Technical University of Munich

Lecture 1: Introduction to GPUs

Informatik elective: GPU Computing

Pratik Nayak

Course Objectives

Theory and basics

- Learn about GPUs:
  - *WHY* are they useful ?
  - *WHEN* are they useful ?
- Parallel programming concepts
- GPU hardware architecture
- GPU programming models
- Efficient GPU algorithms and data-structures

Practical know-how

- Develop GPU programming skills
- Translate algorithms to GPU code
- Analyze GPU code performance
- Reason about algorithms and data-structures suitable for GPUs
- Use GPU libraries

Course information

- Course name: GPU Computing (CITHN4015)
- Lectures every Thursday (excepting holidays): 12:15 to 13:45
- Exercise session every Thursday (excepting holidays): 14:15 to 15:45
- Grading:
  - Final exam (date to be announced later) (IN-PERSON only, and in Campus Heilbronn)
  - Exercise sheets: for grade bonus (deadlines will vary, but generally 1 week of work-time)
- Credits: 6 ECTS
- Course instructors: Pratik Nayak (pratik.nayak@tum.de) and Hartwig Anzt (hartwig.anzt@tum.de)
- Reference: CUDA Programming Guide

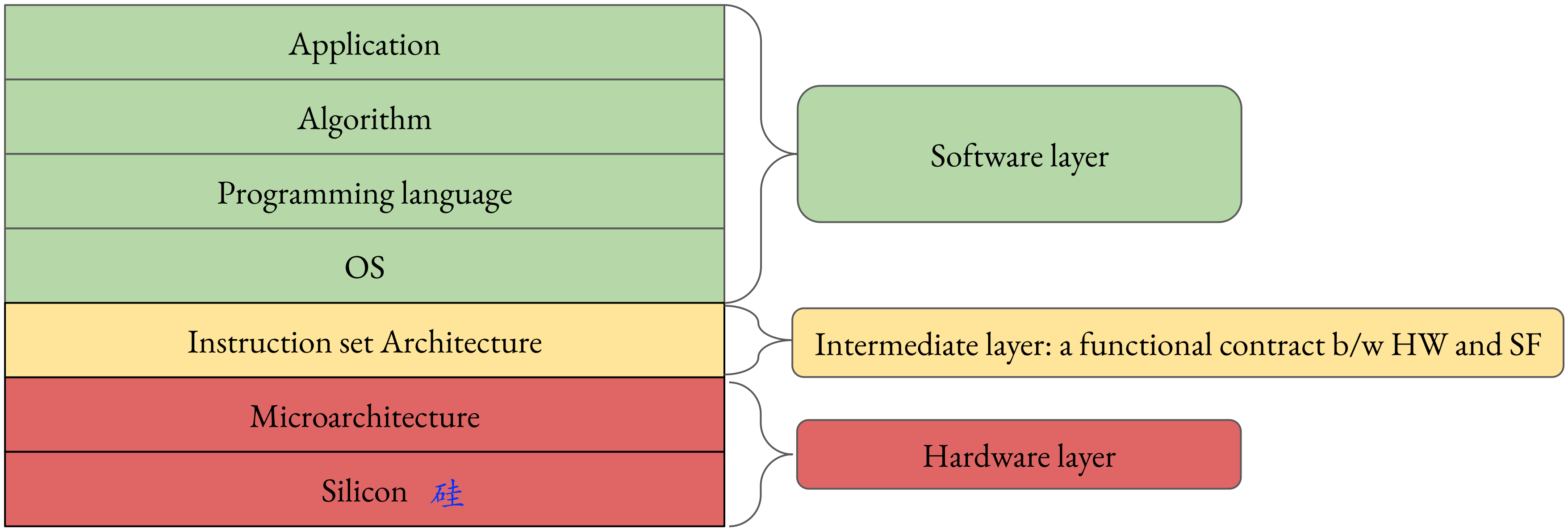
A starter quiz

- Go to menti.com and use code 7740 1049
- Fill this form if you want access to a cluster with GPUs
  - https://forms.gle/uRPNsxyVNSLiCr8h8

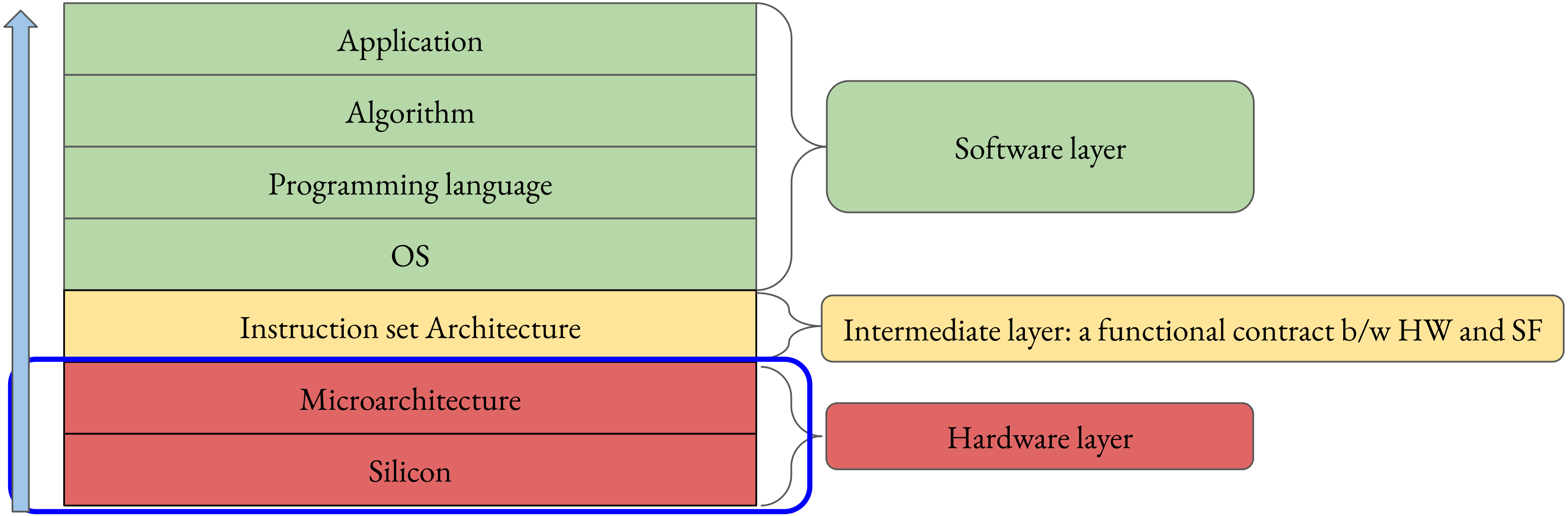
In this session

- Abstract layers in computing
- Recall: Basic terminologies, computer microarchitecture and estimating performance
- Computer architecture taxonomies
- GPU basics and differences to CPUs
- When are GPUs useful ?
- A look at different GPU applications.

Back to basics: Abstract layers in computing

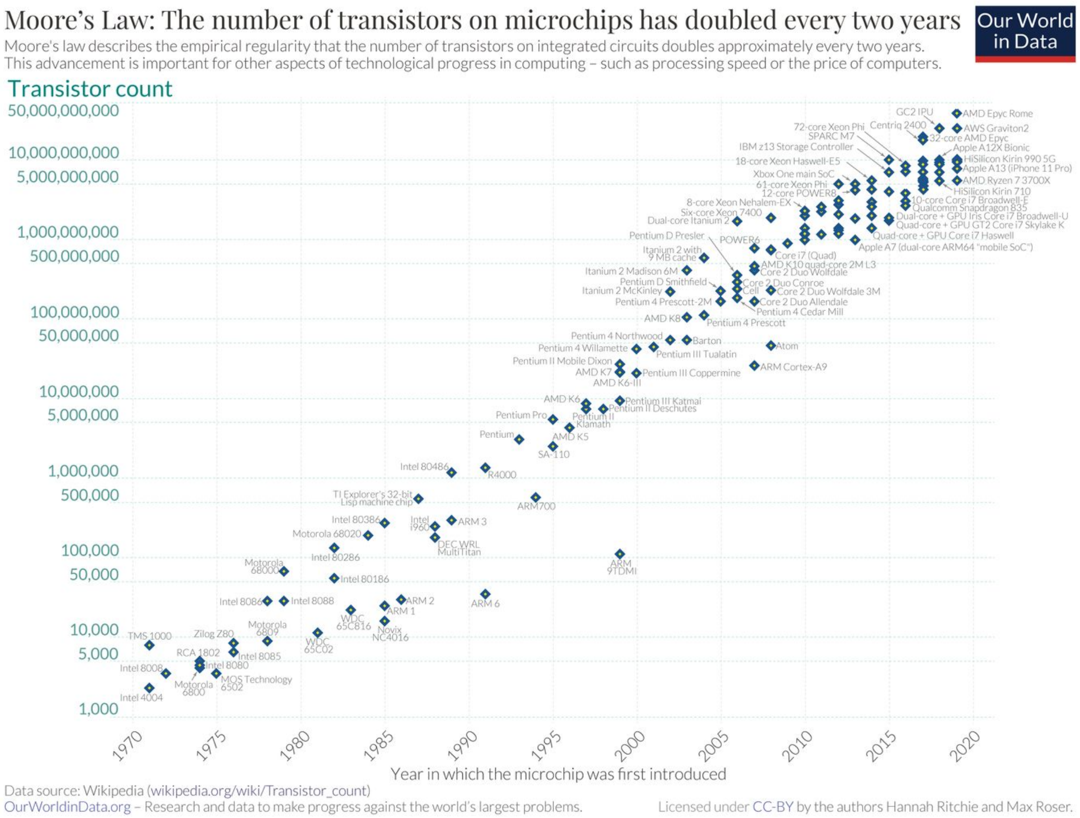


Back to basics: Abstract layers in computing

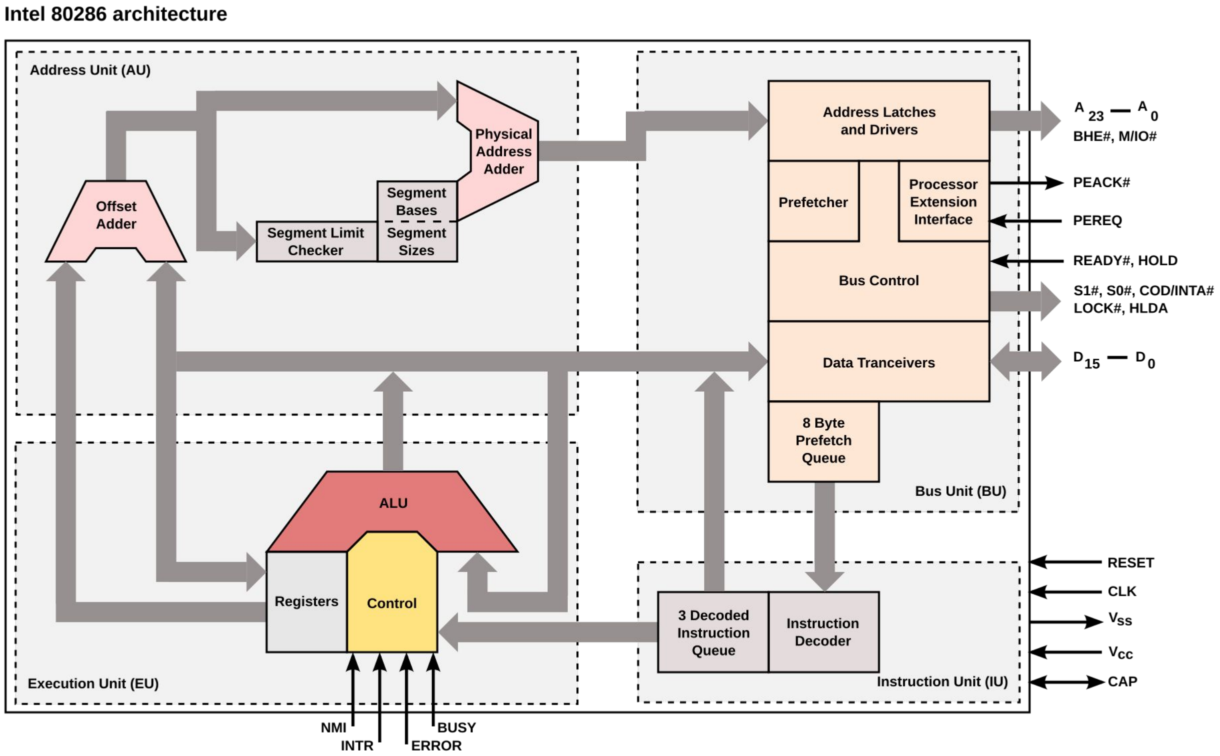


Back to basics: Transistors

- Essentially switches that combined together can perform boolean functions: AND, OR, XOR
- The number of transistors has increased in a regular fashion.
- Largest processors can have somewhere around 50-60 billion transistors, and maybe more.

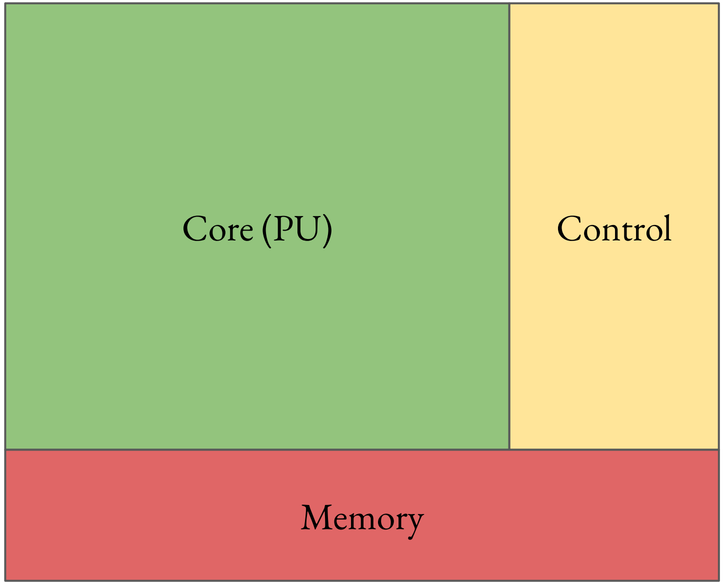


Back to basics: Microarchitecture



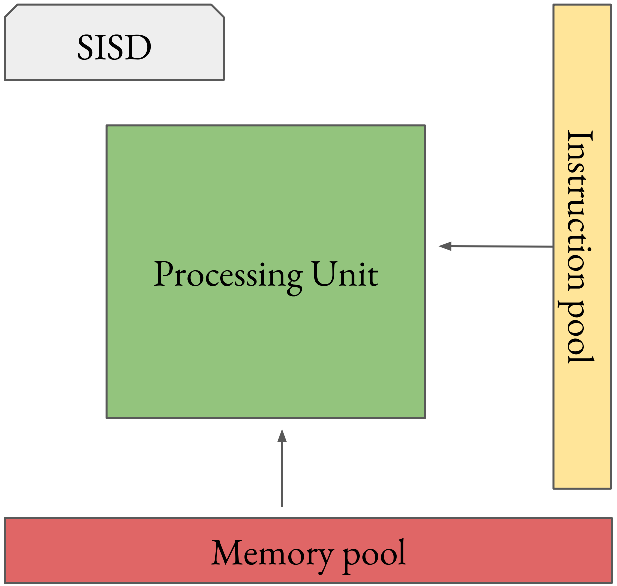
Back to basics: Microarchitecture (Simplified)

- Memory:
  - Store data and instructions
  - Intermediate storage between compute cycles
- Control:
  - Fetch instructions from memory
  - Fetch data from memory and load into registers
- Core/Processing Unit:
  - Do the actual computations according to the fetched instructions.
  - Consists of Arithmetic and logic units (ALU).



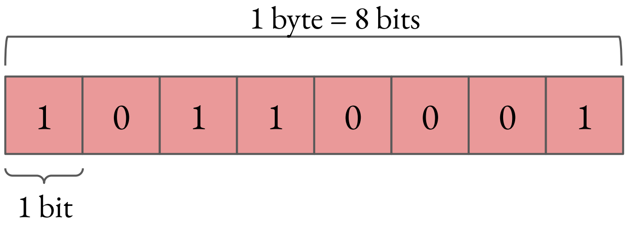
Single Instruction Single Data (SISD) Microarchitecture

- Instructions are sent from memory module to the control unit
- Control unit decodes the instructions, and sends them to the PU.
- Processing unit processes the data from the memory module, processes it and sends the processed data back to the memory module.
- Examples: Pipelined processors, superscalar processors



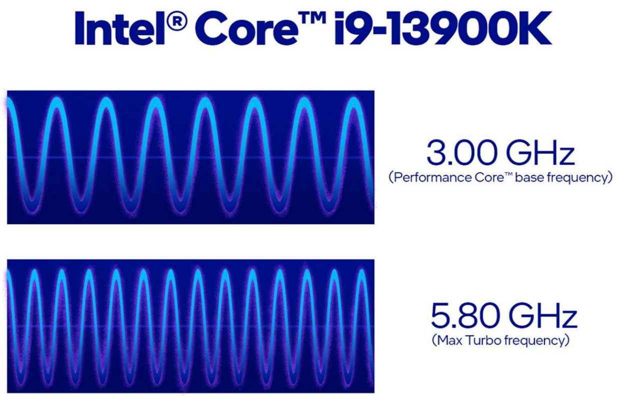
Basic terminology: Data, bits and bytes

- Data:
  - *Bit*: Smallest unit of storage (0 or 1, binary)
  - *Byte*: 1 byte = 8 bits
- Metric:
  - *Bandwidth* (BW): rate of data transfer, usually measure in (bytes/s)



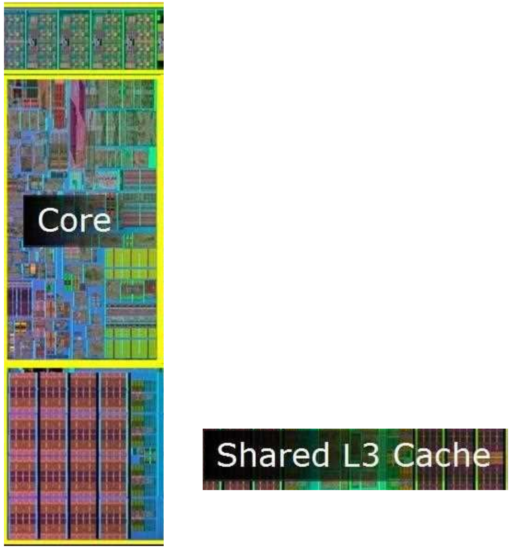
Basic terminology: Clock speed and Flop/s

- Computation:
    - *Clock speed*: Measured in Hertz (Hz), number of clock cycles per second.
    - *instructions/cycle*: Number of instructions in one clock cycle.
    - *Op/s*: Number of operations per second.
    - *Flop/s*: Number of floating point operations per second



Basic terminology: Thread and Core

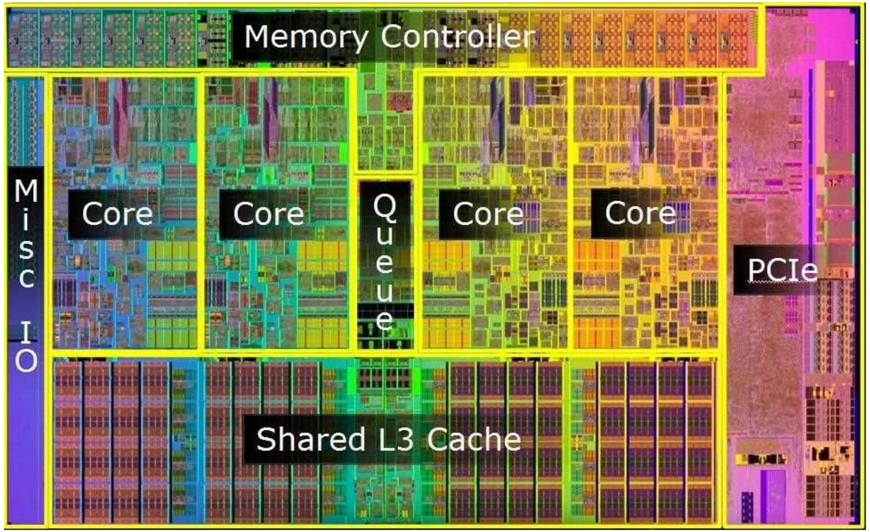
- *Thread (software-level)*: “thread of execution”: an ordered sequence of instructions (software)
- *Core (hardware-level)*: One processor within a CPU die (hardware).



[Intel multi-core CPU]

Basic terminology: Thread and Core

- *Thread (software-level)*: "thread of execution": an ordered sequence of instructions (software)
- *Core (hardware-level)*: One processor within a CPU die (hardware).
- *Multi-core (hardware-level)*: Multiple processors capable of independent execution within one CPU die

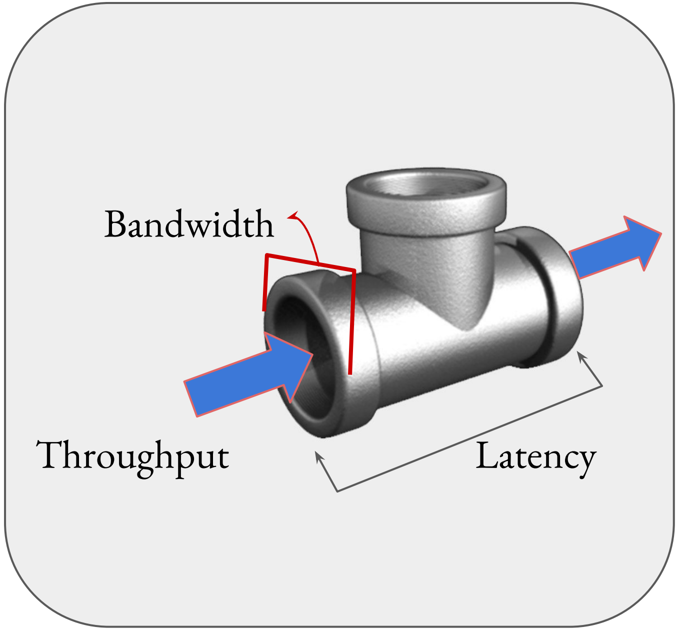


[Intel multi-core CPU]

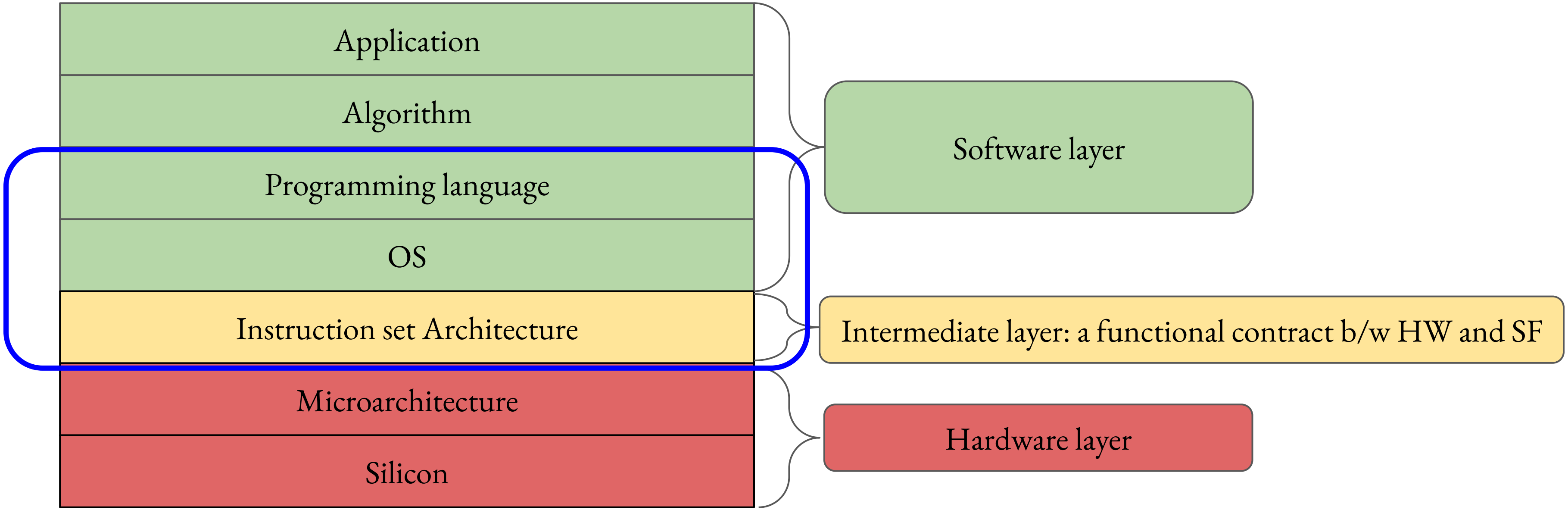
Basic terminology: Throughput, latency and bandwidth

- ***Throughput***: A measure of effective output over time.
- ***Latency***: A measure of delay in a system, duration it takes for data to reach from point A to point B.
- ***Bandwidth***: A measure of the capacity.

Metric	Optimal
Throughput	Higher is better ↑
Latency	Lower is better ↓
Bandwidth	Higher is better ↑



Back to basics: Abstract layers in computing



Back to basics: Abstract layers in computing

- High-level language:
  - Abstraction for productivity and portability
  - Closer to application and algorithm

```
int square(int a){
    return a*a;
}
```

- Assembly language
  - Textual representation of instructions (ISA)

```
square(int):
    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-4], edi
    mov     eax, DWORD PTR [rbp-4]
    imul    eax, eax
    pop     rbp
    ret
```

- Hardware representation
  - 1s and 0s (Binary representation) input to the microarchitecture

```
…100101000101000010101000010101010100101010001
01000001010101010100010101010100000010101010
1010000000000…000101010101010111111111111100
000000000000000000000000000010101010001010100
0100000000000011101000000000000010101010010100
0100000000000000001011001010100…
```

Microarchitecture

Back to basics: Example instructions in ISA

- Arithmetic and logic instructions:
  - `mul, add, fma` …
- Data movement instructions:
  - `mov, push, pop` …
- Control flow instructions:
  - `jump, cmp, call, return` …

```
square(int):
    push    rbp
    mov     rbp, rsp
    mov     DWORD PTR [rbp-4], edi
    mov     eax, DWORD PTR [rbp-4]
    imul    eax, eax
    pop     rbp
    ret
```

Back to basics: Example instructions in ISA

- Ari
  -
- D
  -
- C
  -

This is a small subset of the instructions. For a more complete list, see for example (x86): https://www.felixcloutier.com/x86/

Estimating performance

Considering just the CPU, we can estimate the time for

some program with:

$$CPU\ Time = \frac{Seconds}{Program}$$

$$CPU\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Basic terminology: Estimating performance

Considering just the CPU, we can estimate the time for some program with:

$$CPU\ Time = \frac{Seconds}{Program}$$

$$CPU\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

Hardware

To improve performance:

$\frac{Instructions}{Program}$ Reduce

$\boxed{\frac{Cycles}{Instruction} \text{ Reduce} \qquad \frac{Seconds}{Cycle} \text{ Reduce}}$

Basic terminology: Estimating performance

Considering just the CPU, we can estimate the time for

some program with:

$$CPU\ Time = \frac{Seconds}{Program}$$

$$CPU\ Time = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

To improve performance:

$$\frac{Instructions}{Program}$$

Reduce

$$\frac{Cycles}{Second}$$

Increase

Increase clock frequency

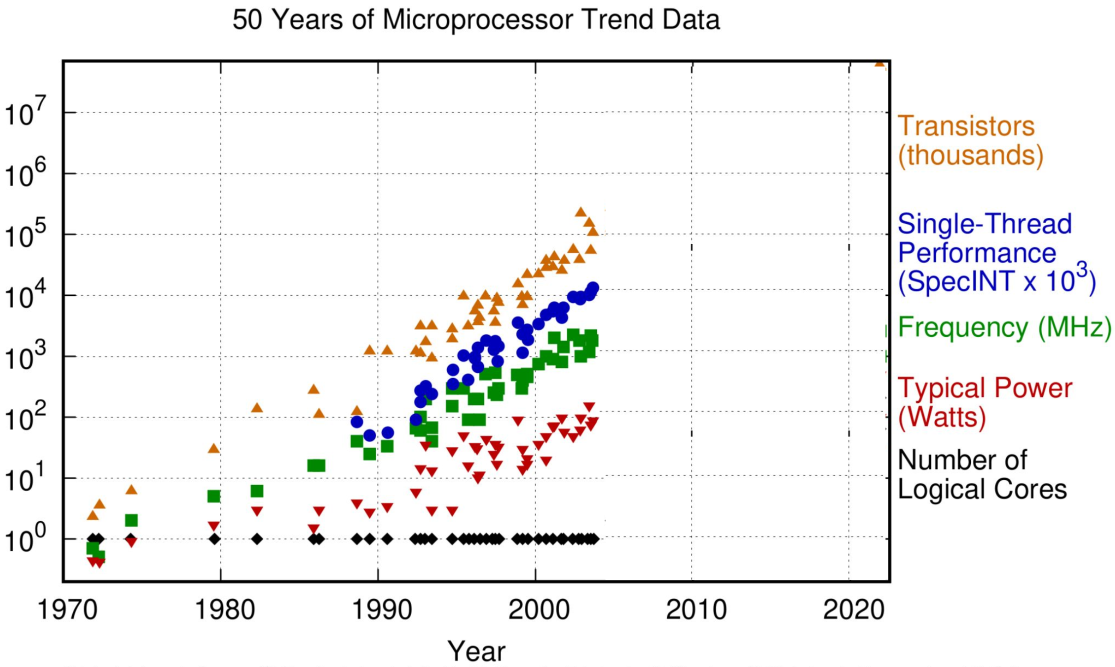
Trends in computing (Until 2000s)

- Single core:
  - Increasing core frequency (clock speed)

Increased single-core performance

Increased power consumption

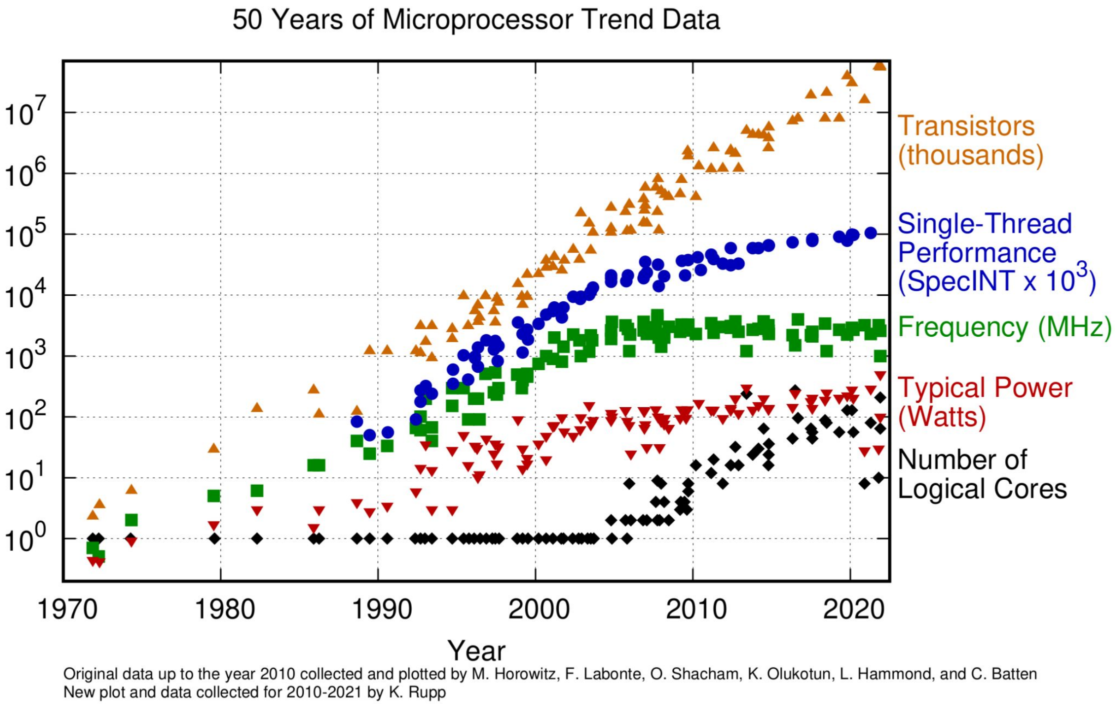
Need to dissipate heat



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

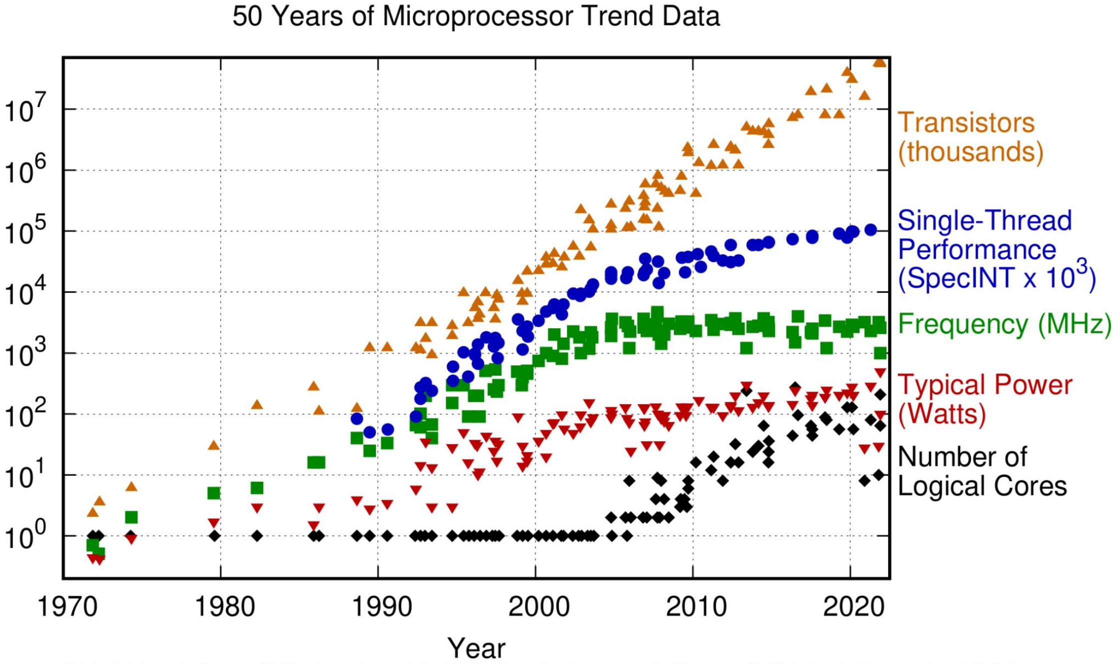
Trends in computing (After 2000s)

- Add more cores:
  - Parallel computing!
- Power and frequency both stall
- Number of logical cores increase significantly.
- Requires a re-think of programming paradigms.



Trends in computing (After 2000s)

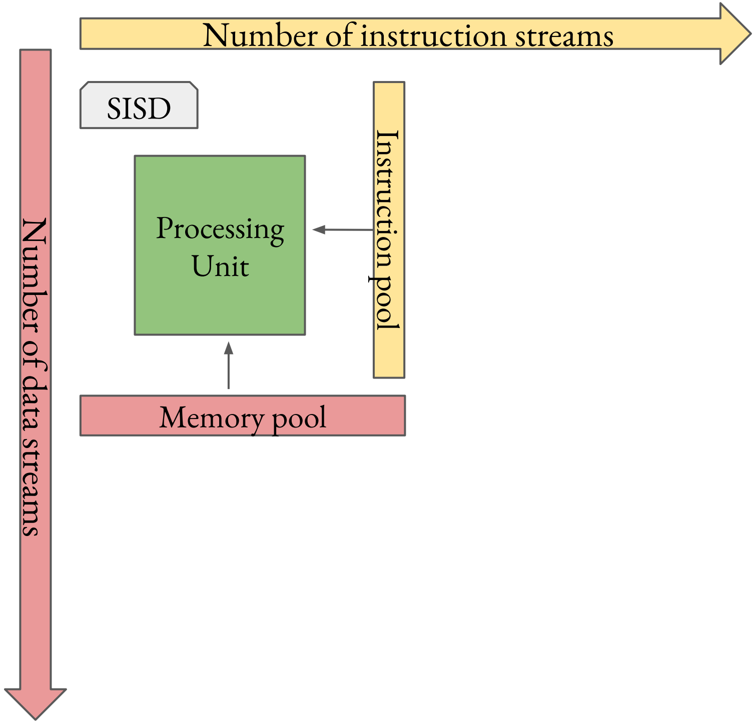
- Add more cores:
  - Parallel computing!
- Power and frequency both stall
- Number of logical cores increase significantly.
- Requires a re-think of programming paradigms.



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

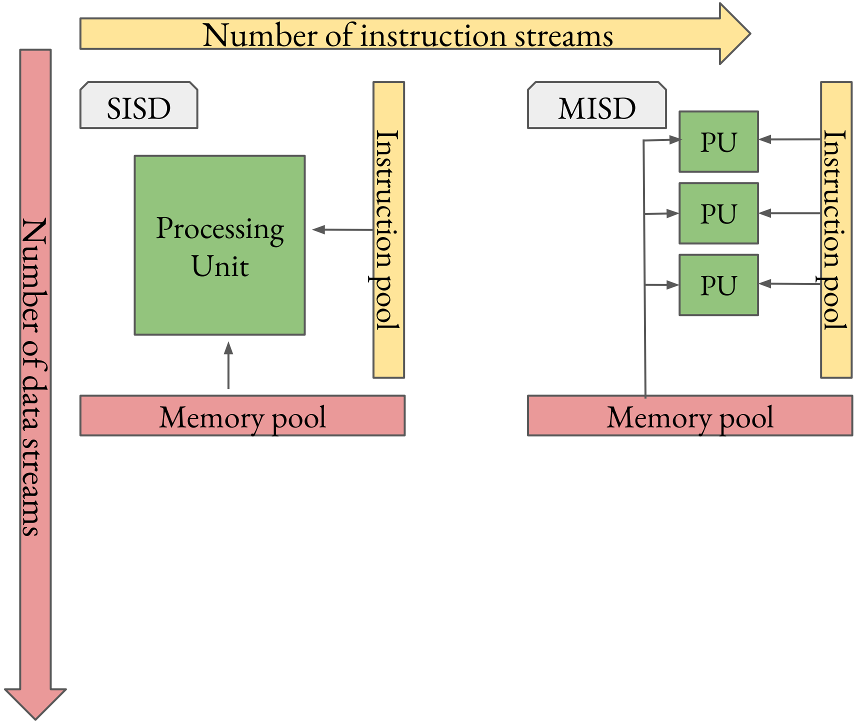
Flynn’s taxonomy

- SISD: **S**ingle **I**nstruction stream, **S**ingle **D**ata stream



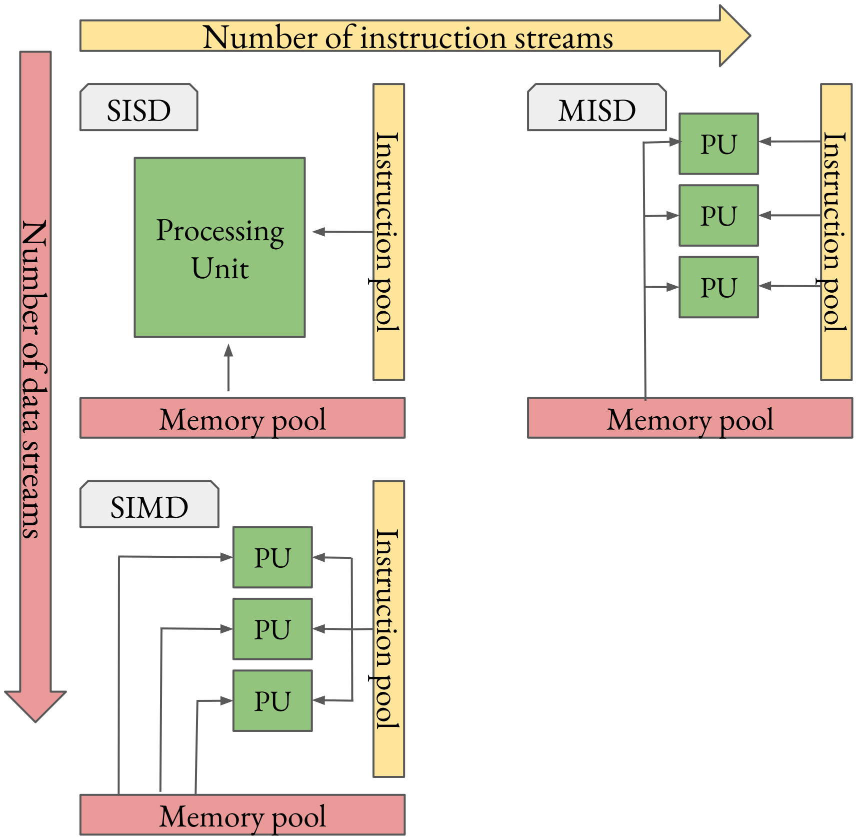
Flynn's taxonomy

- SISD: **S**ingle **I**nstruction stream, **S**ingle **D**ata stream
- MISD: **M**ultiple **I**nstruction streams, **S**ingle **D**ata stream
  - Fault tolerance.
  - Compute on same data multiple times.



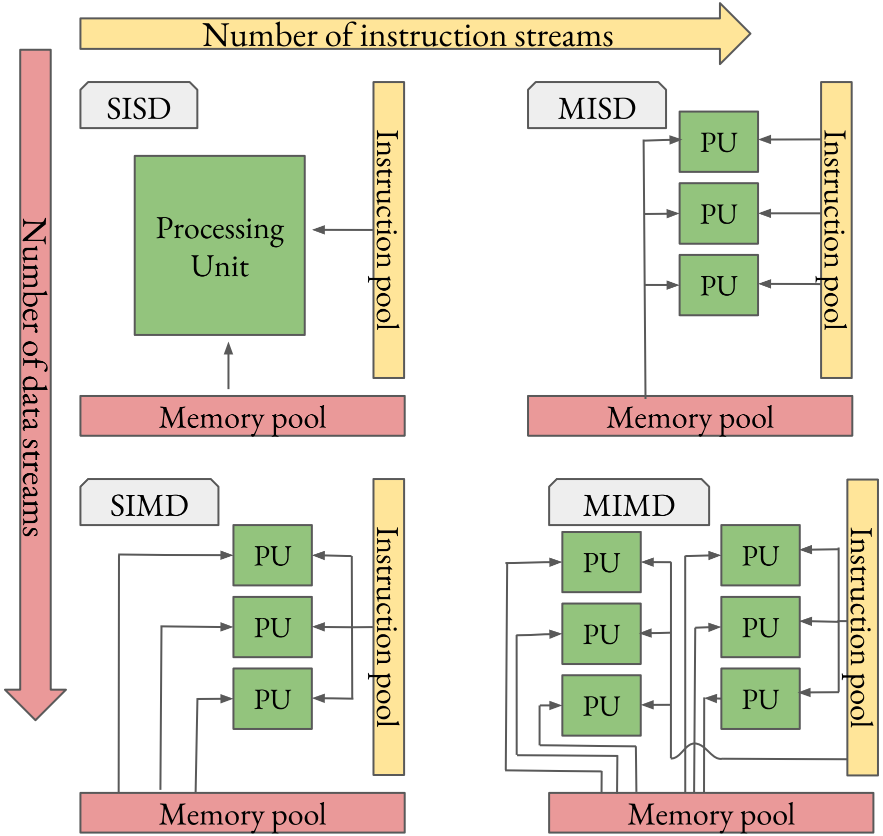
Flynn’s taxonomy

- SISD: **S**ingle **I**nstruction stream, **S**ingle **D**ata stream
- MISD: **M**ultiple **I**nstruction streams, **S**ingle **D**ata stream
- SIMD: **S**ingle **I**nstruction stream, **M**ultiple **D**ata streams.
    - Parallel processing, use single instruction, but process multiple data at once (in parallel)

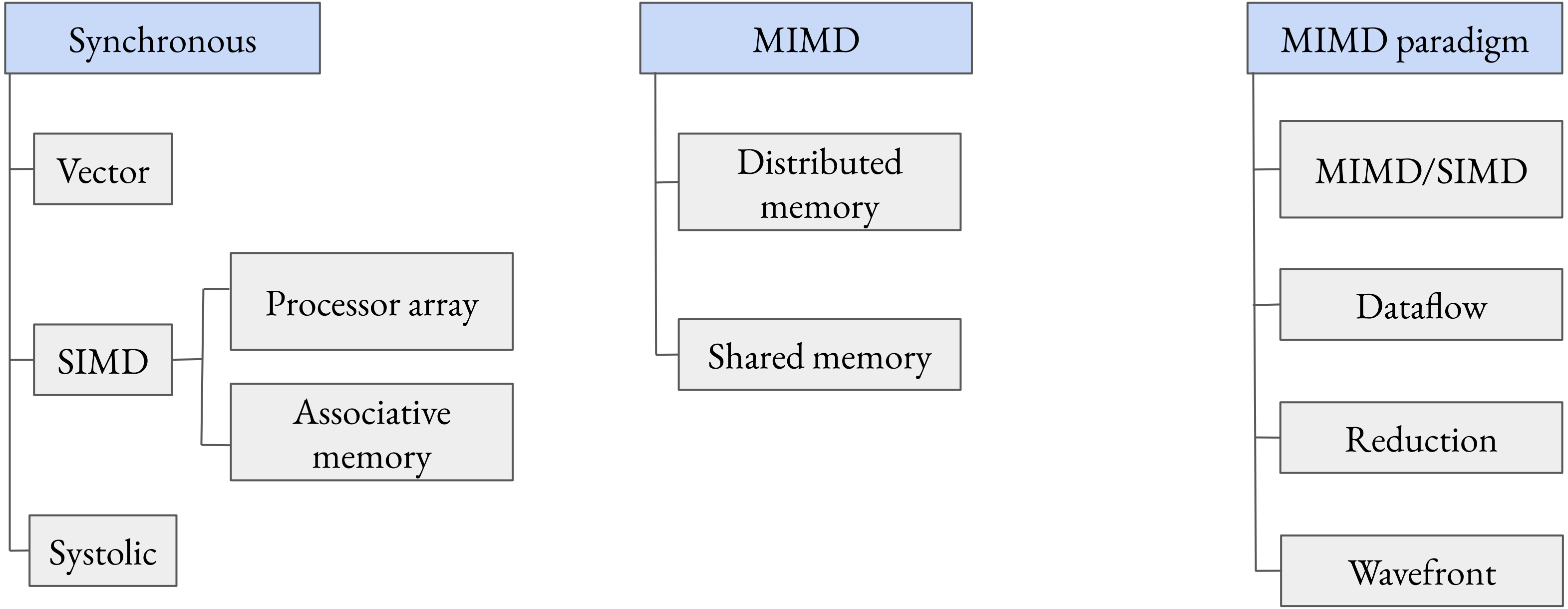


Flynn’s taxonomy

- SISD: **S**ingle **I**nstruction stream, **S**ingle **D**ata stream
- MISD: **M**ultiple **I**nstruction streams, **S**ingle **D**ata stream
- SIMD: **S**ingle **I**nstruction stream, **M**ultiple **D**ata streams.
- MIMD: **M**ultiple **I**nstruction streams, **M**ultiple **D**ata streams



A more representative taxonomy (Duncan's taxonomy)



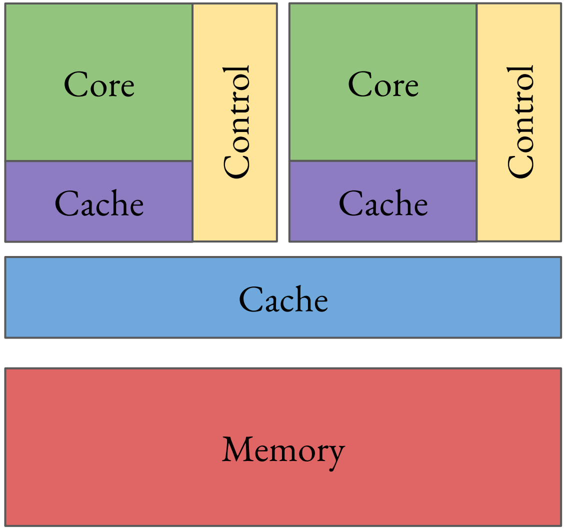
[Duncan, R, A Survey of Parallel Computer Architectures, Feb, 1990, Computer, Vol.23 (2)]

What are GPUs ?

[(2009) Mythbusters demo GPU versus CPU: https://www.youtube.com/watch?v=-P28LKWTzrI]

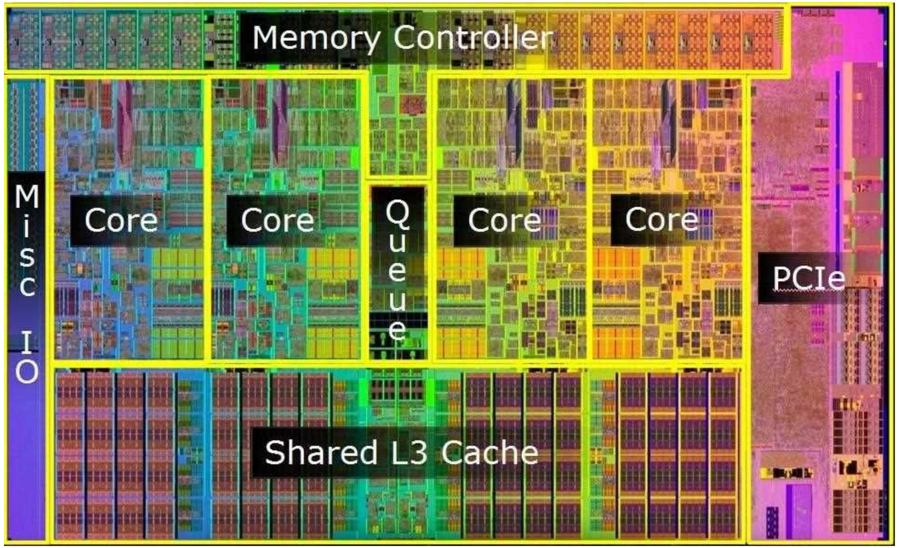
Multi-core CPU schematic

- Fetching data from main memory is very expensive
- Caches: Intermediate memory level for cores to reduce fetches needed from main memory.
- Caches are used for both instructions and data.
- Hierarchical in nature: Multiple levels, of decreasing size towards the core



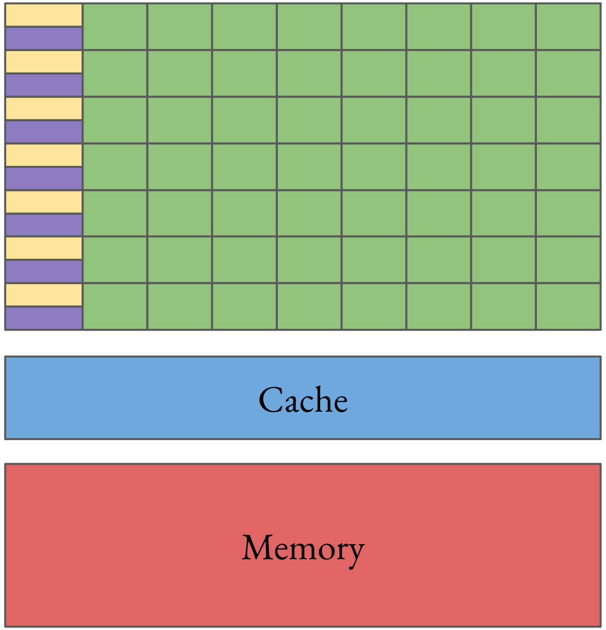
Multi-core CPU schematic

- Fetching data from main memory is very expensive
- Caches: Intermediate memory level for cores to reduce fetches needed from main memory.
- Caches are used for both instructions and data.
- Hierarchical in nature: Multiple levels, of decreasing size towards the core



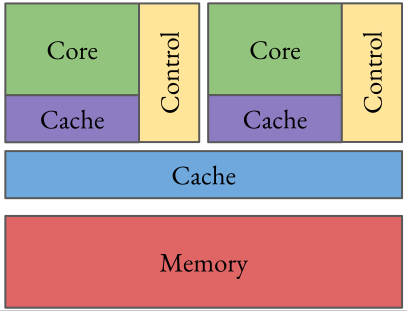
GPU schematic

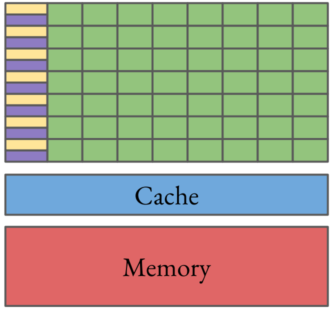
- Devote more resources (transistors) to data processing than caching and control flow.
- Slower single thread performance, but higher overall throughput.
- Smaller, more specialized instruction set.
- Hide memory latencies with computation.



Differences: CPU v/s GPU

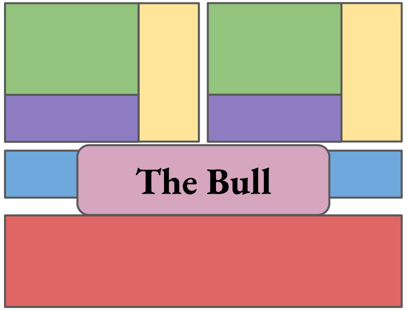
	Typical CPU (compared to GPU)	Typical GPU (compared to CPU)
ISA	Larger, more general instruction set	Smaller, more specialized instruction set
Cores	Few powerful cores	More, less powerful cores
Latency	Low latency	Higher latency
Throughput	Lower throughput	Higher throughput
Parallelism	Lower parallelism	Massive parallelism
Complexity	Suitable for complex tasks	Not suitable for complex tasks



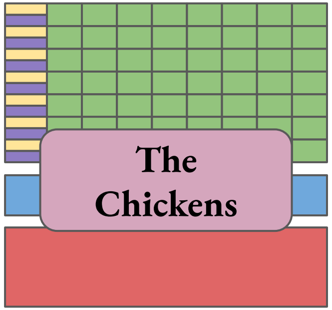


1000 chickens or 1 bull ?

Would you prefer 1000 chickens or 1(few) bull(s) to work on your field ?
- An argument against parallel computing in 1960/1970s



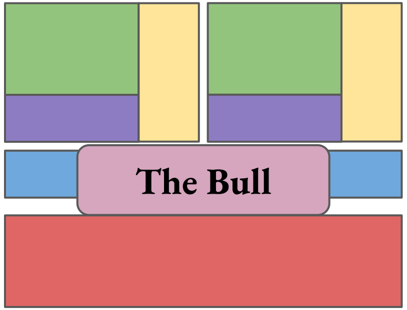
CPU

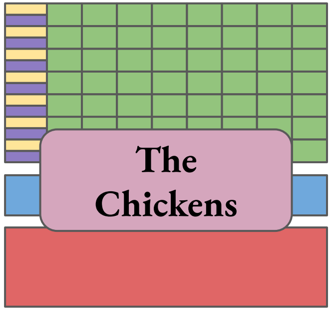


GPU

1000 chickens or 1 bull ?

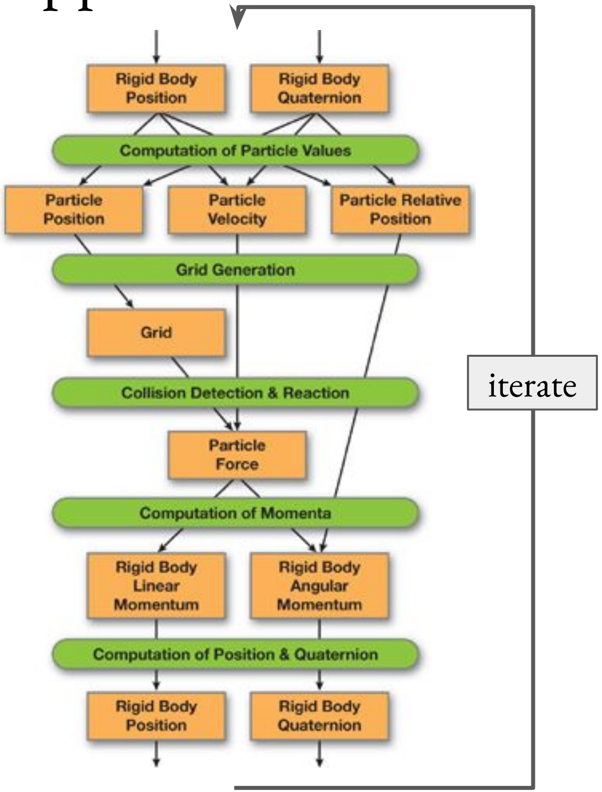
Would you prefer 1000 chickens or 1(few) bull(s) to work on your field ?
- An argument against parallel computing in 1960/1970s

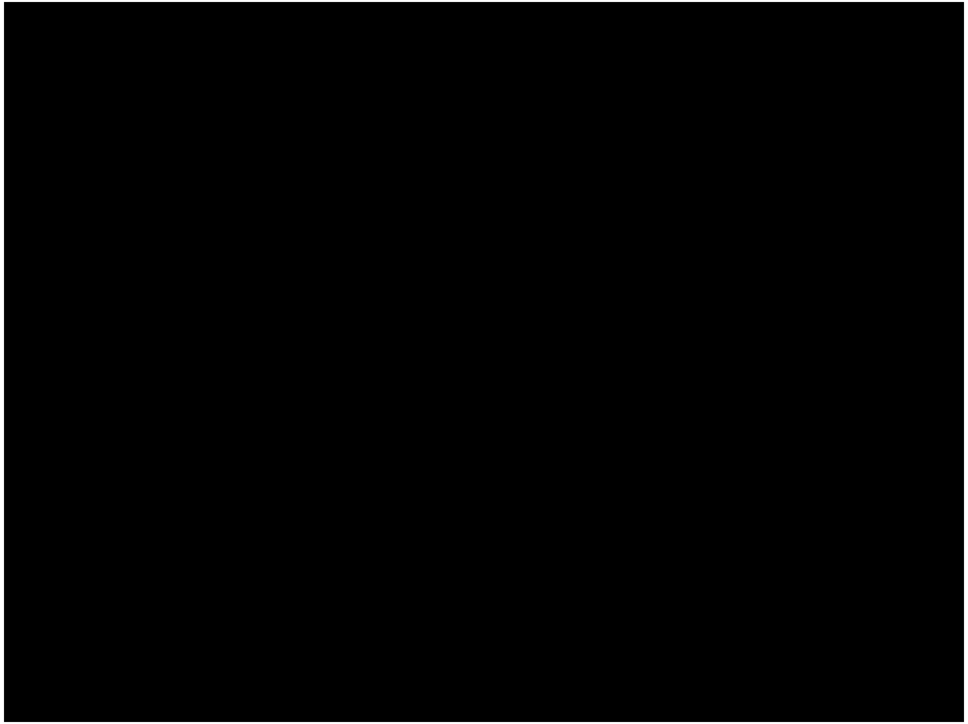




In the modern supercomputing era, the chickens have won, comprehensively.
- Jack Dongarra (Turing Award, 2021)

GPU applications: Simulating physics





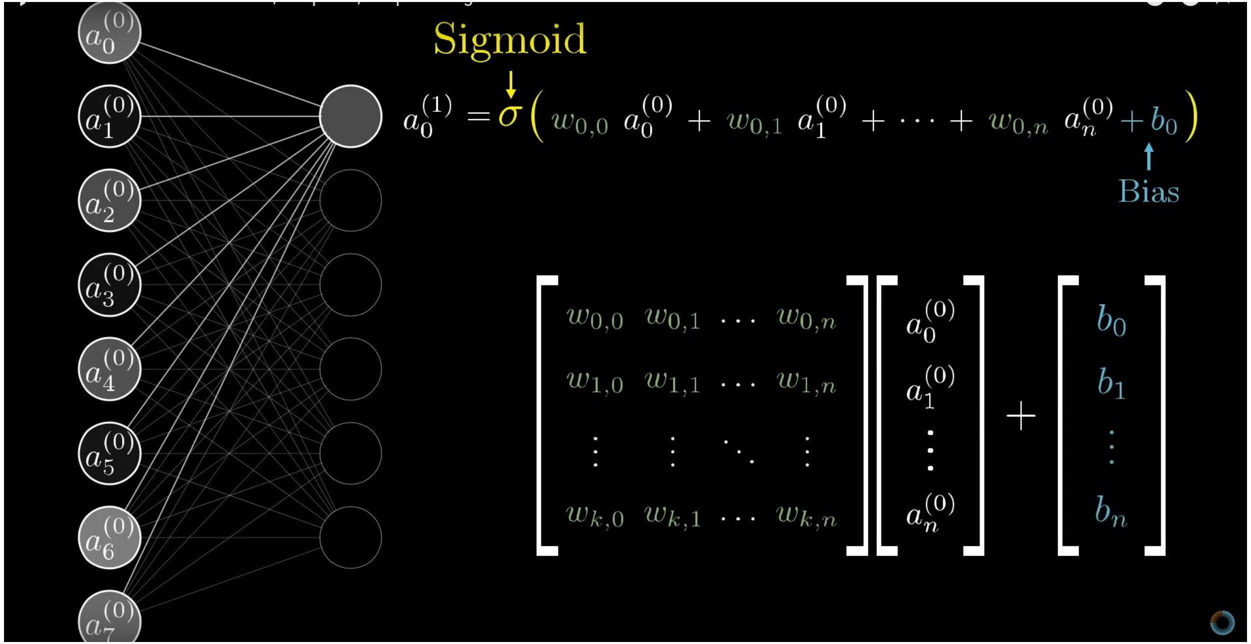
[Chapter 29, GPU Gems 3, NVIDIA]

Type text here

GPU applications: Animations

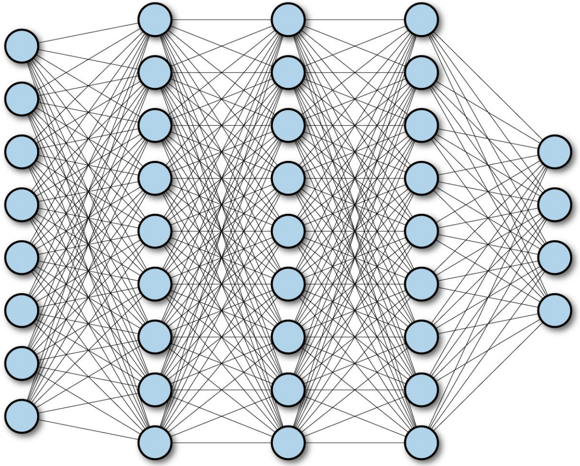
[OpenVDB software catalog]

GPU applications: Deep learning



[3Blue1Brown, YouTube]

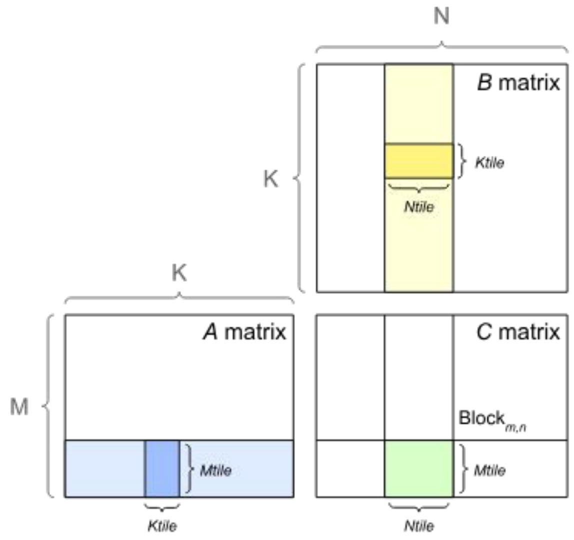
Fully connected neural network

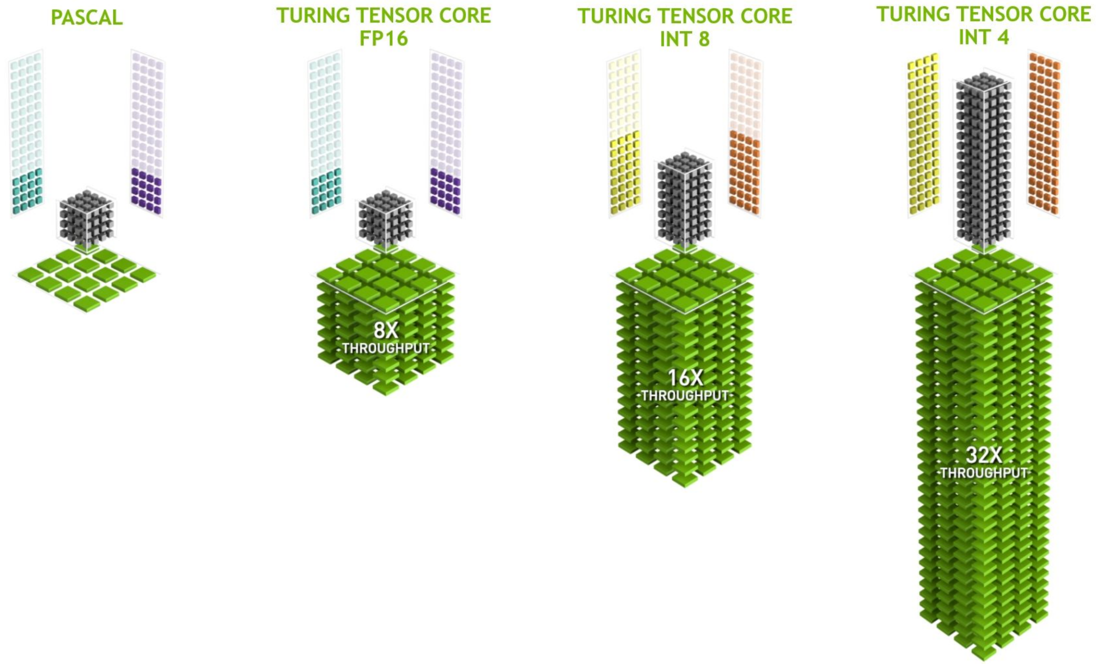


[Oreilly books]

GPU forte: Matrix-matrix multiplications

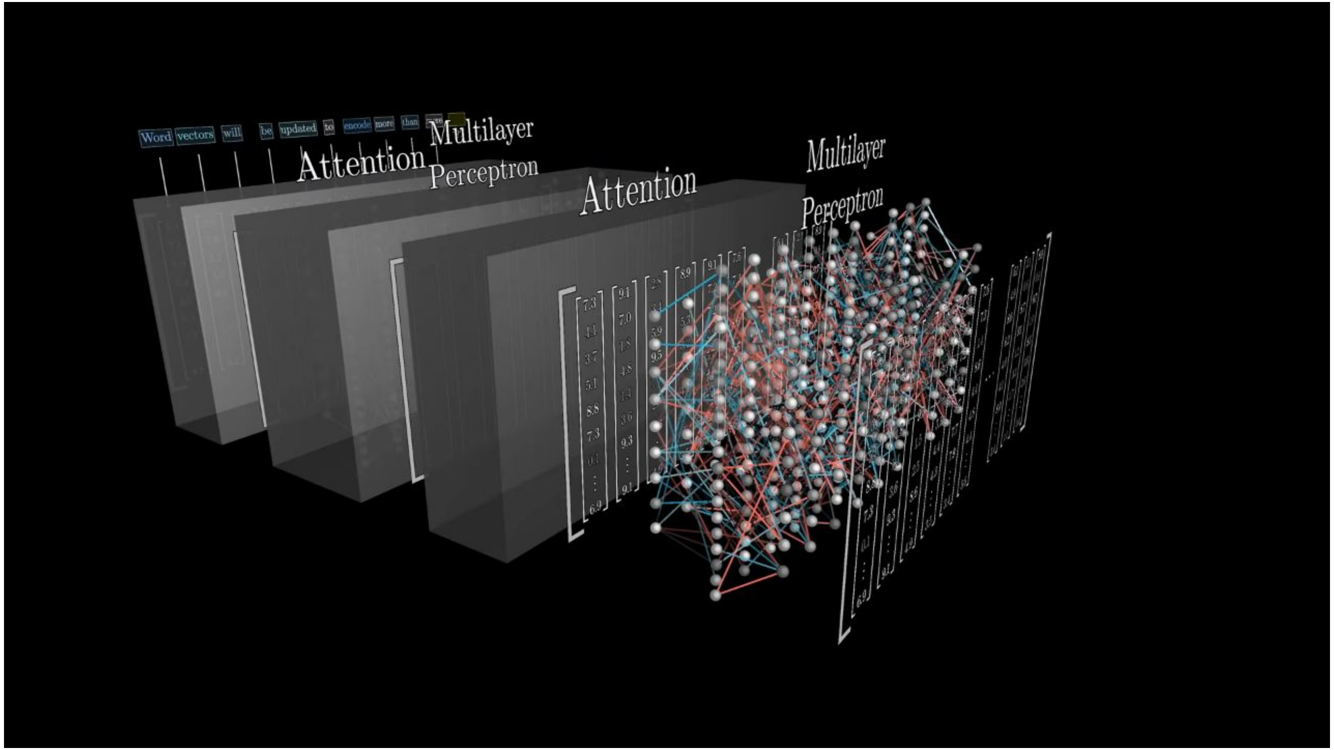
$$C = \alpha AB + \beta C$$





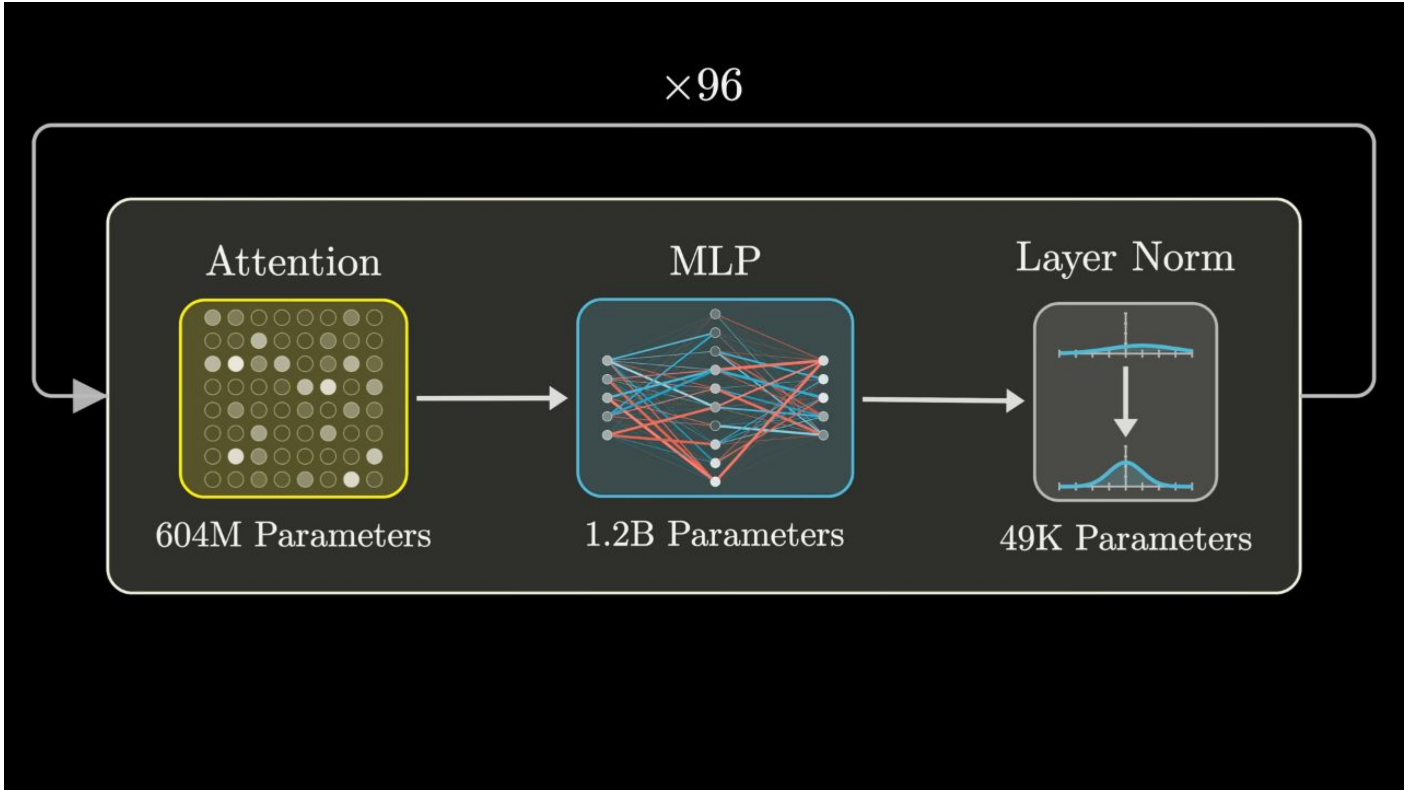
[Tensor cores across generations, NVIDIA]

GPU applications: Large language models



[3Blue1Brown, YouTube]

GPU applications: Large language models



[3Blue1Brown, YouTube]

GPU applications: Large language models

GPT-3

Total weights: 175,181,291,520

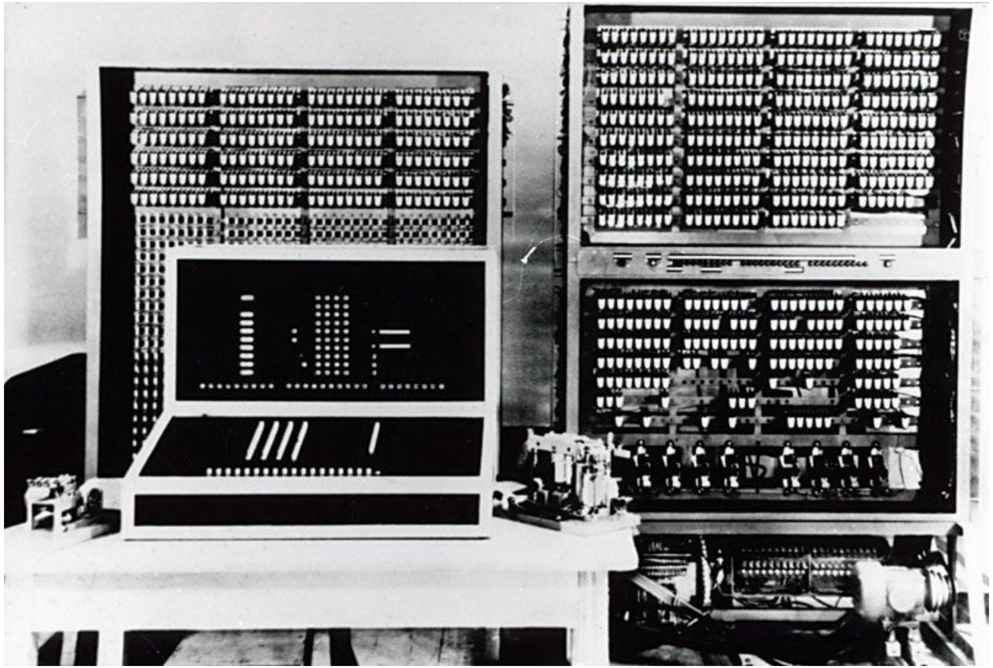
	Formula	Values	Weights
Embedding	d_embed * n_vocab	12,288 * 50,257	= 617,558,016
Key	d_query * d_embed * n_heads * n_layers	128 * 12,288 * 96 * 96	= 14,495,514,624
Query	d_query * d_embed * n_heads * n_layers	128 * 12,288 * 96 * 96	= 14,495,514,624
Value	d_value * d_embed * n_heads * n_layers	128 * 12,288 * 96 * 96	= 14,495,514,624
Output	d_embed * d_value * n_heads * n_layers	12,288 * 128 * 96 * 96	= 14,495,514,624
Up-projection	n_neurons * d_embed * n_layers	49,152 * 12,288 * 96	= 57,982,058,496
Down-projection	d_embed * n_neurons * n_layers	12,288 * 49,152 * 96	= 57,982,058,496
Unembedding	n_vocab * d_embed	50,257 * 12,288	= 617,558,016

[3Blue1Brown, YouTube]

Backup

History of Computing

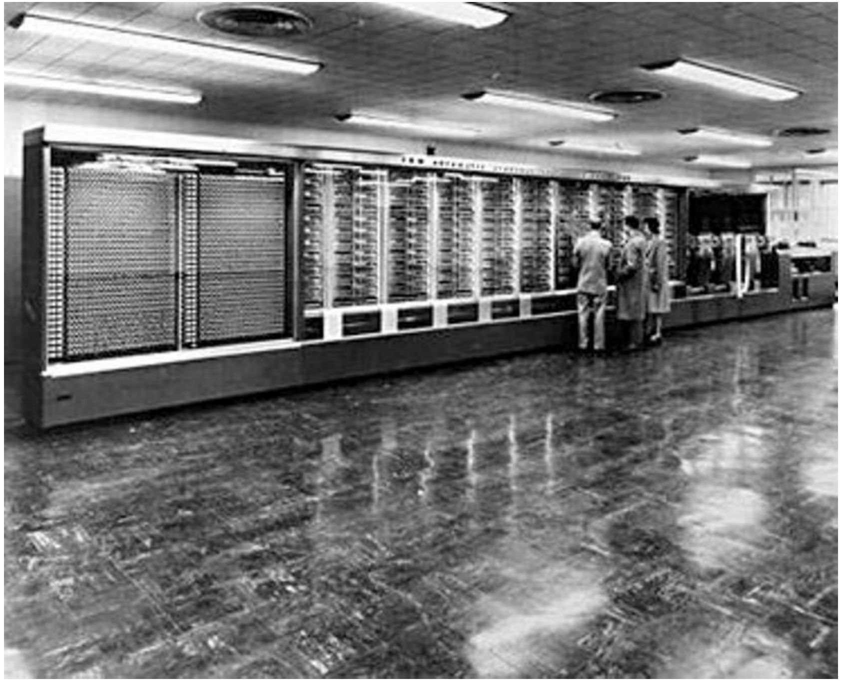
1. 1941 Konrad Zuse (Z3)
   a. 22-bit word length
   b. Destroyed in WW2
   c. Rebuilt and on display in Deutsches Museum in Munich.



[Source: computerhistory.org]

History of Computing

1. 1941: Konrad Zuse (Z3)
2. 1944: Harvard Mark 1



[Source: computerhistory.org]

History of Computing

1. 1941: Konrad Zuse (Z3)
2. 1944: Harvard Mark 1
3. 1945: ENIAC
    a. 1000x faster
    b. Turing-complete
    c. Re-programmable
    d. A whole of 500 Flops
    e. Longest operation without failure: 5 days

[Source: computerhistory.org]

History of Computing

1. 1941: Konrad Zuse (Z3)
2. 1944: Harvard Mark 1
3. 1945: ENIAC
4. 1951: UNIVAC
    a. Commercially available
    b. Later versions programmable in COBOL

[By U.S. Census Bureau employees - https://www.census.gov/history/, Public Domain, https://commons.wikimedia.org/w/index.php?curid=61118833]

History of Computing

1. 1941: Konrad Zuse (Z3)
2. 1944: Harvard Mark 1
3. 1945: ENIAC
4. 1951: UNIVAC
5. 1956: TX-0
    a. Fully Transistorized

[Source: computerhistory.org]

History of Computing

1. 1941: Konrad Zuse (Z3)
2. 1944: Harvard Mark 1
3. 1945: ENIAC
4. 1951: UNIVAC
5. 1956: TX-0
6. 1966: IBM System/360
    a. Popular series of systems
    b. ~7000kg, ~3500 instr. per sec

[By ArnoldReinhold - Own work, CC BY-SA 3.0,
https://commons.wikimedia.org/w/index.php?curid=47096462]

History of Computing

7. 1976: Cray-1 supercomputer
    a. US$7.9 million
    b. 160 MFlops
    c. Serial computation

[By Irid Escent - 20180227_132902, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=85791445]

History of Computing

7. 1976: Cray-1 supercomputer

8. 1977: Apple-II

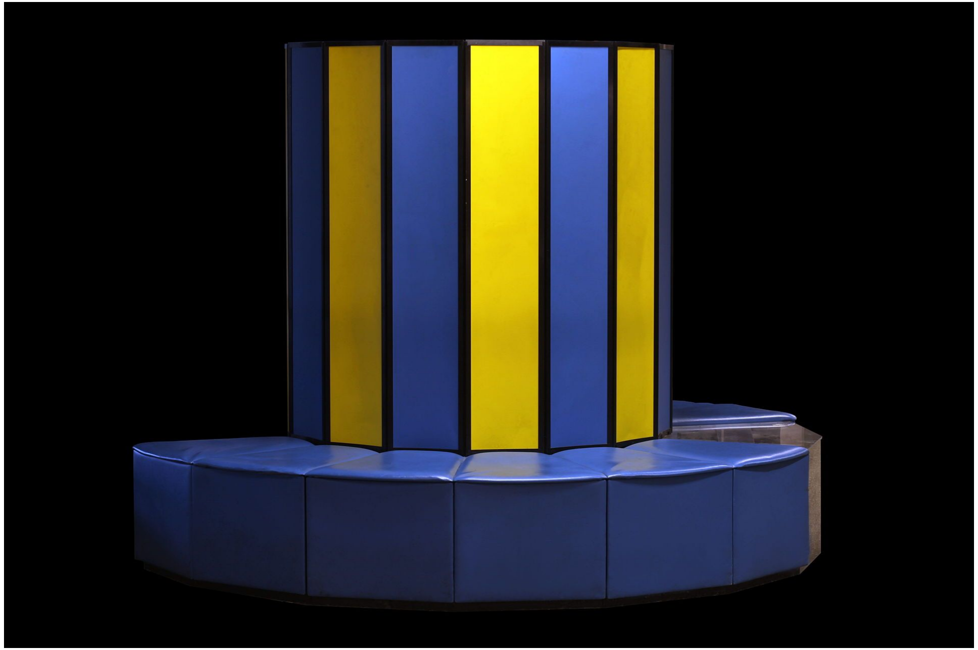
   a. Popularized personal computers.

   b. Millions sold

[Source: computerhistory.org]

History of Computing

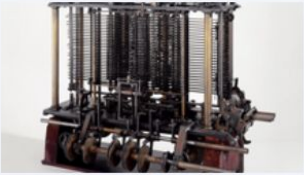
7. 1976: Cray-1 supercomputer
8. 1977: Apple-II
9. 1982: Cray X-MP supercomputer
    a. Parallel vector processor (4 CPUs)
    b. 800 MFlops
    c. US$15 million



[By Photograph by Rama, Wikimedia Commons, CC BY-SA 2.0 fr, https://commons.wikimedia.org/w/index.php?curid=14641017]

History of Computing

Mid-1800s-1930s
Early mechanical computers



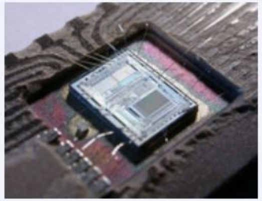
1930s
Electro-Mechanical computers

1940s
Electronic computers

1950s
The first commercial computers

Mid 1950s
Transistors computers

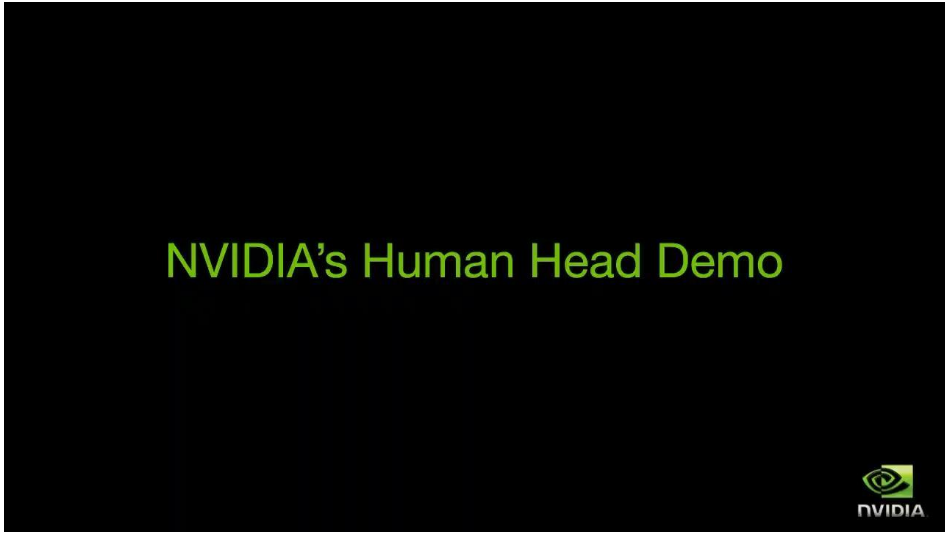
1960s
The Microchip and the Microprocessor



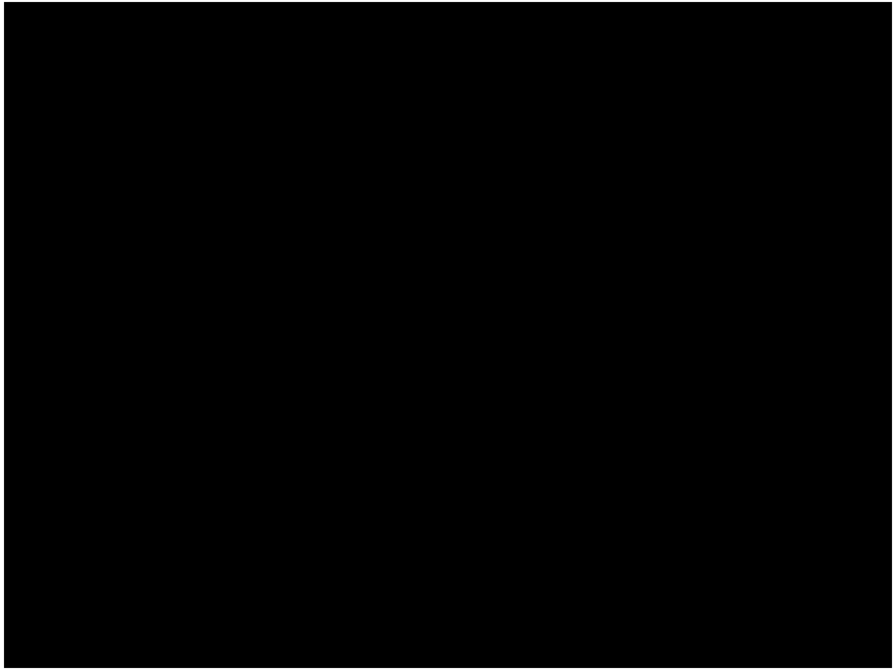
1970s
Personal computers

1980s-1990s
The Early Notebooks and Laptops

GPU applications: Realistic Rendering



[Chapter 14, GPU Gems 3, NVIDIA]



[Chapter 4, GPU Gems 3, NVIDIA]