University Of Diyala
College Of Engineering
Computer Engineering Department

COMPUTER ARCHITECTURE II

PART 7: SIMD PROCESSORS AND GPUS

Asst. Prof. Ahmed Salah Hameed

Second stage

2022-2023

In This Lecture

- SIMD Processing
  - Vector and Array Processors

- Graphics Processing Units (GPUs)

Flynn’s Taxonomy of Computers

- Mike Flynn, “Very High-Speed Computing Systems,” Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

Data Parallelism

- Concurrency arises from performing the same operation on different pieces of data
  - Single instruction multiple data (SIMD)
  - E.g., dot product of two vectors
- Contrast with data flow
  - Concurrency arises from executing different operations in parallel (in a data driven manner)
- Contrast with thread ("control") parallelism
  - Concurrency arises from executing different threads of control in parallel
- SIMD exploits operation-level parallelism on different data
  - Same operation concurrently applied to different pieces of data
  - A form of ILP where instruction happens to be the same across data

SIMD Processing

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

- Time-space duality

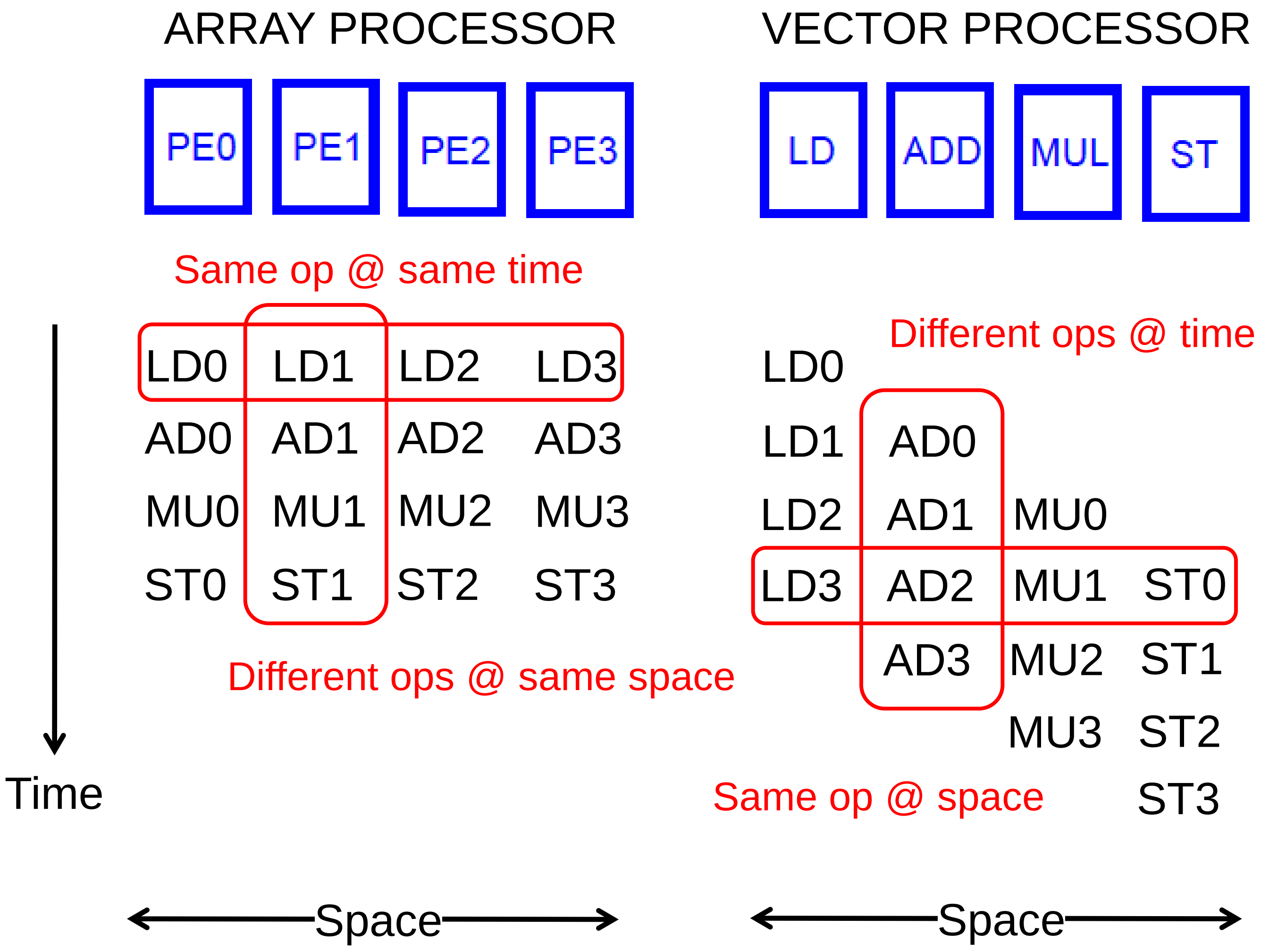
  - Array processor: Instruction operates on multiple data elements at the same time using different spaces

  - Vector processor: Instruction operates on multiple data elements in consecutive time steps using the same space

Array vs. Vector Processors

Instruction Stream

LD	VR ← A[3:0]
ADD	VR ← VR, 1
MUL	VR ← VR, 2
ST	A[3:0] ← VR



ARRAY PROCESSOR

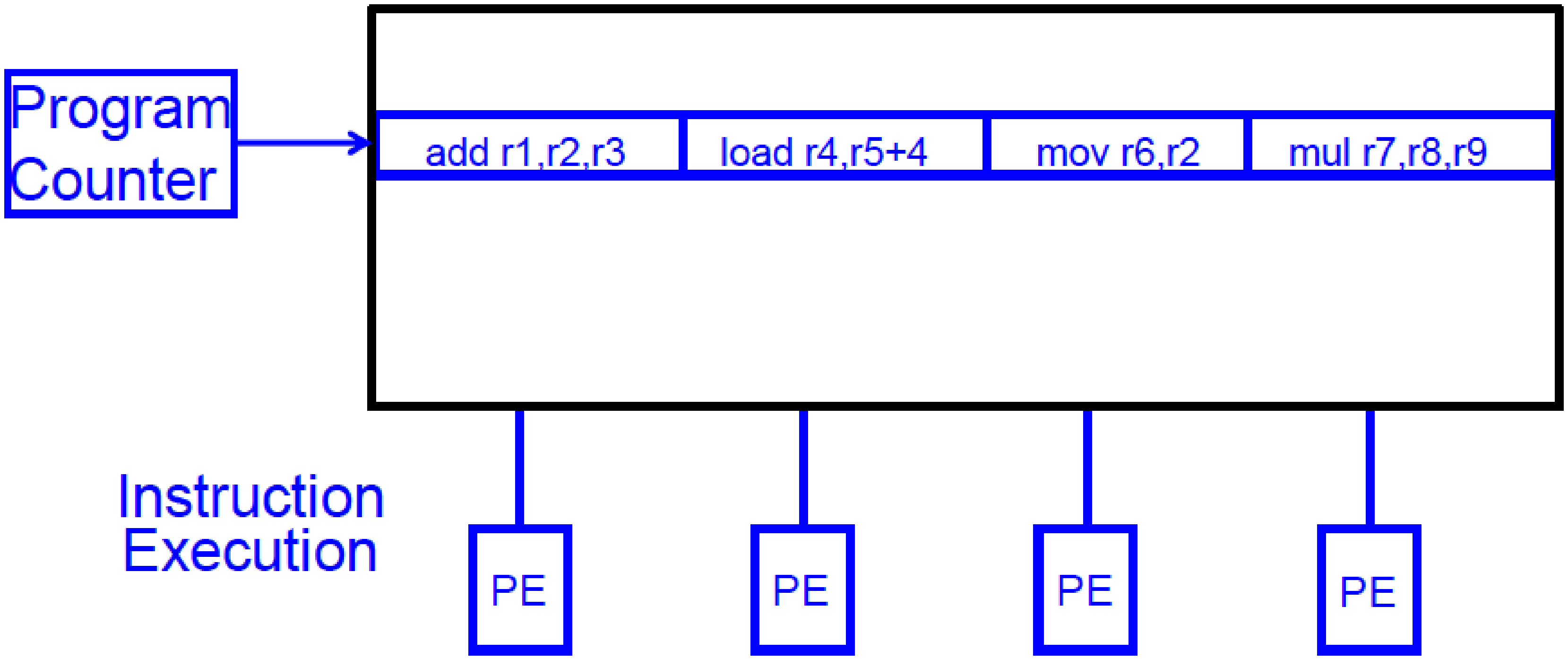
PE0	PE1	PE2	PE3
LD0	LD1	LD2	LD3
AD0	AD1	AD2	AD3
MU0	MU1	MU2	MU3
ST0	ST1	ST2	ST3

VECTOR PROCESSOR

LD	ADD	MUL	ST
LD0			
LD1	AD0		
LD2	AD1	MU0	
LD3	AD2	MU1	ST0
	AD3	MU2	ST1
		MU3	ST2
			ST3

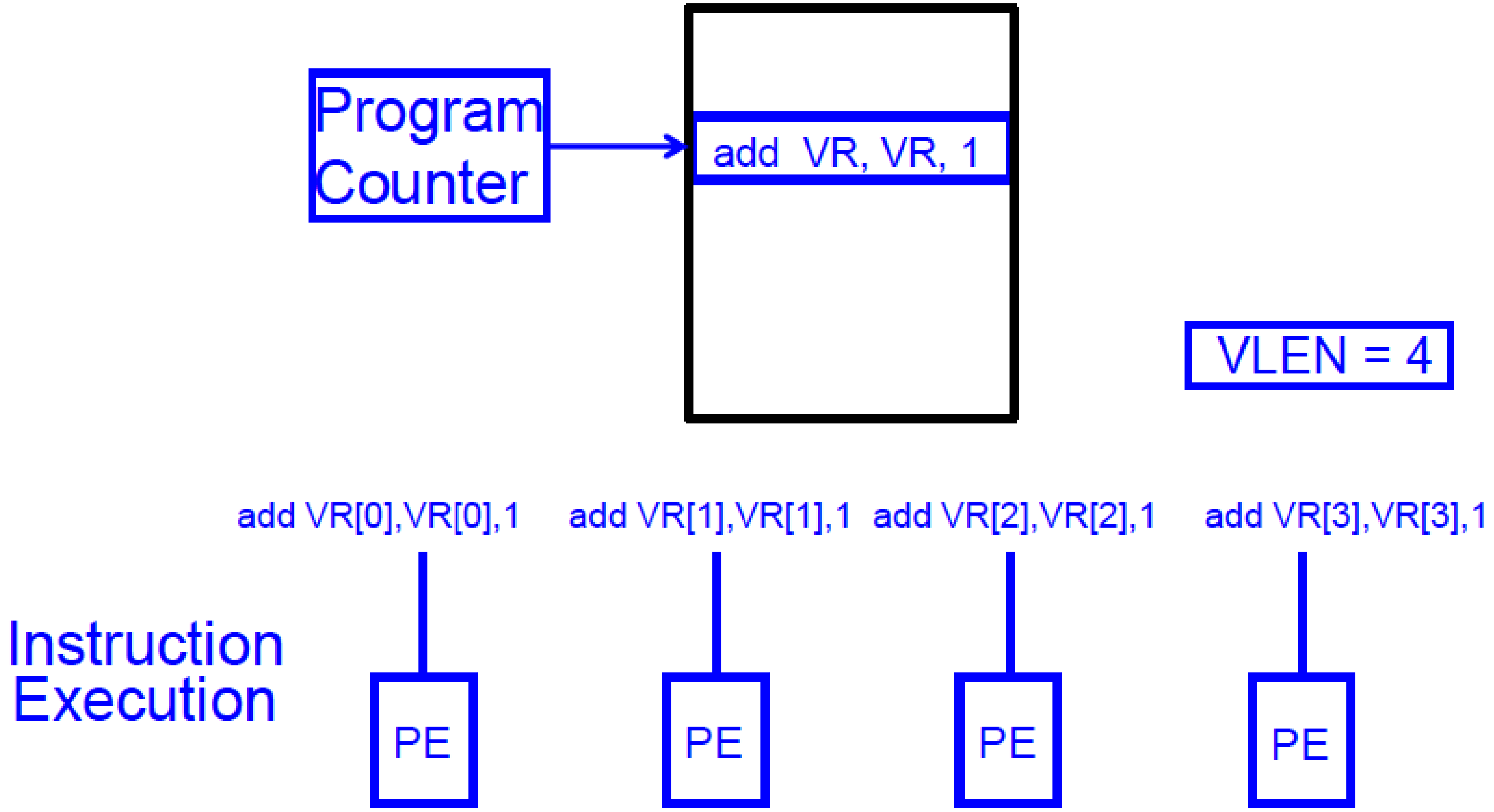
SIMD Array Processing vs. **VLIW**

- VLIW (Very Long Instruction Word): Multiple independent operations packed together by the compiler



SIMD Array Processing vs. VLIW

- Array processor: Single operation on multiple (different) data elements



Vector Processors (I)

- A vector is a one-dimensional array of numbers
- Many scientific/commercial programs use vectors

```
for (i = 0; i<=49; i++)
    C[i] = (A[i] + B[i]) / 2
```

- A vector processor is one whose instructions operate on vectors rather than scalar (single data) values
- Basic requirements
  - Need to load/store vectors → vector registers (contain vectors)
  - Need to operate on vectors of different lengths → vector length register (VLEN)
  - Elements of a vector might be stored apart from each other in memory → vector stride register (VSTR)
    - Stride: distance in memory between two elements of a vector

Vector Processors (II)

- A vector instruction performs an operation on each element in consecutive cycles
  - Vector functional units are pipelined
  - Each pipeline stage operates on a different data element

- Vector instructions allow deeper pipelines
  - No intra-vector dependencies → no hardware interlocking needed within a vector
  - No control flow within a vector
  - Known stride allows easy address calculation for all vector elements
    - Enables prefetching of vectors into registers/cache/memory

Vector Processor Advantages

+ No dependencies within a vector
  - Pipelining & parallelization work really well
  - Can have very deep pipelines, no dependencies!

+ Each instruction generates a lot of work
  - Reduces instruction fetch bandwidth requirements

+ Highly regular memory access pattern

+ No need to explicitly code loops
  - Fewer branches in the instruction sequence

Vector Processor Disadvantages

-- Works (only) if parallelism is regular (data/SIMD parallelism)

++ Vector operations

-- Very inefficient if parallelism is irregular

-- How about searching for a key in a linked list?

Vector Processor Limitations

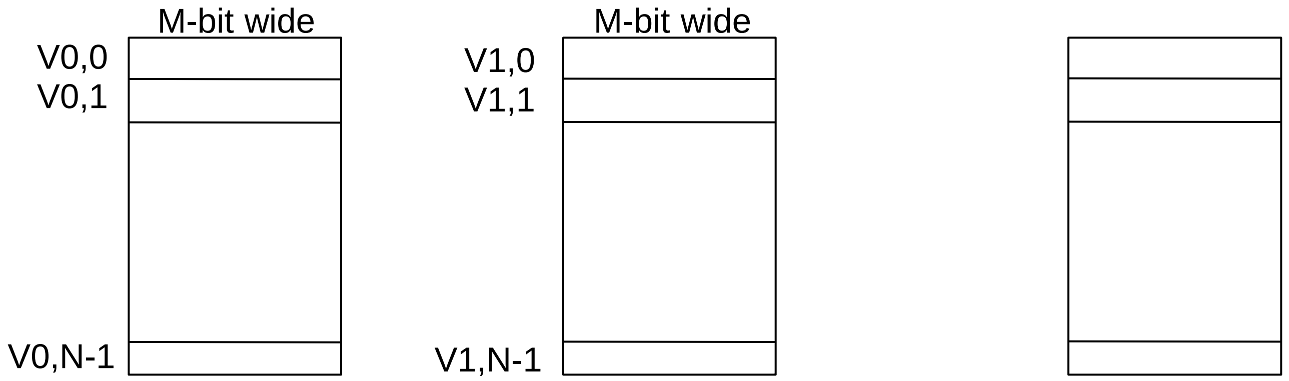
-- Memory (bandwidth) can easily become a bottleneck, especially if

1. compute/memory operation balance is not maintained
2. data is not mapped appropriately to memory banks

Vector Processing in More Depth

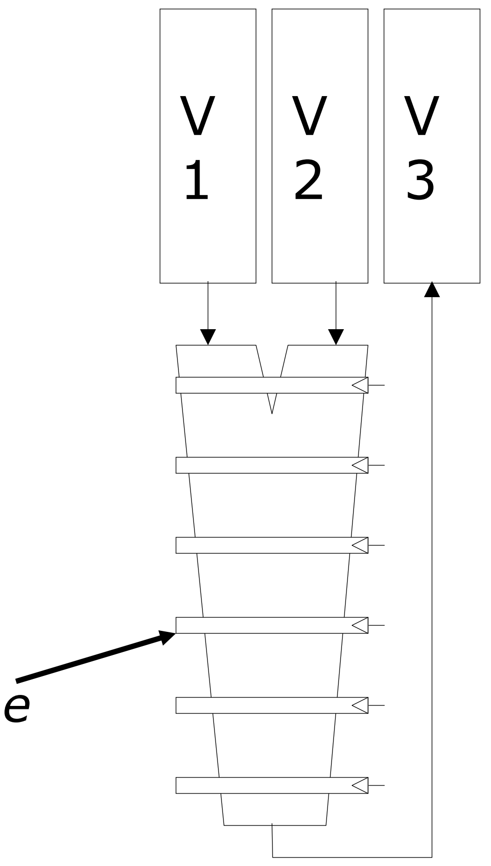
Vector Registers

- Each vector data register holds N M-bit values
- Vector control registers: VLEN, VSTR, VMASK
- Maximum VLEN can be N
  - Maximum number of elements stored in a vector register
- Vector Mask Register (VMASK)
  - Indicates which elements of vector to operate on
  - Set by vector test instructions
    - e.g., VMASK[i] = (V_k[i] == 0)



Vector Functional Units

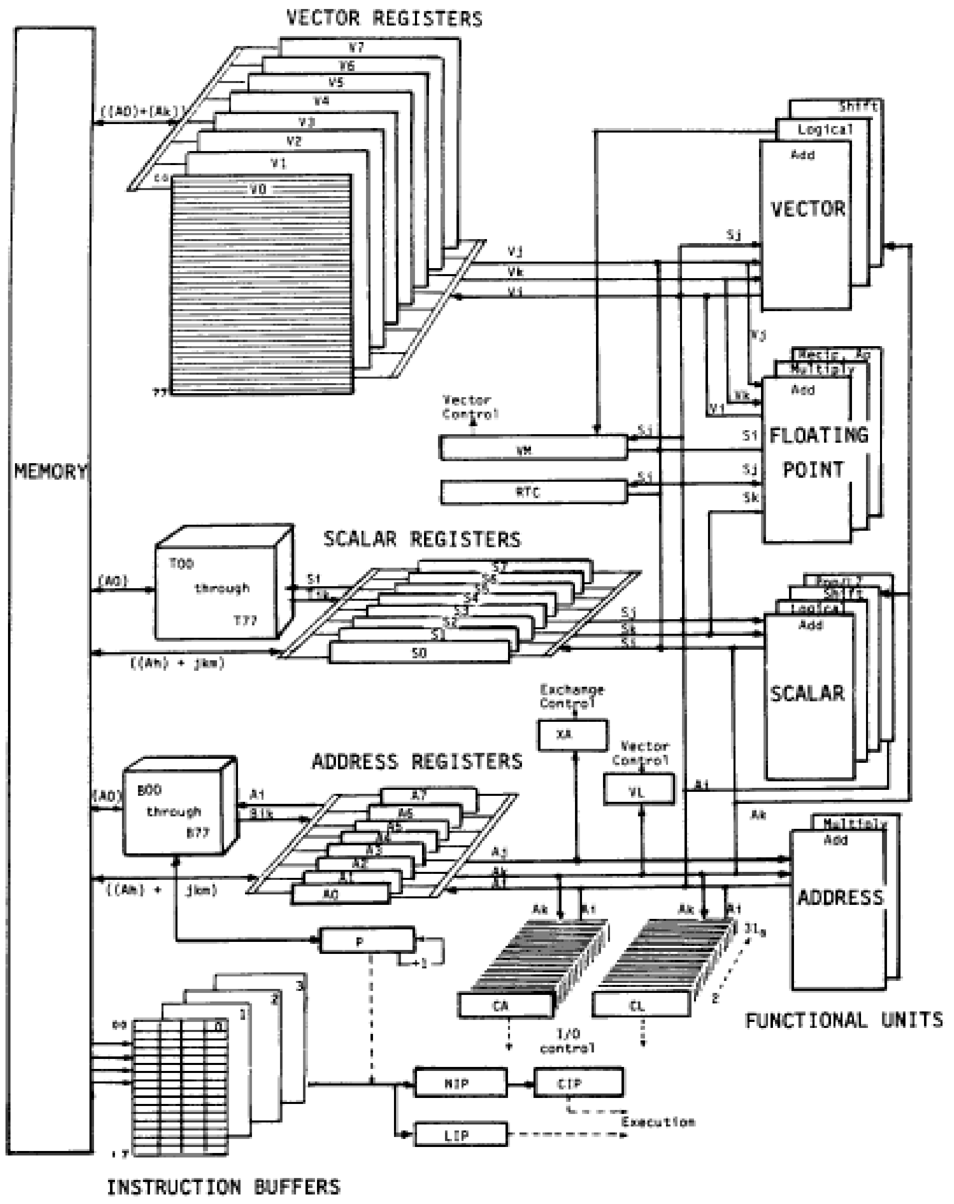
- Use a deep pipeline to execute element operations
  → fast clock cycle
- Control of deep pipeline is simple because elements in vector are independent



V1 * V2 → V3

Slide credit: Krste Asanovic

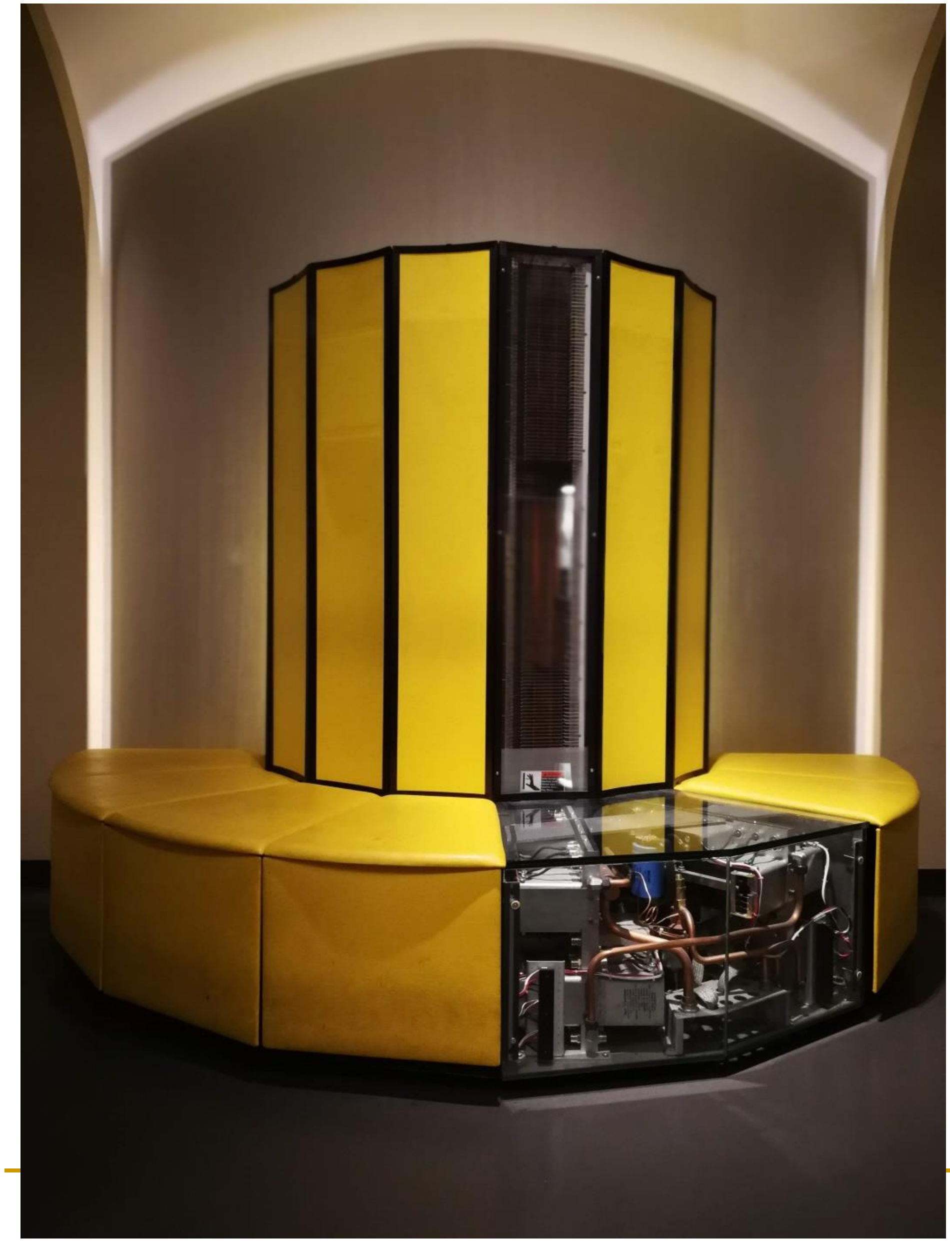
Vector Machine Organization (CRAY-1)

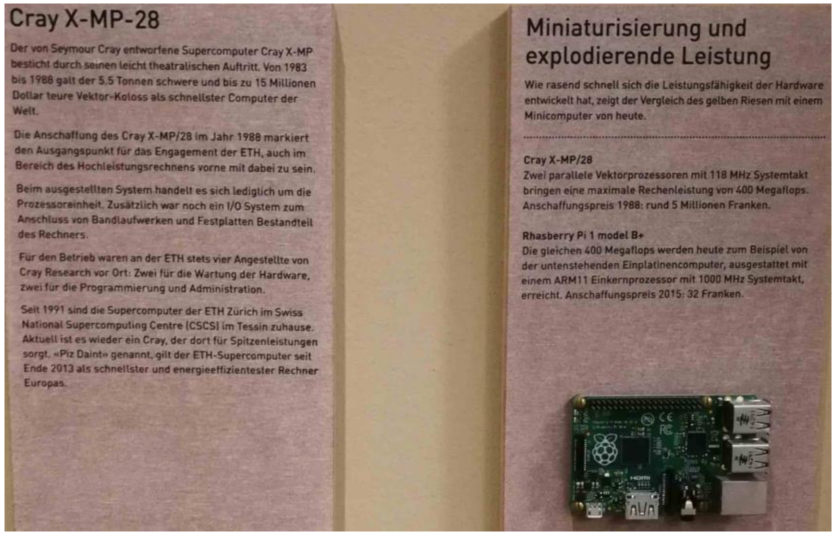


- CRAY-1
- Russell, "The CRAY-1 computer system," CACM 1978.

- Scalar and vector modes
- 8 64-element vector registers
- 64 bits per element
- 16 memory banks
- 8 64-bit scalar registers
- 8 24-bit address registers

CRAY X-MP-28 @ ETH (CAB, E Floor)





Cray X-MP-28

Der von Seymour Cray entworfene Supercomputer Cray X-MP besticht durch seinen leicht theatralischen Auftritt. Von 1983 bis 1988 galt der 5.5 Tonnen schwere und bis zu 15 Millionen Dollar teure Vektor-Koloss als schnellster Computer der Welt.

Die Anschaffung des Cray X-MP/28 im Jahr 1988 markiert den Ausgangspunkt für das Engagement der ETH, auch im Bereich des Hochleistungsrechnens vorne mit dabei zu sein.

Beim ausgestellten System handelt es sich lediglich um die Prozessoreinheit. Zusätzlich war noch ein I/O System zum Anschluss von Bandlaufwerken und Festplatten Bestandteil des Rechners.

Für den Betrieb waren an der ETH stets vier Angestellte von Cray Research vor Ort: Zwei für die Wartung der Hardware, zwei für die Programmierung und Administration.

Seit 1991 sind die Supercomputer der ETH Zürich im Swiss National Supercomputing Centre (CSCS) im Tessin zuhause. Aktuell ist es wieder ein Cray, der dort für Spitzenleistungen sorgt. «Piz Daint» genannt, gilt der ETH-Supercomputer seit Ende 2013 als schnellster und energieeffizientester Rechner Europas.

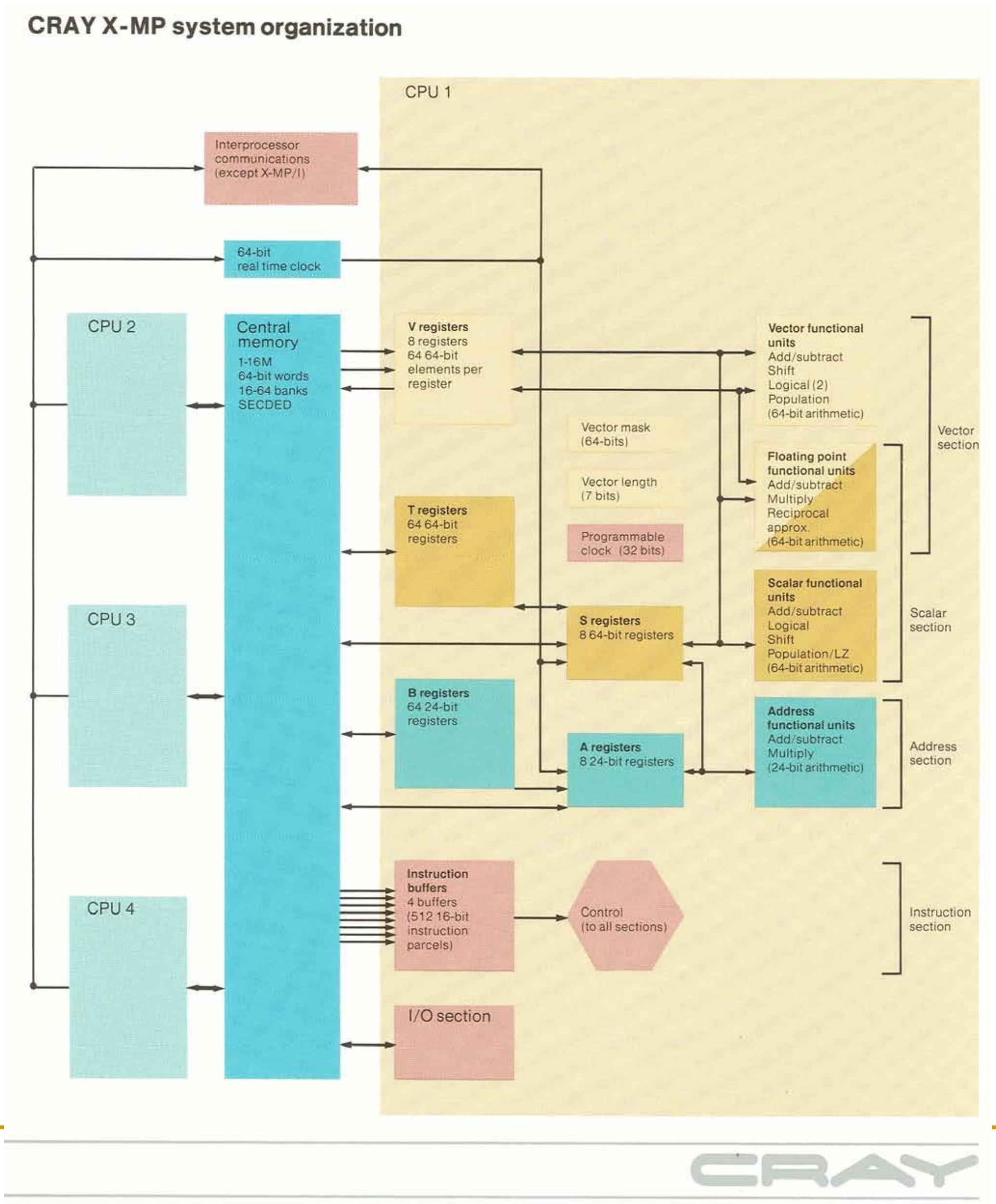
Miniaturisierung und explodierende Leistung

Wie rasend schnell sich die Leistungsfähigkeit der Hardware entwickelt hat, zeigt der Vergleich des gelben Riesen mit einem Minicomputer von heute.

Cray X-MP/28
Zwei parallele Vektorprozessoren mit 118 MHz Systemtakt bringen eine maximale Rechenleistung von 400 Megaflops. Anschaffungspreis 1988: rund 5 Millionen Franken.

Rhasberry Pi 1 model B+
Die gleichen 400 Megaflops werden heute zum Beispiel von der untenstehenden Einplatinencomputer, ausgestattet mit einem ARM11 Einkernprozessor mit 1000 MHz Systemtakt, erreicht. Anschaffungspreis 2015: 32 Franken.

CRAY X-MP System Organization

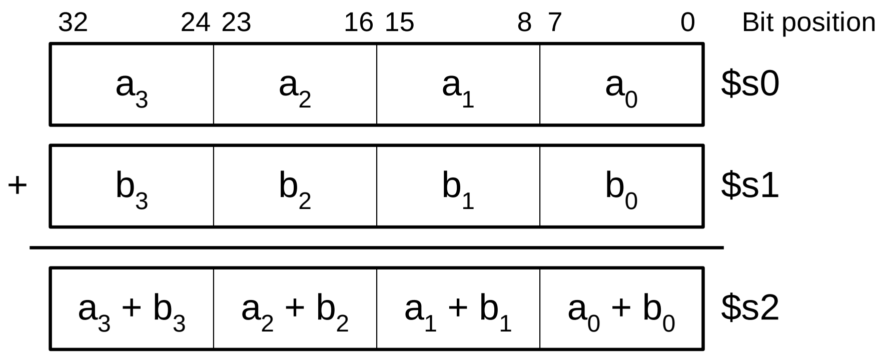

CRAY X-MP system organization
CPU 1
Interprocessor communications (except X-MP/1)
64-bit real time clock
CPU 2
CPU 3
CPU 4
Central memory 1-16M 64-bit words 16-64 banks SECDED
V registers 8 registers 64 64-bit elements per register
Vector mask (64-bits)
Vector length (7 bits)
Programmable clock (32 bits)
T registers 64 64-bit registers
S registers 8 64-bit registers
B registers 64 24-bit registers
A registers 8 24-bit registers
Vector functional units Add/subtract Shift Logical (2) Population (64-bit arithmetic)
Floating point functional units Add/subtract Multiply Reciprocal approx. (64-bit arithmetic)
Scalar functional units Add/subtract Logical Shift Population/LZ (64-bit arithmetic)
Address functional units Add/subtract Multiply (24-bit arithmetic)
Vector section
Scalar section
Address section
Instruction buffers 4 buffers (512 16-bit instruction parcels)
Control (to all sections)
Instruction section
I/O section
CRAY

Cray Research Inc., “The CRAY X-MP Series of Computer Systems,” 1985

SIMD ISA Extensions

- Single Instruction Multiple Data (SIMD) extension instructions
  - Single instruction acts on multiple pieces of data at once
  - Common application: graphics
  - Perform short arithmetic operations (also called *packed arithmetic*)
- For example: add four 8-bit numbers
- Must modify ALU to eliminate carries between 8-bit values

```
padd8 $s2, $s0, $s1
```



Intel Pentium MMX Operations

- Idea: One instruction operates on multiple data elements simultaneously
  - *À la* array processing (yet much more limited)
  - Designed with multimedia (graphics) operations in mind

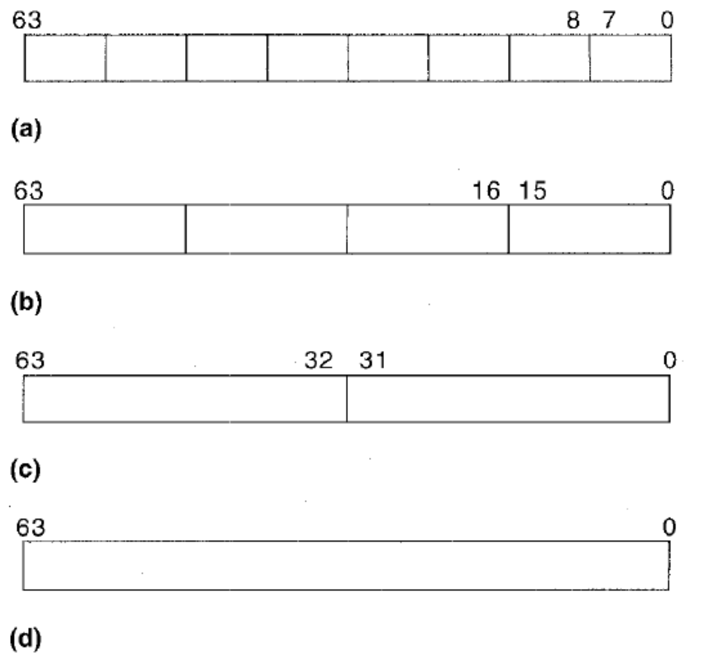


Figure 1. MMX technology data types: packed byte (a), packed word (b), packed doubleword (c), and quadword (d).

No VLEN register
Opcode determines data type:
8 8-bit bytes
4 16-bit words
2 32-bit doublewords
1 64-bit quadword

Stride is always equal to 1.

Peleg and Weiser, "MMX Technology Extension to the Intel Architecture," IEEE Micro, 1996.

MMX Example: Image Overlaying (I)

- Goal: Overlay the human in image 1 on top of the background in image 2

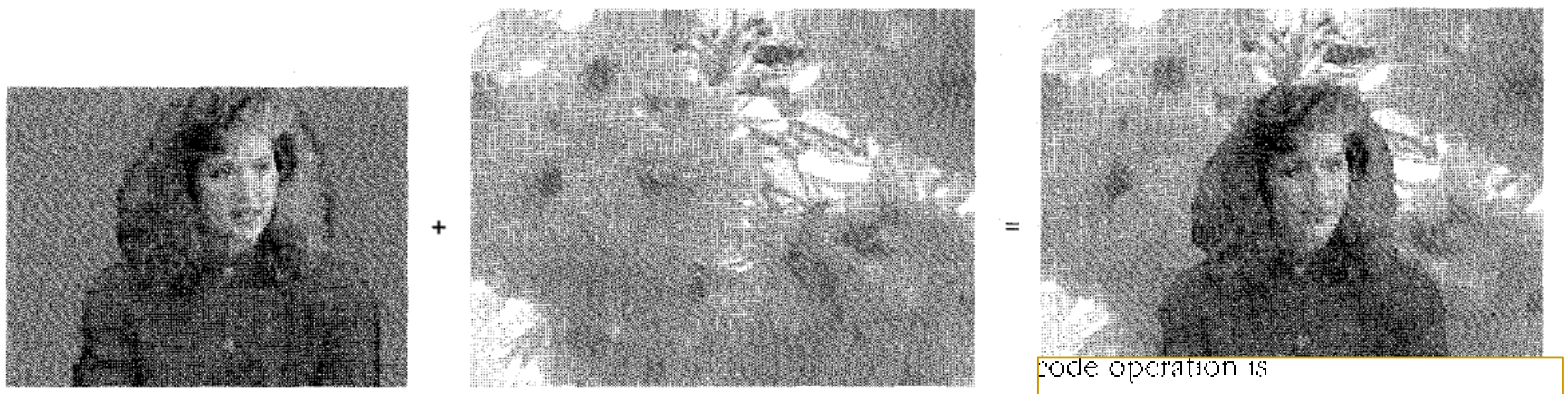


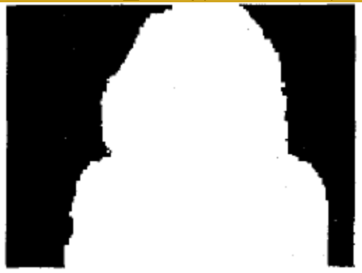
Figure 8. Chroma keying: image overlay using a background color.

code operation is

```
for (i=0; i<image_size; i++) {
if (x[i] == Blue) new_image[i] =y[i];
    else new_image[i] = x[i];
```

PCMPEQB MM1, MM3

MM1	Blue	Blue	Blue	Blue	Blue	Blue	Blue	Blue
MM3	X7!=blue	X6!=blue	X5=blue	X4=blue	X3!=blue	X2!=blue	X1=blue	X0=blue
MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF

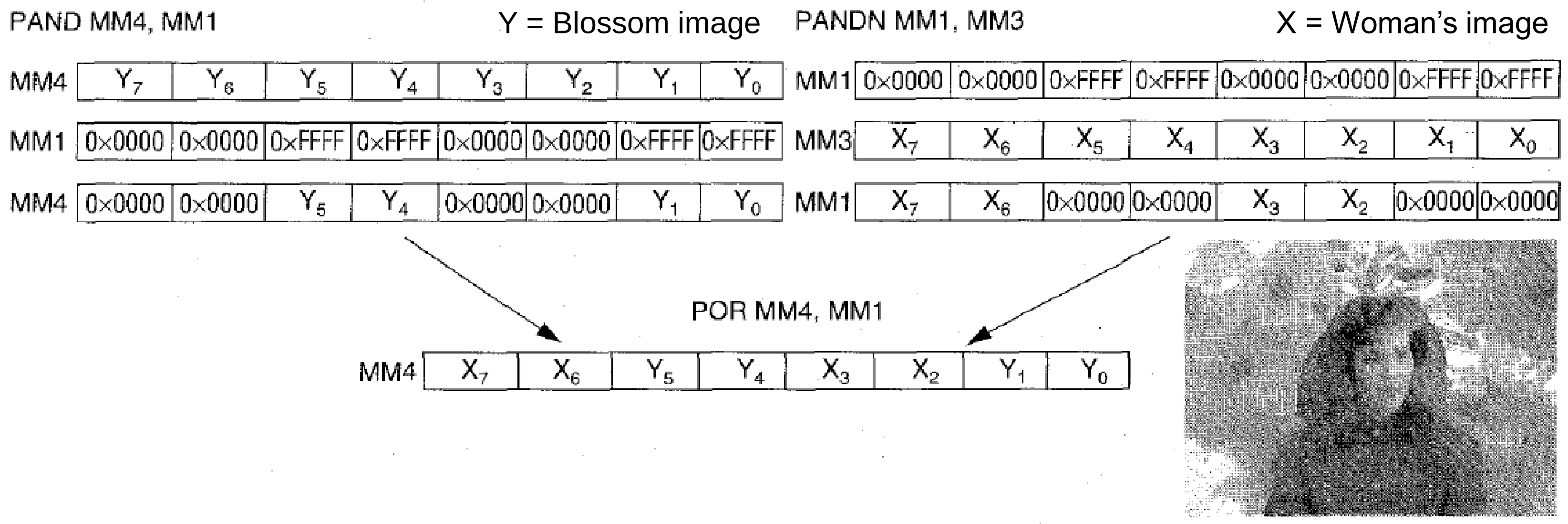


Bitmask

Figure 9. Generating the selection bit mask.

Peleg and Weiser, "MMX Technology Extension to the Intel Architecture," IEEE Micro, 1996.

MMX Example: Image Overlaying (II)



MM4	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF
MM4	0x0000	0x0000	Y_5	Y_4	0x0000	0x0000	Y_1	Y_0

MM1	0x0000	0x0000	0xFFFF	0xFFFF	0x0000	0x0000	0xFFFF	0xFFFF
MM3	X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0
MM1	X_7	X_6	0x0000	0x0000	X_3	X_2	0x0000	0x0000

MM4	X_7	X_6	Y_5	Y_4	X_3	X_2	Y_1	Y_0

Figure 10. Using the mask with logical MMX instructions to perform a conditional select.

```
Movq       mm3, mem1    /* Load eight pixels from
                           woman's image
Movq       mm4, mem2    /* Load eight pixels from the
                           blossom image
Pcmpeqb    mm1, mm3
Pand       mm4, mm1
Pandn      mm1, mm3
Por        mm4, mm1
```

Figure 11. MMX code sequence for performing a conditional select.

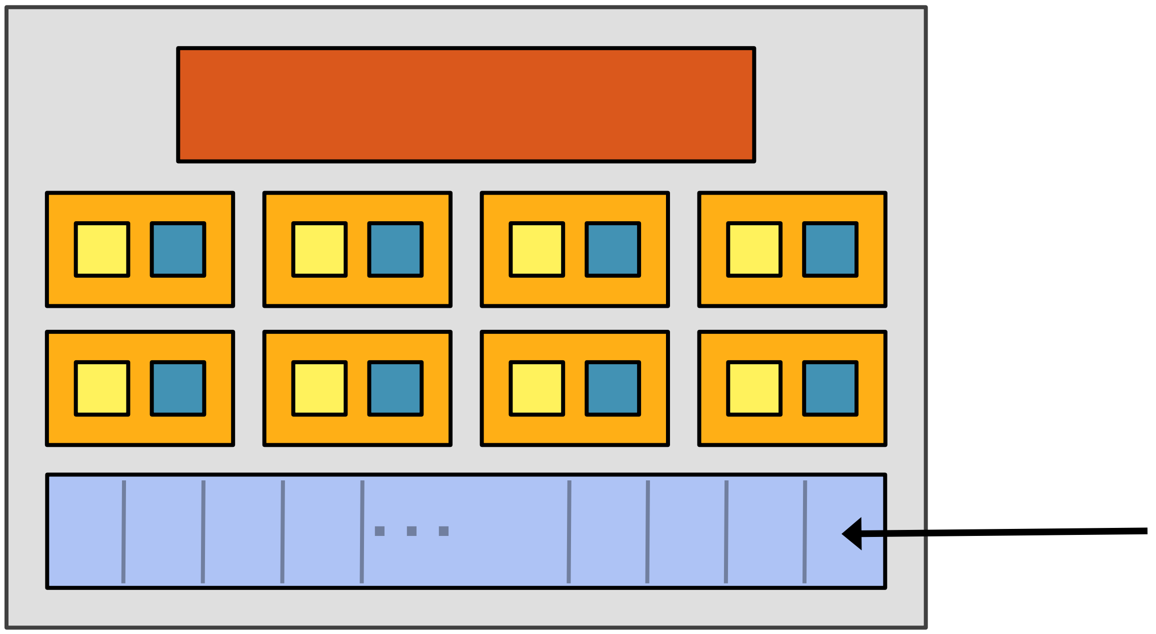
Peleg and Weiser, "MMX Technology Extension to the Intel Architecture," IEEE Micro, 1996.

NVIDIA GeForce GTX 285

- NVIDIA-speak:
  - 240 stream processors
  - “SIMT execution”

- Generic speak:
  - 30 cores
  - 8 SIMD functional units per core

NVIDIA GeForce GTX 285 “core”



64 KB of storage for thread contexts (registers)

🟧 = SIMD functional unit, control shared across 8 units

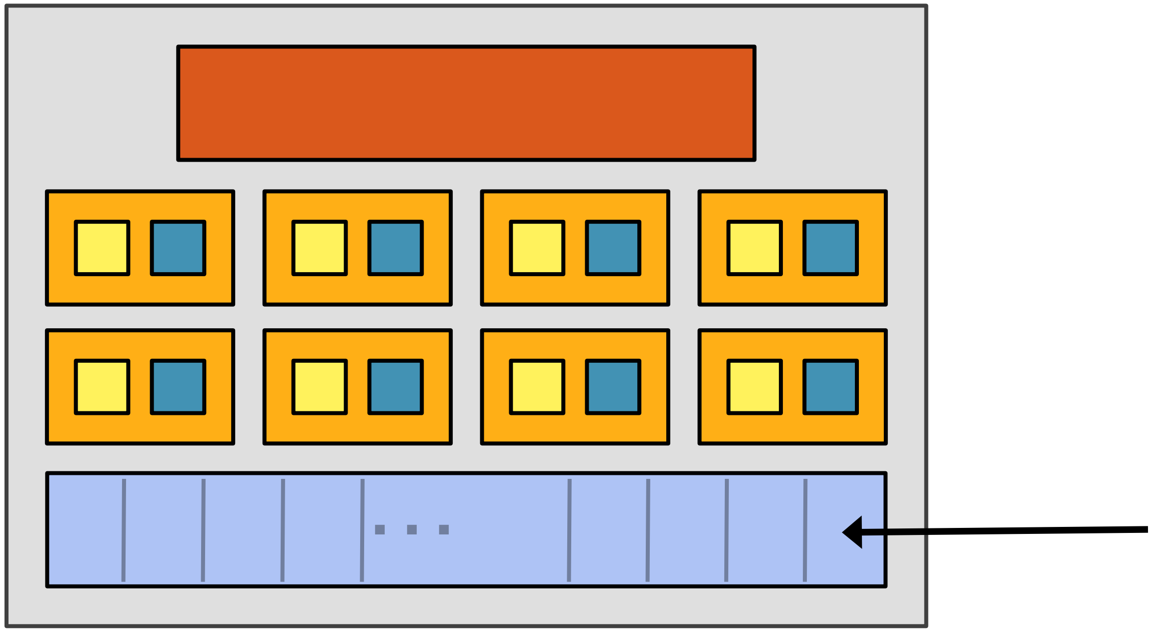
🟨 = multiply-add

🟦 = multiply

🟥 = instruction stream decode

🟦 = execution context storage

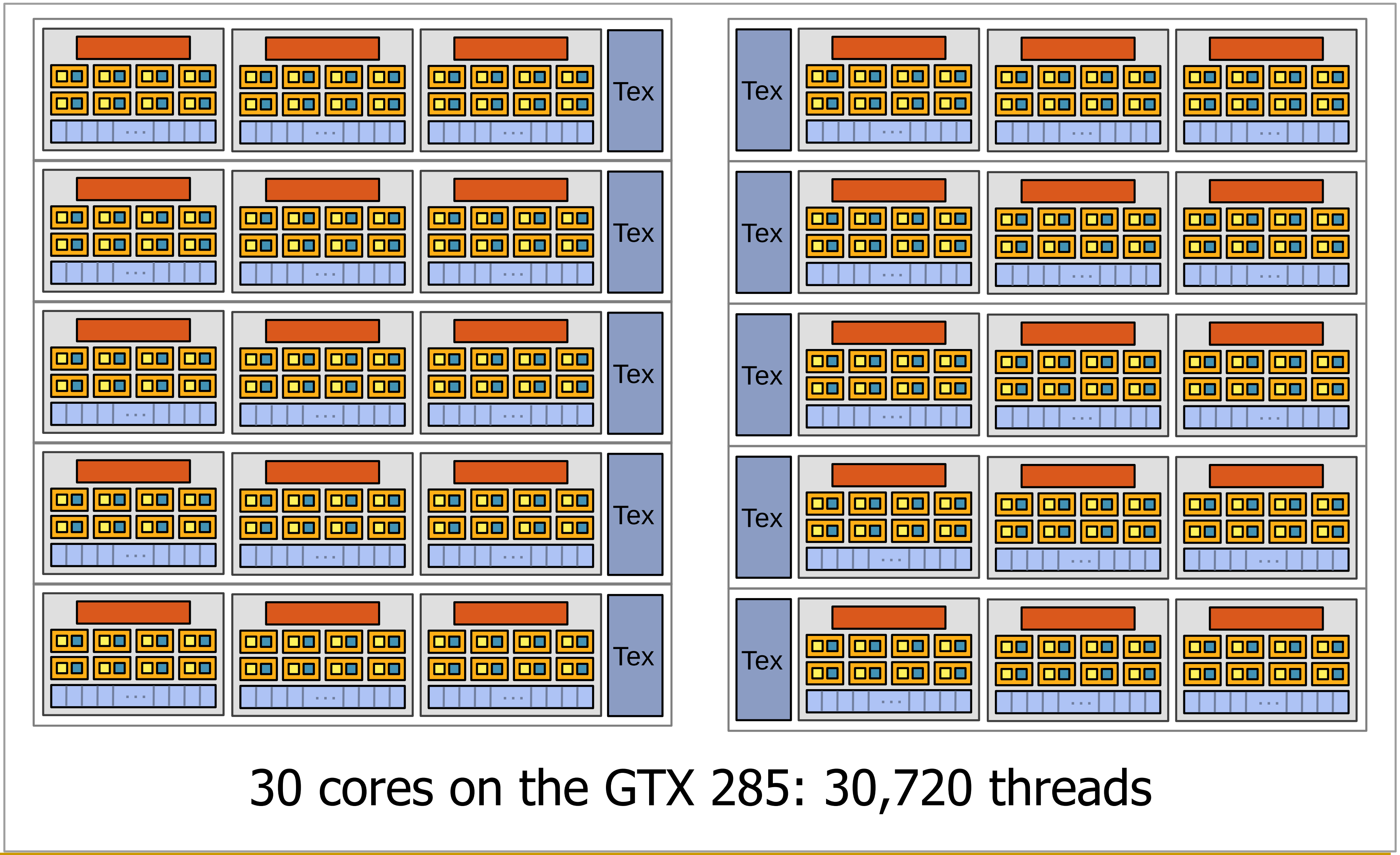
NVIDIA GeForce GTX 285 “core”



64 KB of storage for thread contexts (registers)

- Groups of 32 threads share instruction stream (each group is a Warp)
- Up to 32 warps are simultaneously interleaved
- Up to 1024 thread contexts can be stored

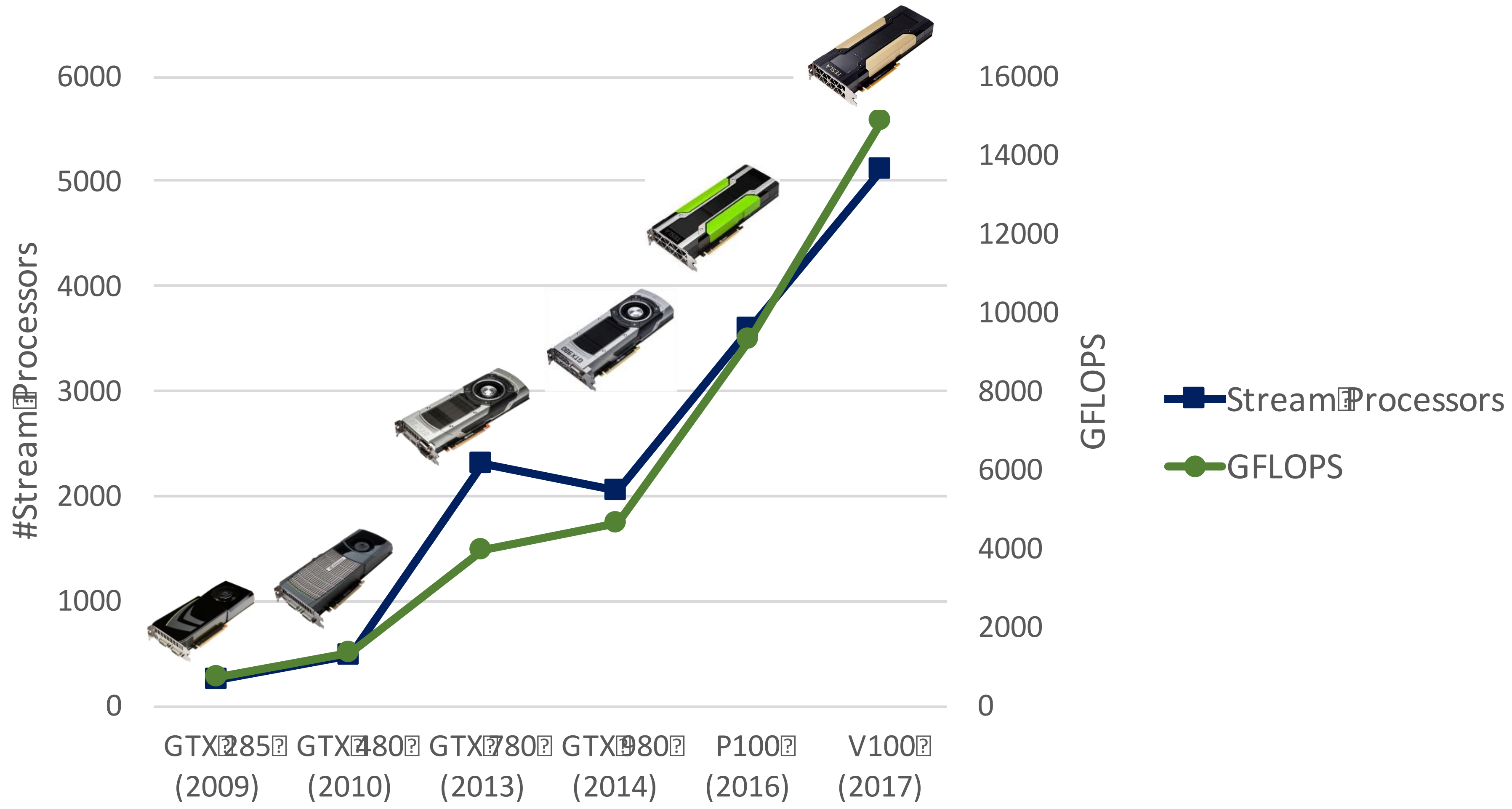
NVIDIA GeForce GTX 285



30 cores on the GTX 285: 30,720 threads

Slide credit: Kayvon Fatahalian

Evolution of NVIDIA GPUs



NVIDIA V100

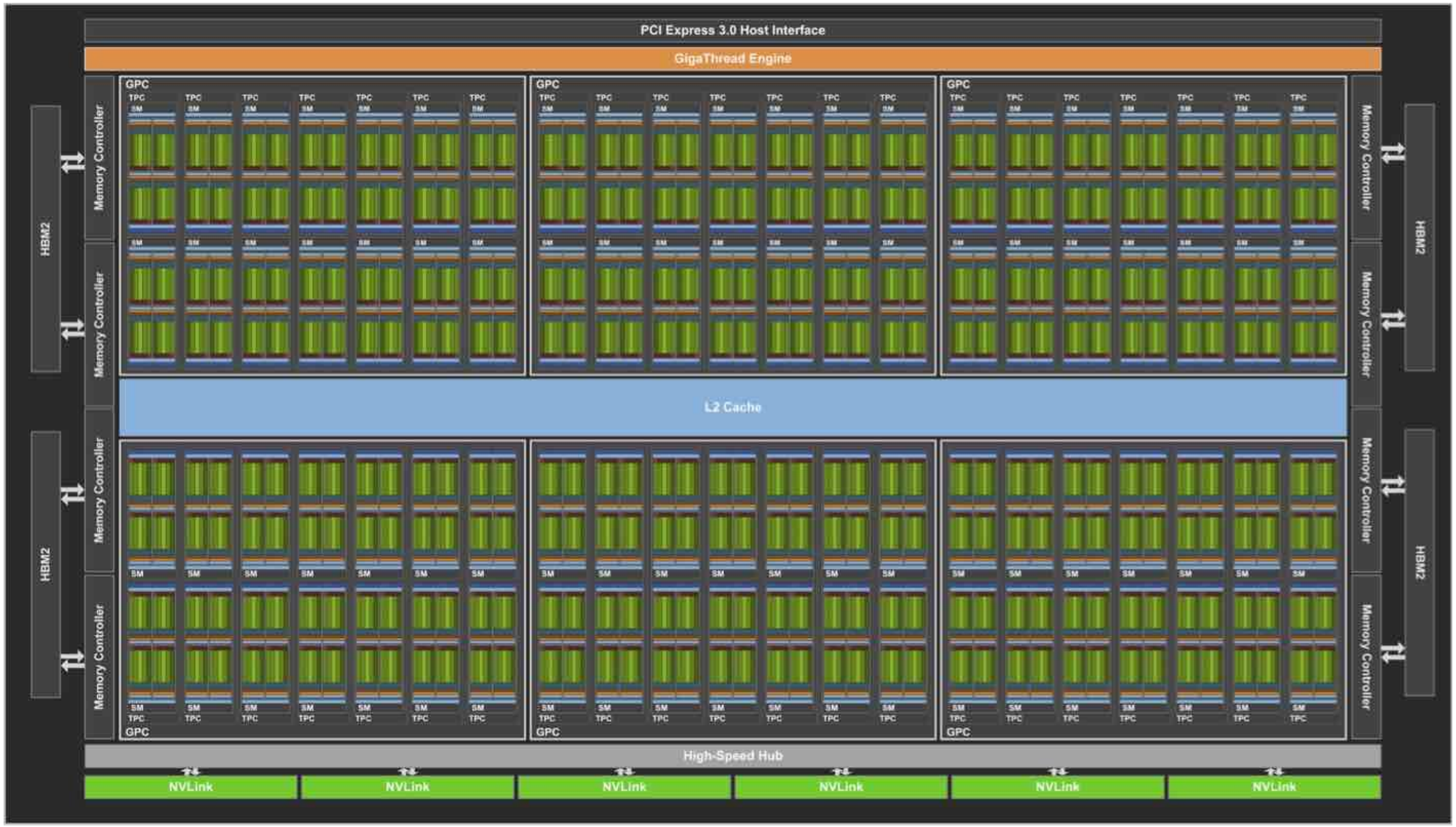
- NVIDIA-speak:
  - 5120 stream processors
  - "SIMT execution"

- Generic speak:
  - 80 cores
  - 64 SIMD functional units per core

  - Tensor cores for Machine Learning

- NVIDIA, "NVIDIA Tesla V100 GPU Architecture. White Paper," 2017.

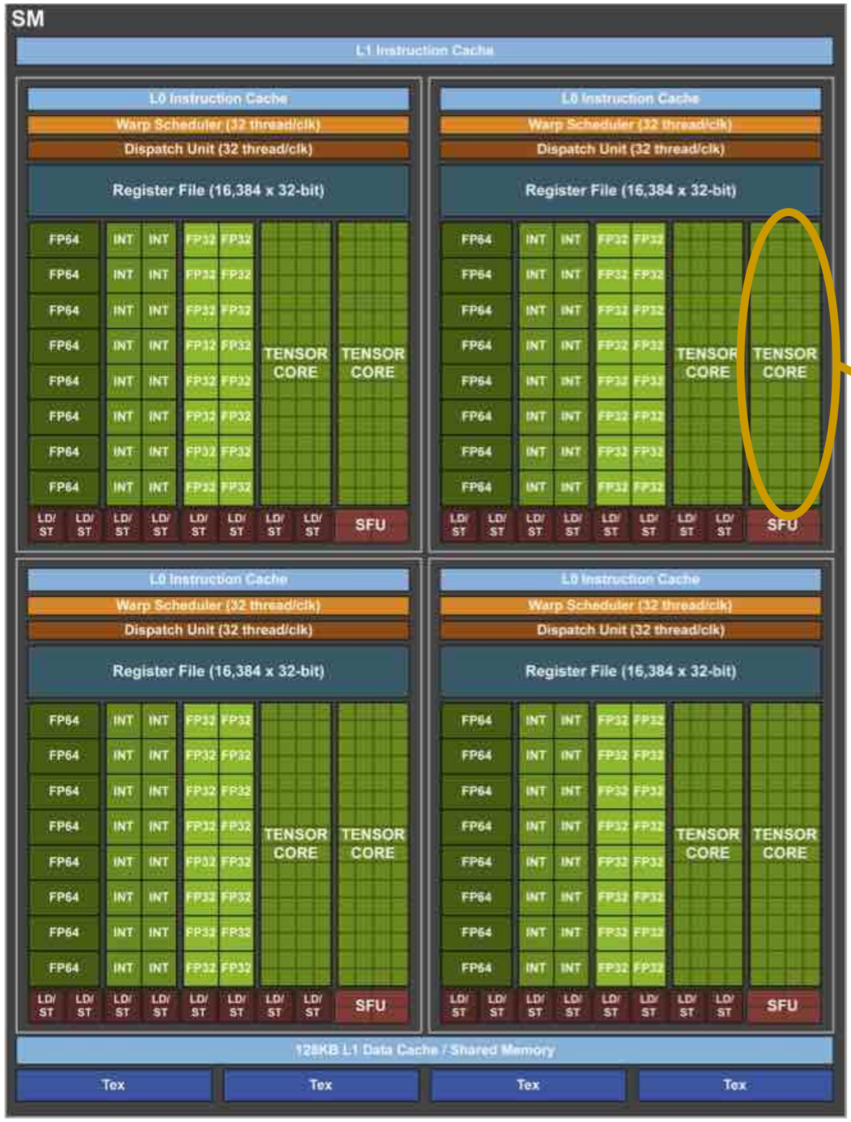
NVIDIA V100 Block Diagram



https://devblogs.nvidia.com/inside-volta/

80 cores on the V100

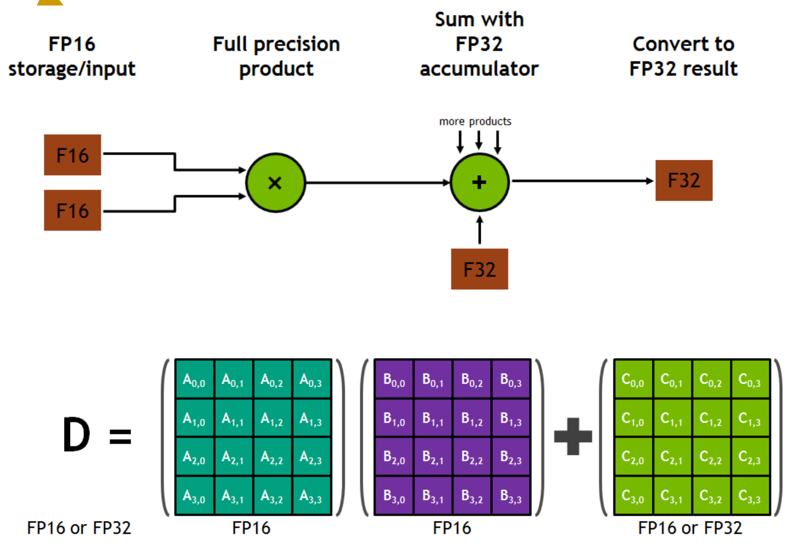
NVIDIA V100 Core


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

15.7 TFLOPS Single Precision

7.8 TFLOPS Double Precision

125 TFLOPS for Deep Learning (Tensor cores)



$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

https://devblogs.nvidia.com/inside-volta/

Food for Thought

- What is the main bottleneck in GPU programs?

- "Tensor cores":
  - Can you think about other operations than matrix multiplication?
  - What other applications could benefit from specialized cores?

- Compare and contrast GPUs vs other accelerators (e.g., systolic arrays)

  - Which one is better for machine learning?

  - Which one is better for image/vision processing?

  - What types of parallelism each one exploits?

  - What are the tradeoffs?

Clarification of some GPU Terms

Generic Term	NVIDIA Term	AMD Term	Comments
Vector length	Warp size	Wavefront size	Number of threads that run in parallel (lock-step) on a SIMD functional unit
Pipelined functional unit / Scalar pipeline	Streaming processor / CUDA core	-	Functional unit that executes instructions for one GPU thread
SIMD functional unit / SIMD pipeline	Group of N streaming processors (e.g., N=8 in GTX 285, N=16 in Fermi)	Vector ALU	SIMD functional unit that executes instructions for an entire warp
GPU core	Streaming multiprocessor	Compute unit	It contains one or more warp schedulers and one or several SIMD pipelines