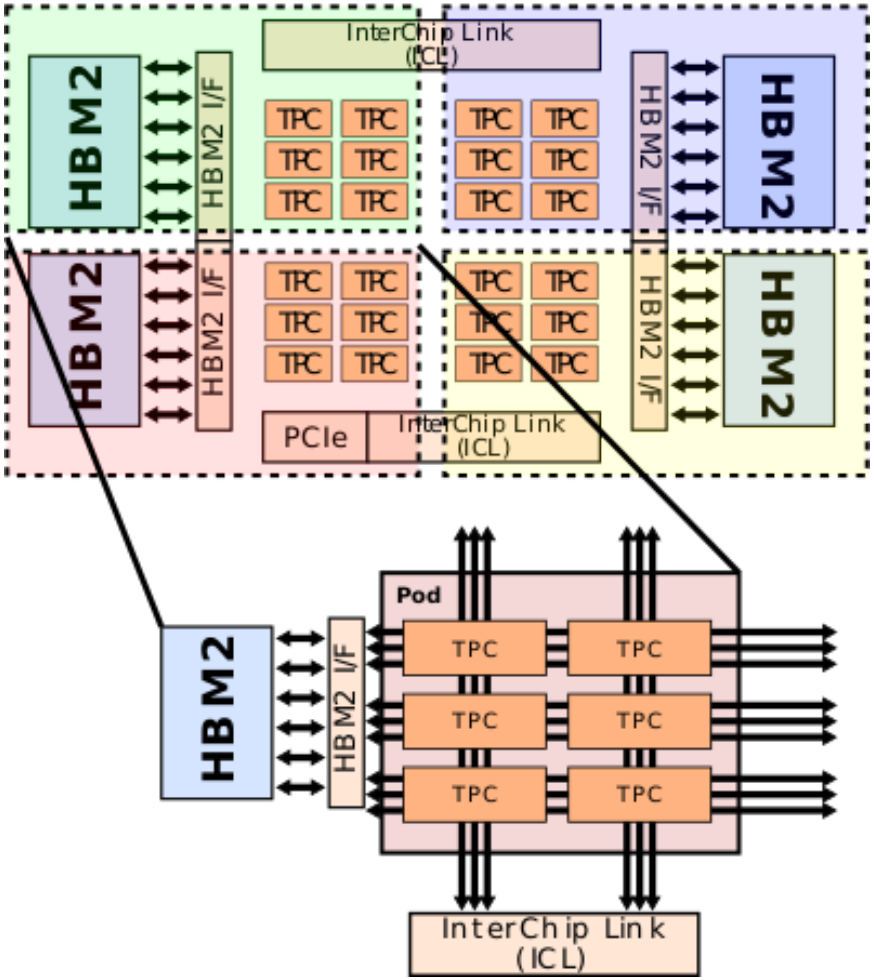
Intel Crest

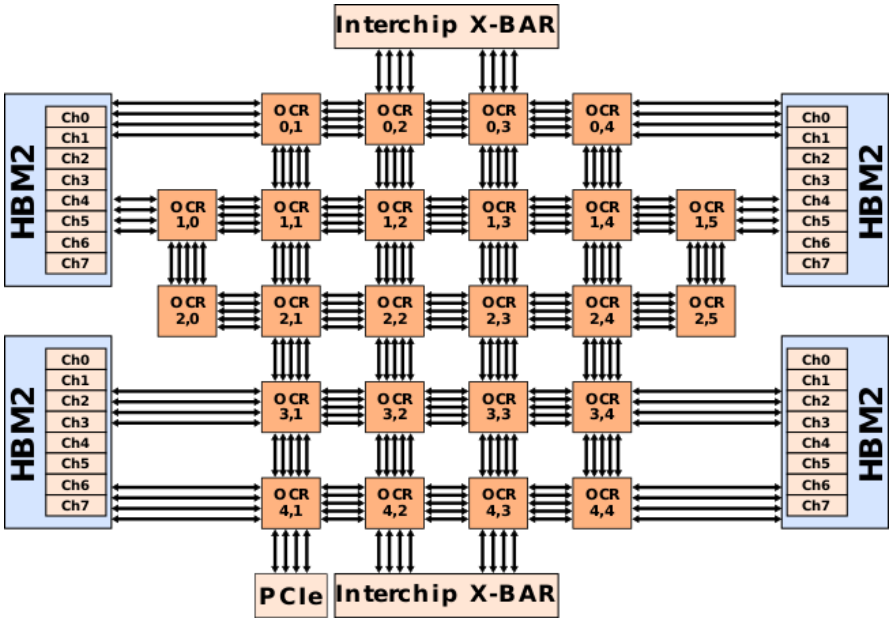
- A neural processor microarchitecture designed by Nervana
- Lake Crest in 2016
- Spring Crest (SCR) in 2019: fabricated on TSMC's 16 nm process
  - 24 high-performance tensor processor clusters (TPCs)
  - 2 MAC processing units (MPU)/TPC
  - Each of the MPU pairs integrates a 32x32 array for a total of 98,304 FLOPs each cycle for a total of up to 119 TOPS of compute
  - The MPUs are fed by 60 MiB of distributed SRAM (24 TPCs x 2.5 MiB)
  - Using bfloat16 with a 32-bit (SP FP) accumulate





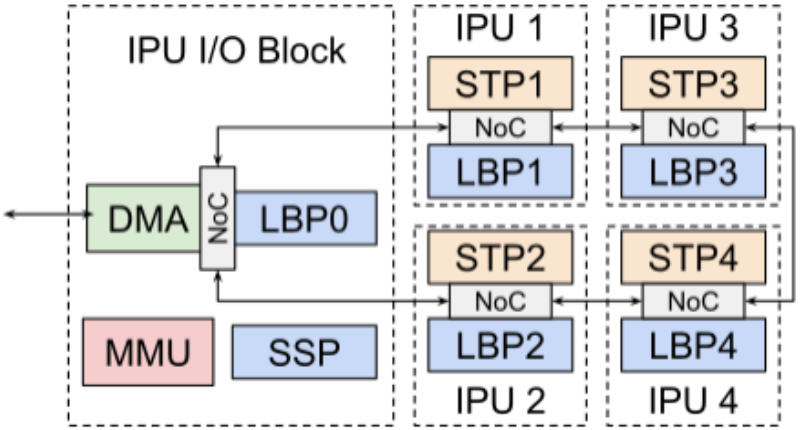
Intel Crest
Network-on-Chip (NoC)

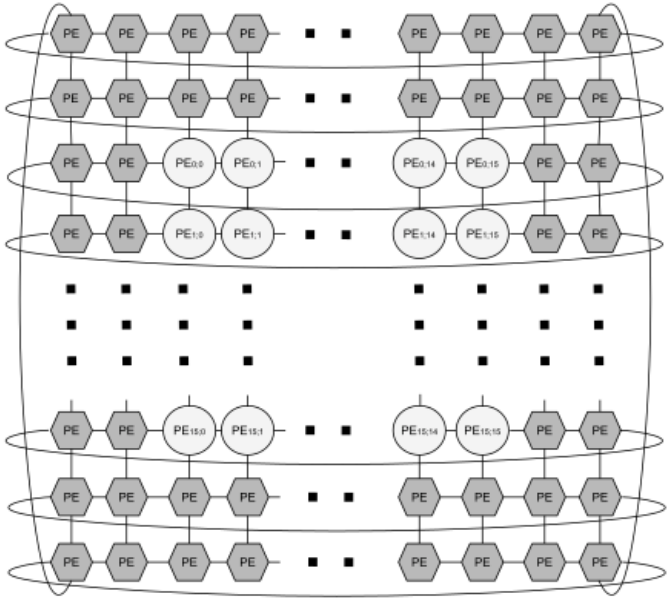




Pixel Visual Core (PVC)

- A series of ARM-based system in package (SiP) image processors designed by Google
- Used in Google Pixel 3 and 3 XL
- The core of a PVC is the image processing unit (IPU) a programmable unit tailored for image processing
- IPU
  - 1 stencil processor (STP), 1 Line Buffer Pool (LBP) and 1 NoC
- STP
  - a 2-D array of PEs: Ex a 16x16 array of full PEs and four lanes of simplified PEs called "halo"
  - Each PEs features 2x 16-bit arithmetic logic units (ALUs), 1x 16-bit Multiplier–accumulator unit (MAC), 10x 16-bit registers, and 10x 1-bit predicate registers

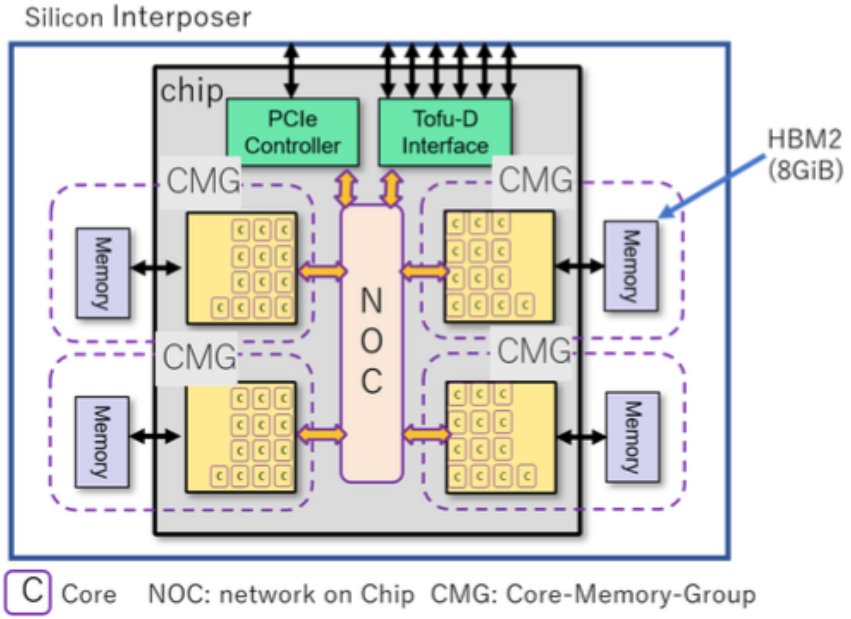




Fugaku's FUjitsu A64fx processor

GPU-like High performance in HPC, AI/Big Data, Auto Driving...

- **An Many-Core ARM CPU...**
  - 48 compute cores + 2 or 4 assistant (OS) cores
  - Brand new core design
  - Near Xeon-Class Integer performance core
  - ARM V8 --- 64bit ARM ecosystem
  - Tofu-D + PCIe 3 external connection



- **...but also an accelerated GPU-like processor**
  - SVE 512 bit x 2 vector extensions (ARM & Fujitsu)
  - Integer (1, 2, 4, 8 bytes) + Float (16, 32, 64 bytes)
  - Cache + scratchpad-like local memory (sector cache)
  - HBM2 on package memory – Massive Mem BW (Bytes/DPF ~0.4)
  - Streaming memory access, strided access, scatter/gather etc.
  - Intra-chip barrier synch. and other memory enhancing features

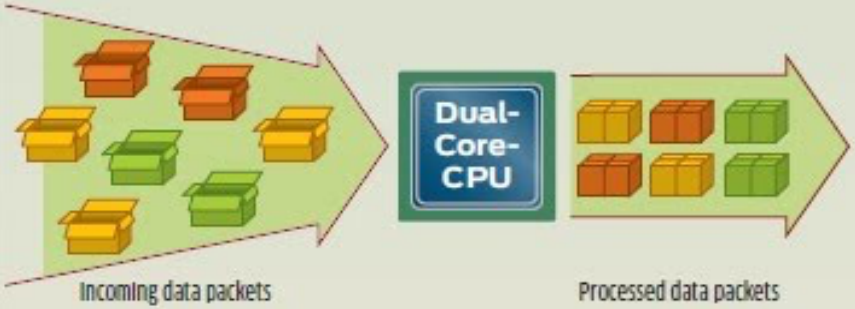
CPU AND GRAPHICS PROCESSORS IN COMPARISON

A classic CPU or a graphics chip? The choice depends on the processing task. GPUs beat the flexible CPUs in spite of their considerably lower clock rate when processing multiple similar data packets.

Different data packets: CPU advantage

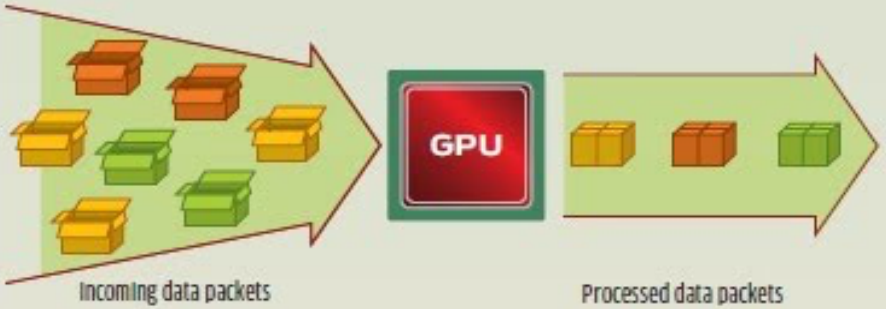
Dual-core CPU

If the requirements are different, the dual-core CPU can process two packets per core parallely.



Graphics processor

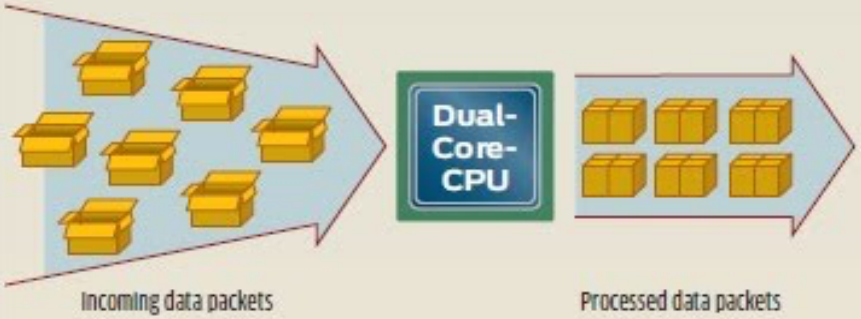
The GPU cannot work parallely in case of complex problems and processes the packets individually and at a much slower speed.



Identical data packets: GPU advantage

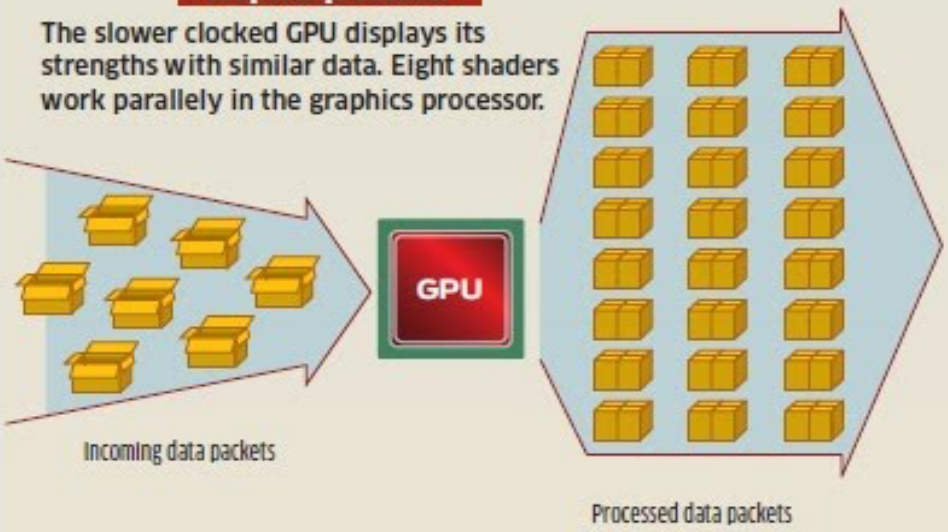
Dual-core CPU

The dual-core CPU processes two data packets parallely even in case of identical tasks.



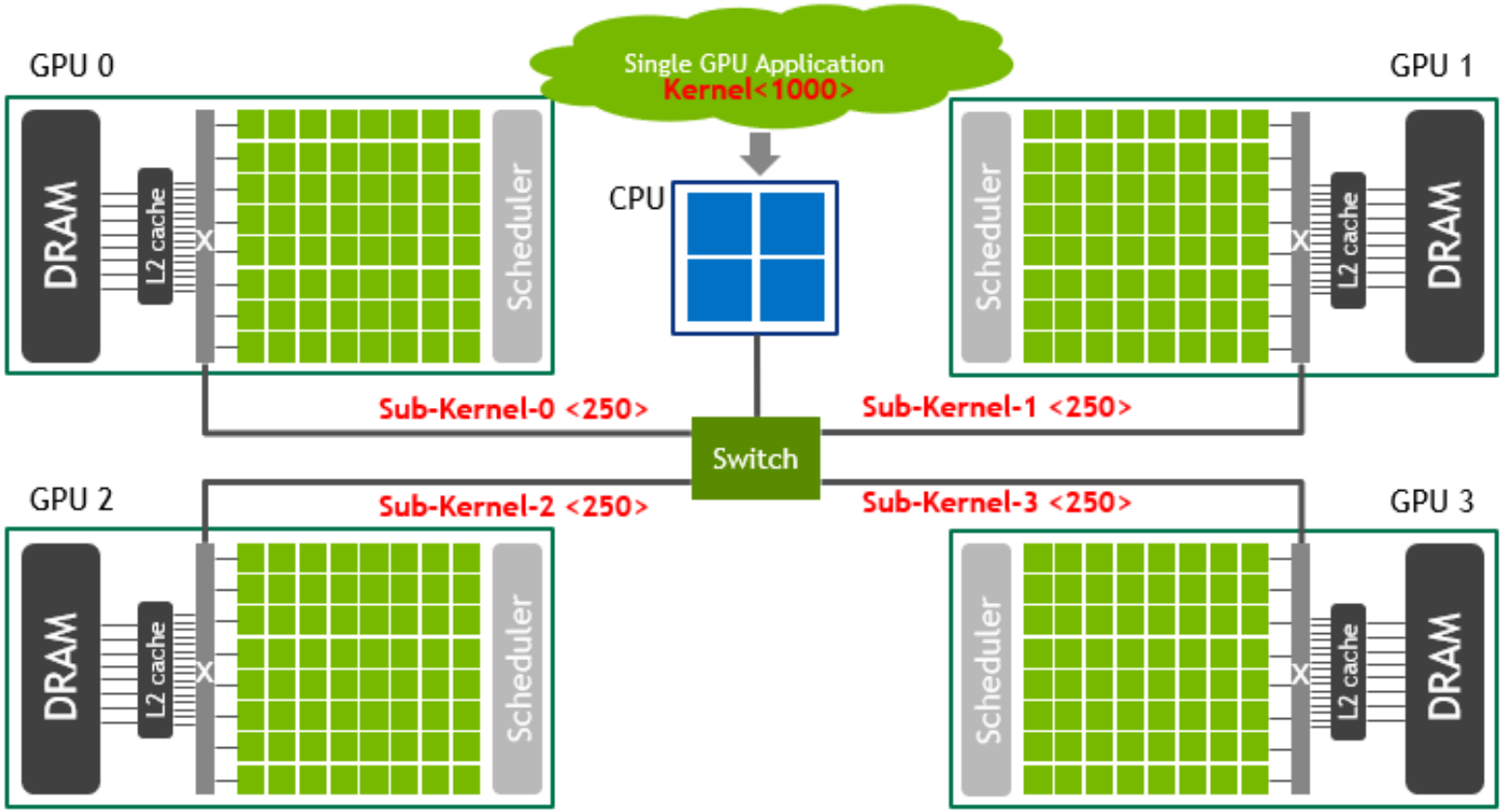
Graphics processor

The slower clocked GPU displays its strengths with similar data. Eight shaders work parallely in the graphics processor.



NUMA AWARE GPUS

Enable Scaling of Single GPU Applications Transparently on Multi-GPUs



- Dynamic and asymmetric NVLINK bandwidth reconfiguration
- Dynamic NUMA-aware caching strategies