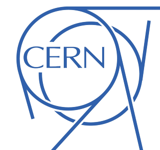


Introduction to Architecture and Performance

Felice Pantaleo, Tim Mattson

CERN Experimental Physics Department

felice@cern.ch

Who is Felice?

- CERN Staff Physicist @ CMS Experiment by day
  - Diaper-Changing Superhero by Night
- Parallel Algorithms, GPU Programming, Performance Portability
- Optimizing Algorithms vs. Optimizing Nap Time
- Email: felice@cern.ch

Why computing?

CERN

"The purpose of computing is [to gain] insight" (Richard Hamming)

We gain and generate insight by solving problems

How do we ensure problems are solved by electrons?

Source: http://www.sia-online.org (semiconductor industry association)

CERN

Source: http://www.sia-online.org (semiconductor industry association)

Source: https://iq.intel.com/5-awesome-uses-for-drone-technology/

CERN

Source:
http://sm.pcmag.com/pcmag_uk/photo/g/google-self-driving-car-the-guts/google-self-driving-car-the-guts_dwx8.jpg

CERN

Overview

Algorithm:

Step-by-step procedure that is guaranteed to terminate where each step is precisely stated and can be carried out by a computer

- Finiteness
- Definiteness
- Effective computability

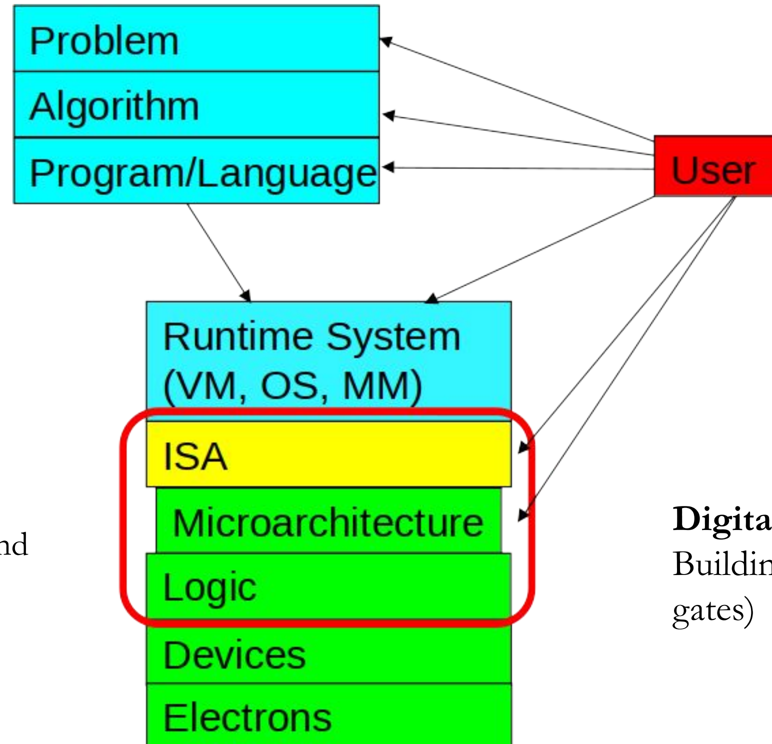
Many algorithms for the same problem

ISA (Instruction Set Architecture)

- Interface/contract between SW and HW.
- What the programmer assumes hardware will satisfy.

Microarchitecture:

- An implementation of the ISA



Digital logic circuits
Building blocks of micro-arch (e.g., gates)

The Power of Abstraction

CERN

Levels of transformation create abstractions

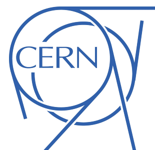
- Abstraction: A higher level only needs to know about the interface to the lower level, not how the lower level is implemented
- E.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions

Abstraction improves productivity

- No need to worry about decisions made in underlying levels
- E.g., programming in Python vs. C++ vs. assembly vs. binary vs. by specifying control signals of each transistor every cycle

Then, why would you want to know what goes on underneath or above?

Crossing the Abstraction Layers



As long as everything goes well, not knowing what happens underneath (or above) is not a problem.

What if:

- The program you wrote is running slow?
- The program you wrote does not run correctly?
- The program you wrote consumes too much energy?
- Your system just shut down and you have no idea why?
- Someone just compromised your system and you have no idea how?

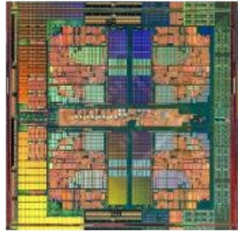
What if:

- The hardware you designed is too hard to program?
- The hardware you designed is too slow because it does not provide the right primitives to the software?

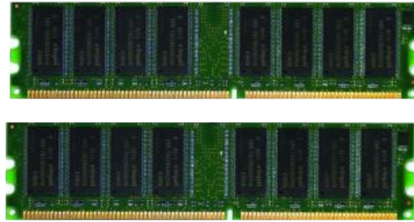
Understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer

Past systems

- Computing landscape is very different from 10-20 years ago
- Every component and its interfaces are being re-examined



Microprocessor



Main Memory

Storage (SSD/HDD)

Modern systems

- Applications and technology demand novel architectures
  - Driven by huge hunger for data (Big Data), new applications (ML/AI, graph analytics, genomics), ever-greater realism
  - We can easily collect more data than we can analyze/understand
  - Five walls: Energy, reliability, complexity, security, scalability

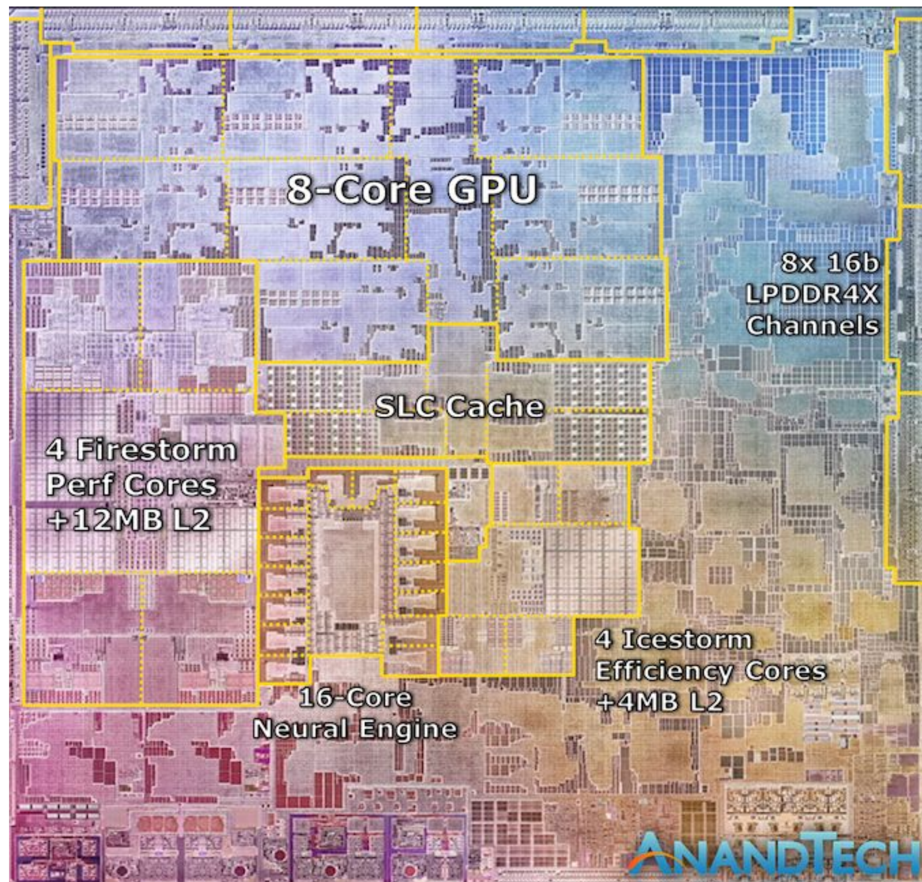
FPGA

Hybrid Memory

Persistent memory/Storage

Heterogeneous Processors and Accelerators

GPU



Source: https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested

What you will master soon

DRAM

DRAM

Control

ALU ALU ALU ALU

Cache Cache Cache Cache

Socket 0

Cache

Control

ALU ALU ALU ALU

Cache Cache Cache Cache

Socket 1

Cache

DRAM

DRAM

GPU 0

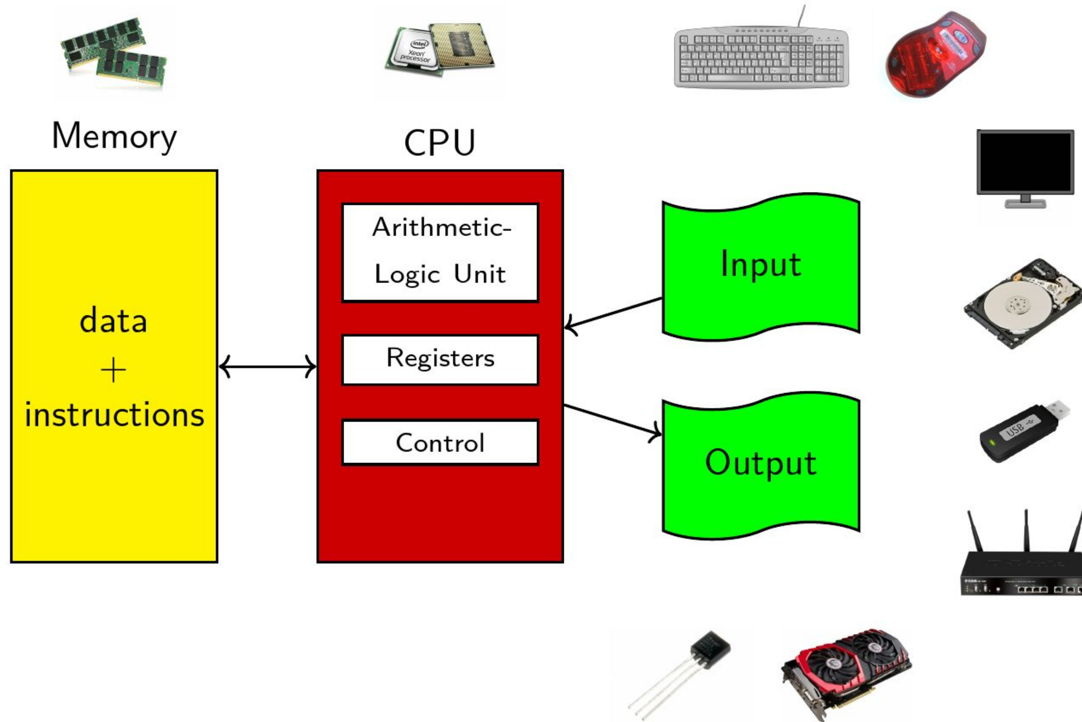
GPU 1

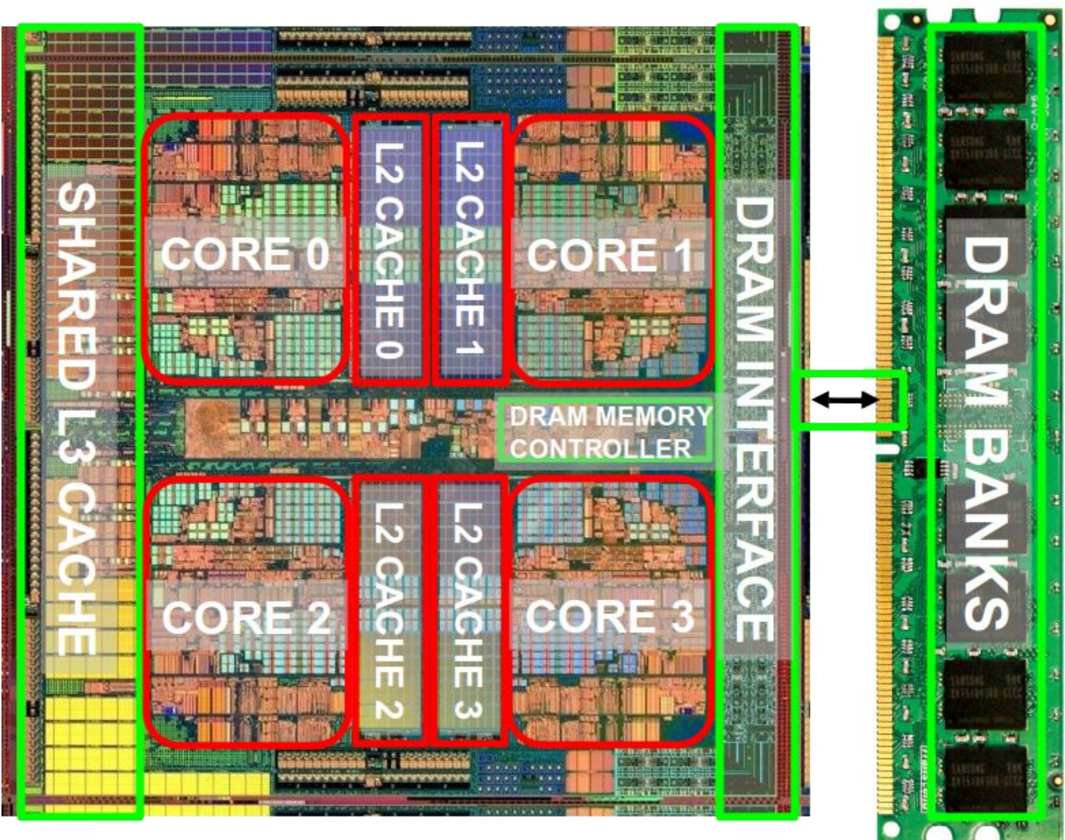
Interconnect

Network Interface

Other Machine

Von Neumann Architecture

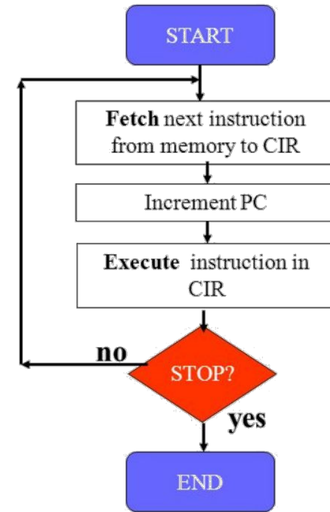




CERN

Von Neumann Architecture

- The basic operation that every Processing Unit (PU) has to process is called instruction and the address in memory containing the instruction is saved
- A *Program Counter* (PC) holds the address of the next instruction
- *fetch*: the content of the memory stored at the address pointed by the PC is loaded in the Current Instruction Register (CIR) and the PC is increased to point to the next instruction's address
- *decode*: the content of the CIR is interpreted to determine the actions that need to be performed
- *execute*: an Arithmetic Logic Unit performs the decoded actions.



A glance into CPU performance

CPU time

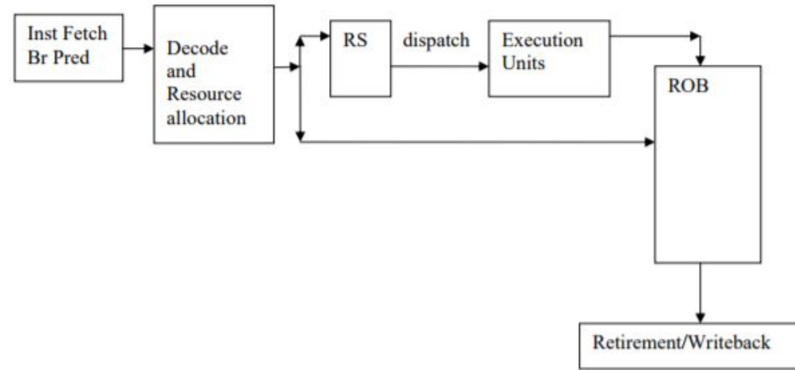
You want to minimize the CPU time and understand what handles you have

$$Time_{CPU} = \text{Instruction Count} \times \text{Cycles per Instruction} \times \text{Clock Cycle Time}$$

$$Time_{CPU} = \sum_i (\text{IC}_i \times \text{CPI}_i) \times \text{Clock Cycle Time}$$

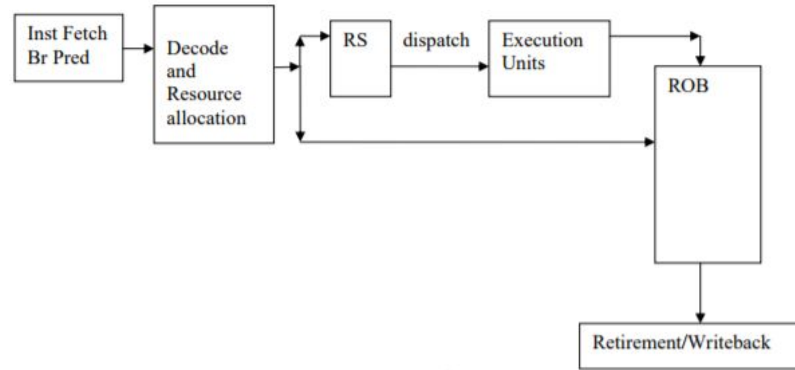
Speculative execution

- Modern processors execute many more instructions than the program flow needs (Core Out Of Order pipeline).
- The Front-end fetches the program code decodes instructions into one or more low-level hardware operations called micro-ops (uOps).
- The uOps are then fed to the Back-end in a process called allocation.
- Leaving the Retirement Unit means that:
  - the instructions are finally executed
  - their results are correct and visible in the architectural state as if they execute in-order



Retired instructions

- Instructions that were "proven" as indeed needed by the program execution flow are **retired**
- Instructions and uOps of incorrectly predicted paths are flushed
- Then the uOps associated with the instruction to be retired have completed (together with older instructions)
- Retirement of the correct execution path instructions can proceed



Clockticks per Instructions Retired (CPI)

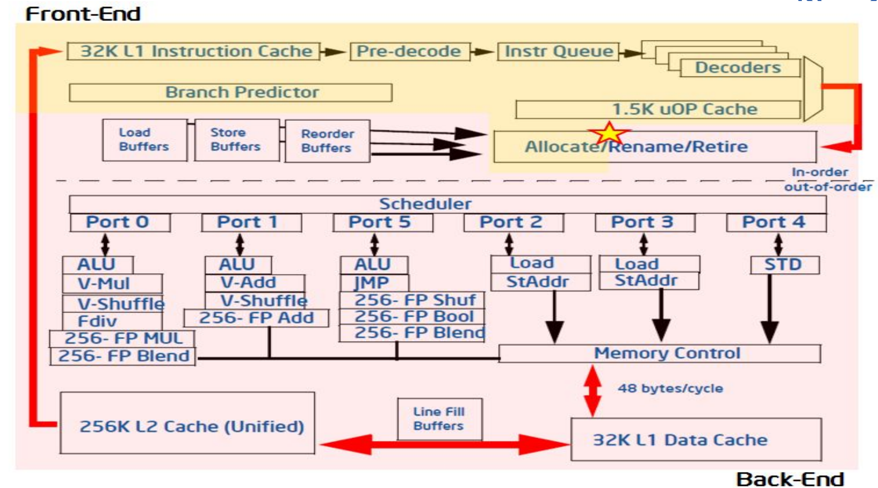
- The CPI value of an application or function is an indication of how much **latency** affected its execution
  - Higher CPI means: on average, it took more clockticks for an instruction to retire.
  - Latency in your system can be caused by cache misses, I/O, or other bottlenecks
- CPI < 1: instruction bound code
- CPI > 1: stall cycle bound or memory bound.

CPI vs Retired instructions

- Optimizations will affect either CPI or the number of instructions to execute, or both.
- Using CPI without considering the number of instructions executed can lead to an incorrect interpretation of your results.

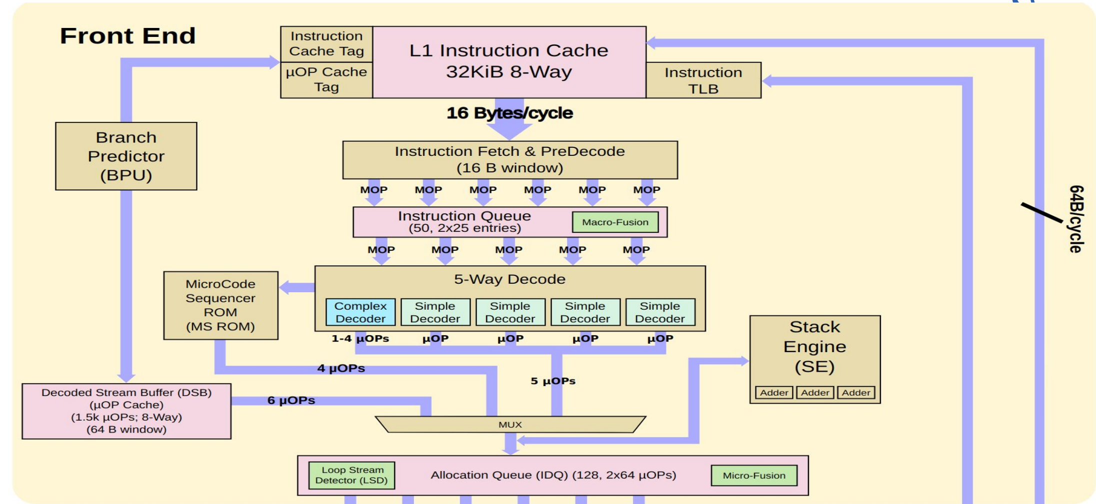
Instructions pipeline

- The Front-end of the pipeline can allocate four uOps per cycle
- The Back-end can retire four uOps per cycle
- A pipeline slot represents the hardware resources needed to process one uOp.
- For each CPU core, on each clock cycle, there are four pipeline slots available.
- During any cycle, a pipeline slot can either be empty or filled with a uOp. If a slot is empty during one clock cycle, this is attributed to a stall. The next step needed to classify this pipeline slot is to determine whether the Front-end or the Back-end portion of the pipeline caused the stall



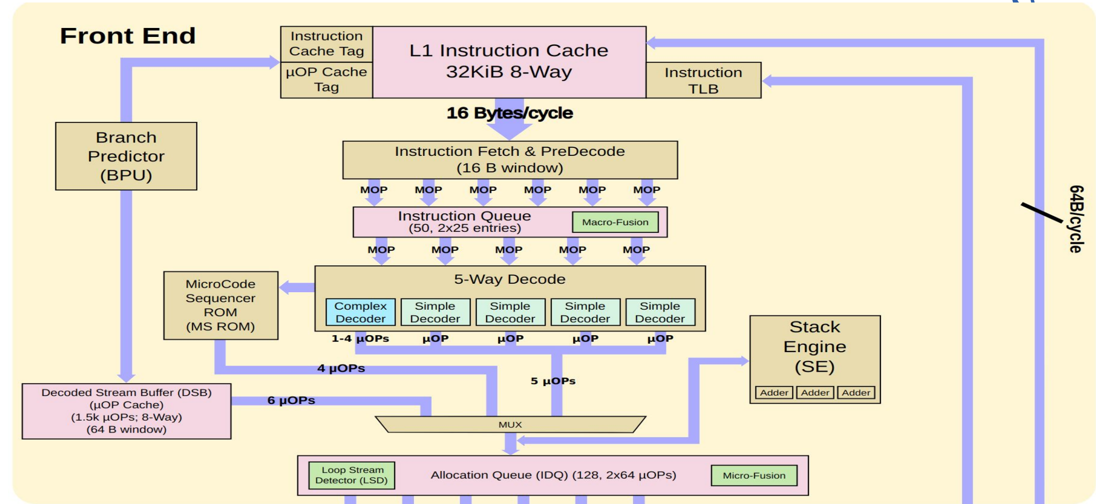
https://en.wikichip.org/w/images/e/ee/skylake_server_block_diagram.svg

Front-end



- Feeds “decoded” instructions to the scheduler
- Affected by instruction non-locality (iCache-miss, iTLB misses) and mispredicted branches

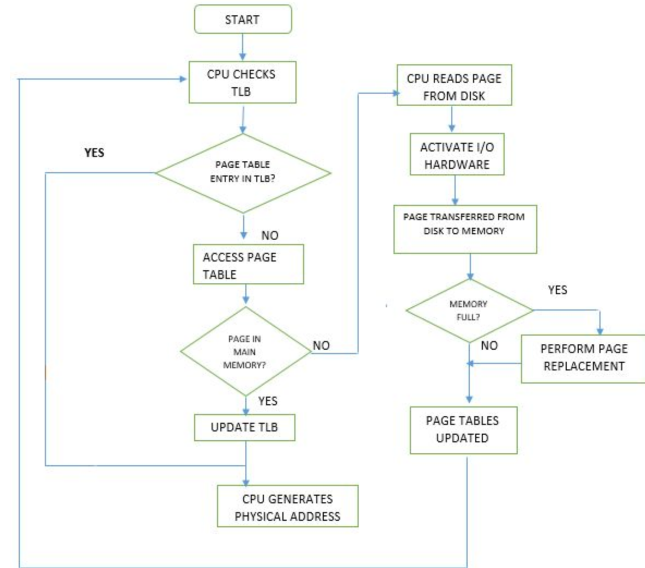
Front-end



- Feeds "decoded" instructions to the scheduler
- Affected by instruction non-locality (iCache-miss, iTLB misses) and mispredicted branches
- Main metrics:
- `L1-icache-load-misses (icache.ifdata_stall )` Cycles where a code fetch is stalled due to L1 instruction cache miss.
- `branch-misses (br_misp_retired.all_branches)` This event counts all mispredicted branch instructions retired.

TLB

- The CPU has to access main memory for an instruction-cache miss, data-cache miss, or TLB miss.
- A **translation lookaside buffer** (TLB) is a memory cache that stores the recent translations of virtual memory to physical memory.
- TLB miss usually worse than instruction-cache or data cache miss



Helping the Front-end

CERN

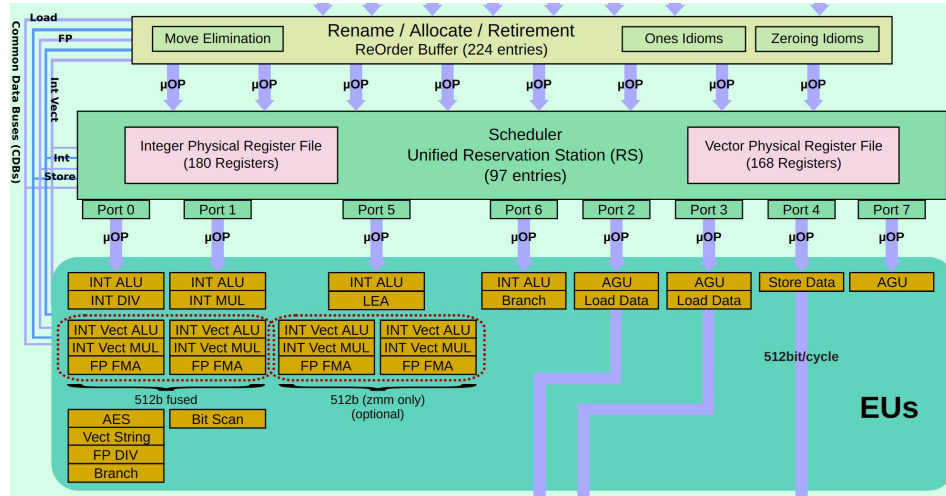
- Avoid complex branching patterns
- Keep code local (inline)
- Keep loop short (so they fit in μOp cache)

Back-end

Computational engine of the CPU:

Affected by

- instruction dependency
  - instruction parallelism
  - pipelining
- Memory access
- Latency of slow instructions
  - div sqrt
- Vectorization

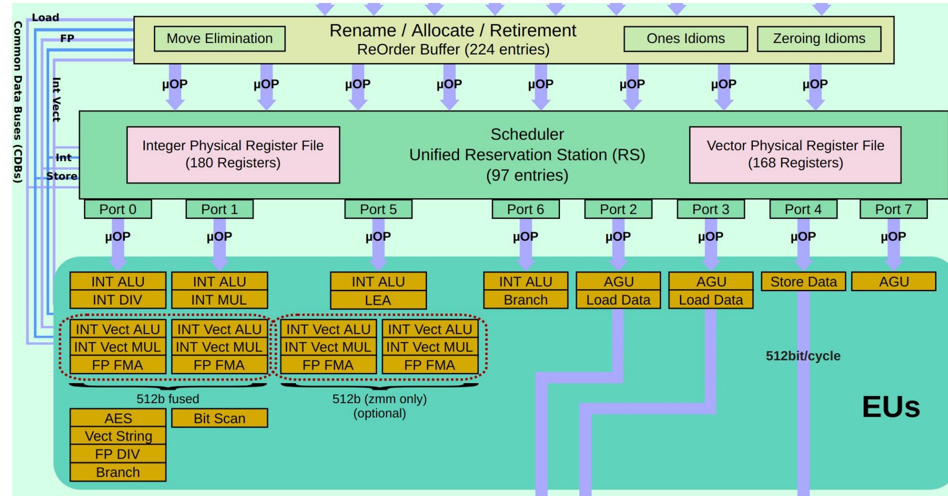


?

31

Back-end

Computational engine of the CPU:
Affected by
- instruction dependency
  - instruction parallelism
  - pipelining
- Memory access
- Latency of slow instructions
  - div sqrt
- Vectorization



?

Main Metrics:

uops_executed.stall_cycles

This event counts cycles during which no uops were dispatched from the Reservation Station (RS)

uops_executed.thread

Number of uops to be executed each cycle.

cycle_activity.stalls_mem_any

Execution stalls while memory subsystem has an outstanding load.

arith.divider_active

Cycles when divide unit is busy executing divide or square root operations. Accounts for integer and floating-point

32

Real-life latencies

- Most integer/logic instructions have a one-cycle execution latency:
  - For example
    - ADD, AND, SHL (shift left), ROR (rotate right)
  - Amongst the exceptions:
    - IMUL (integer multiply): 3
    - IDIV (integer divide): 13 – 23
- Floating-point latencies are typically multi-cycle
  - FADD (3), FMUL (5)
    - Same for both x87 and SIMD double-precision variants
  - Exception: FABS (absolute value): 1
  - Many-cycle, no pipepine : FDIV (20), FSQRT (27)
  - Other math functions: even more
- As of Haswell:
  - FMA (5 cycles)
- As of Skylake:
  - SIMD ADD, MUL, FMA: 4 cycles

- http://www.agner.org/optimize/instruction_tables.pdf

Helping the Back-end

- Keep data at hand
- Vectorize
- Recast loop to help the compiler to vectorize
- Avoid divisions and sqrt!

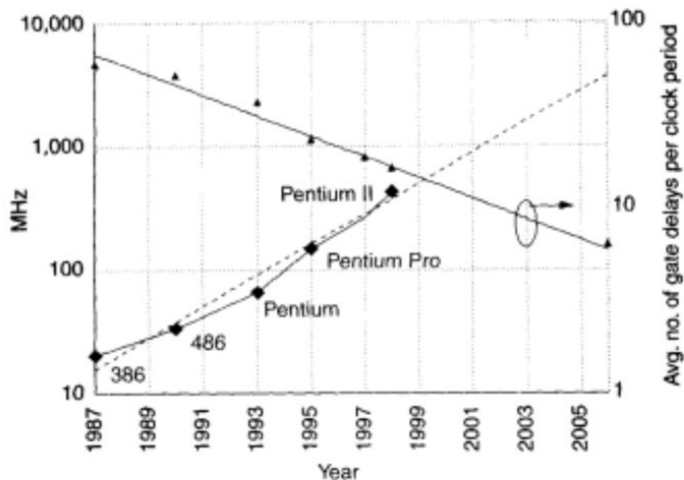
Helping the compiler to vectorize

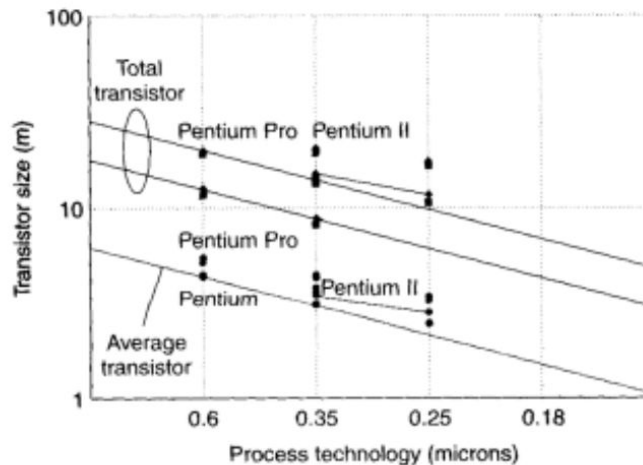
- Vectorization is enabled in gcc by the flags:
  - -ftree-vectorize
  - -O3
- Vectorizable:
  - Countable innermost loops
  - No variations in the control flow
  - Contiguous memory access
  - Independent memory access
- Avoid aliasing problems with restrict
- Use countable loops, with no side effects (break, continue, non-inlined function calls)
- Avoid indirect memory access (x[y[i]])

Not vectorizable

```
while(x[i] != 42)
{
   if(x[i] == 0)
      x[i] = x[i-1]
}
```

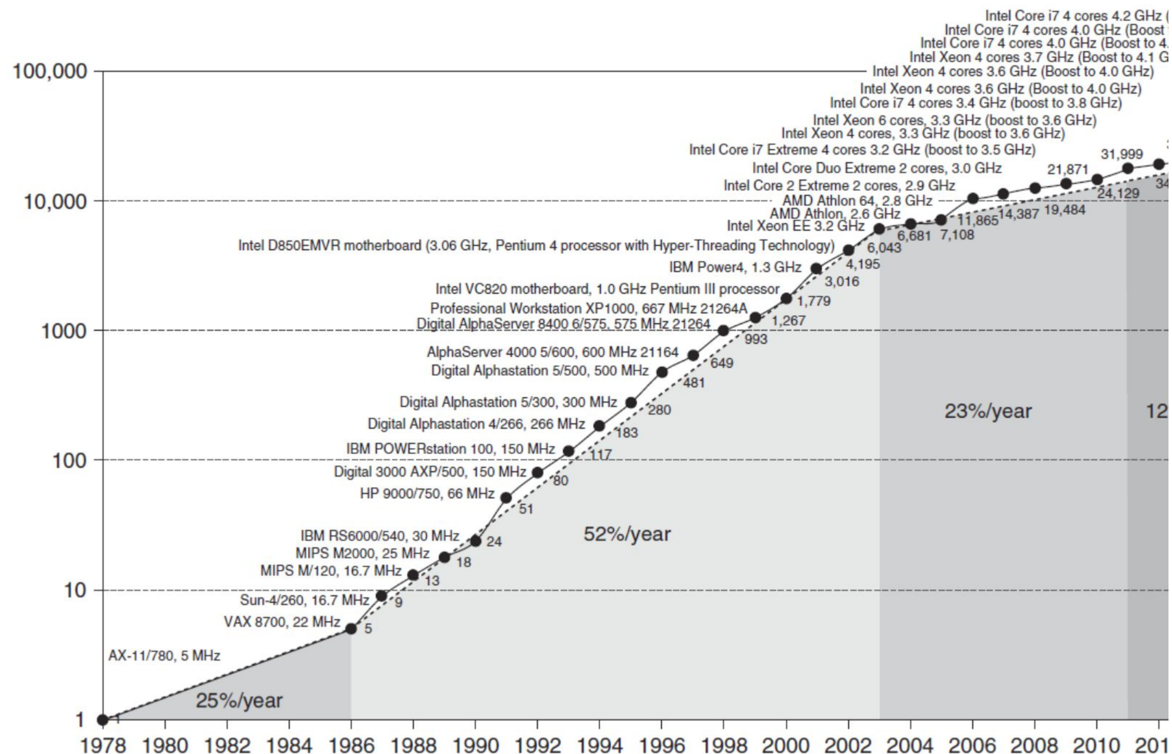
Previously, in Moore's Paradise



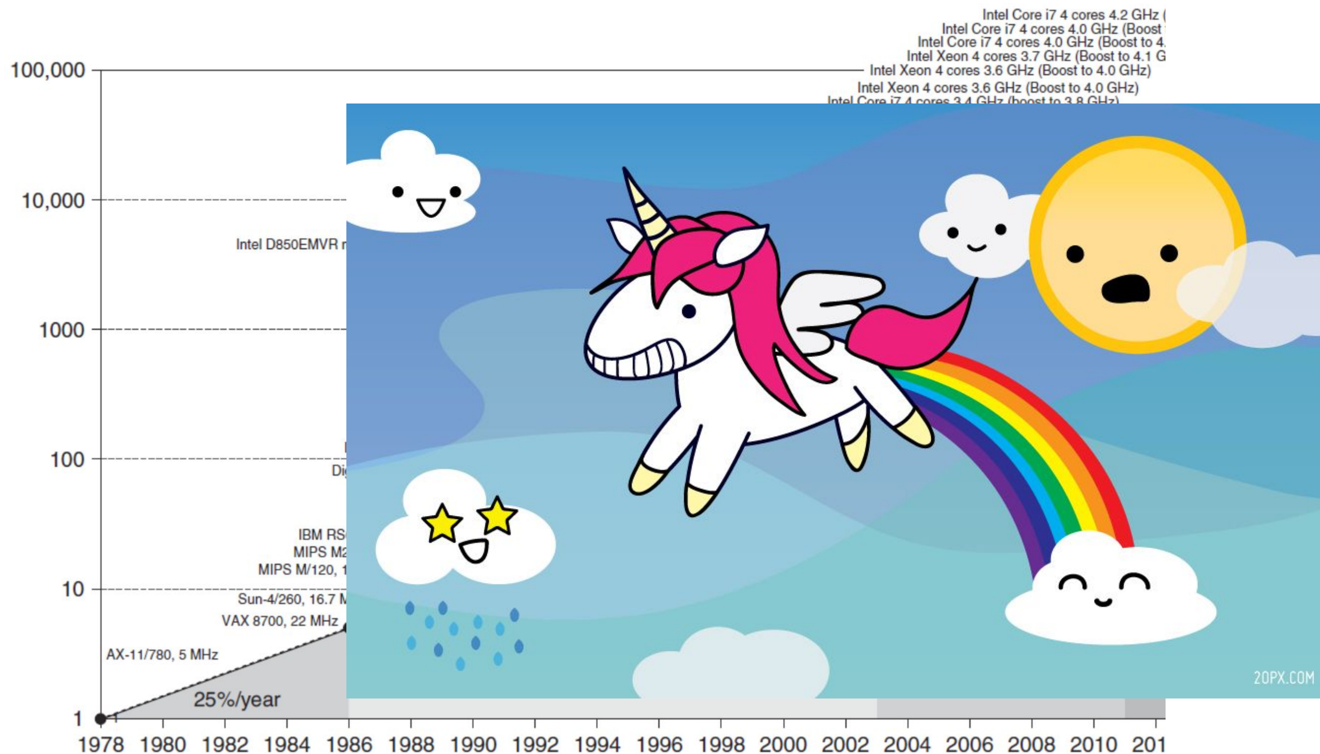


- The main contribution to the gain in microprocessor performance at this stage came by increasing the clock frequency.
- Applications' performance doubled every 18 months without having to redesign the software or changing the source code

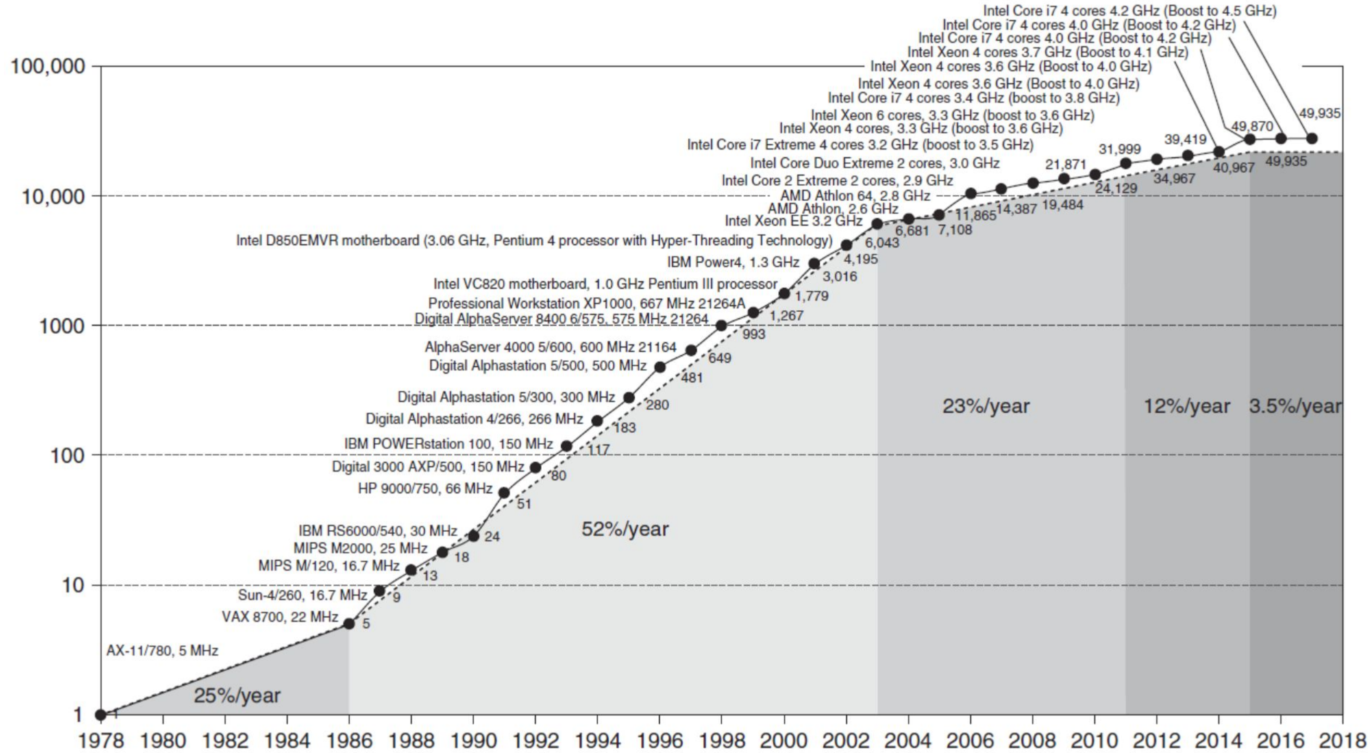
Moore's Law (ctd.)



Moore's Law (ctd.)



Moore's Law (ctd.)



•

CERN

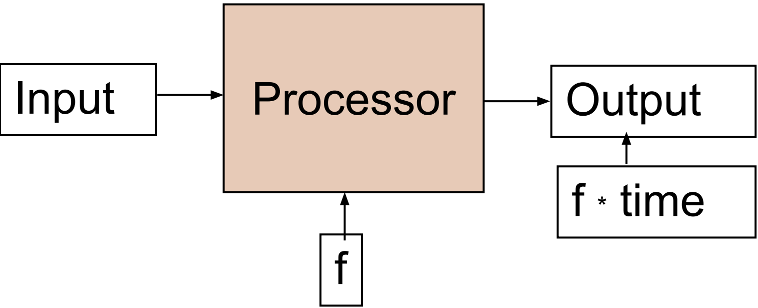
"The party isn't exactly over, but the police have arrived, and the music has been turned way down" (P. Kogge, IBM)

Power and Energy

- Thermal Design Power (TDP)
- Characterizes sustained power consumption
- Used as target for power supply and cooling system
- Lower than peak power (usually 1.5X higher), higher than average power consumption

- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

Consider power in a chip …



Capacitance = C
Voltage = V
Frequency = f
Power = CV^2f

C = capacitance … it measures the ability of a circuit to store energy:

$$C = q/V \qquad q = CV$$

Work is pushing something (charge or q) across a "distance" … in electrostatic terms pushing q from 0 to V:

$$V * q = W.$$

But for a circuit $q = CV$ so

$$W = CV^2$$

power is work over time … or how many times per second we oscillate the circuit

$$\text{Power} = W * f \qquad \text{Power} = CV^2 f$$

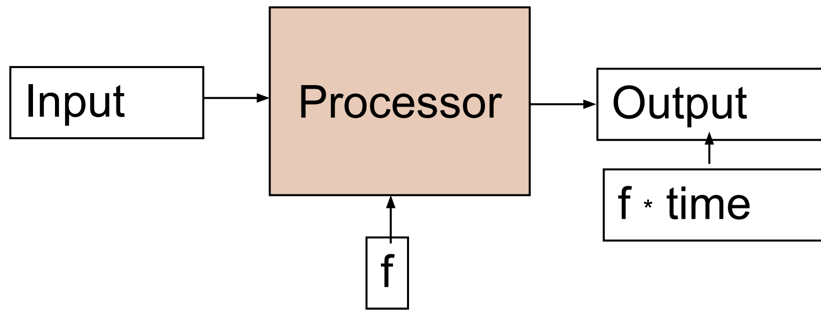
Back to Earth

- The power dissipated by a processor scales as

$$P = QCV^2 f + V I_{\text{leakage}}$$

- Q number of transistors
- C capacity
- V voltage across the gate
- f the clock frequency
- I current
- In the early 2000s, the layer of silicon dioxide insulating the transistor's gate from the channels through which current flows was just five atoms thick and could not be shrunk anymore

… Reduce power for a fixed throughput by adding cores



Capacitance = C
Voltage = V
Frequency = f
Power = CV^2f

Input

Processor

f/2

Processor

f/2

Output

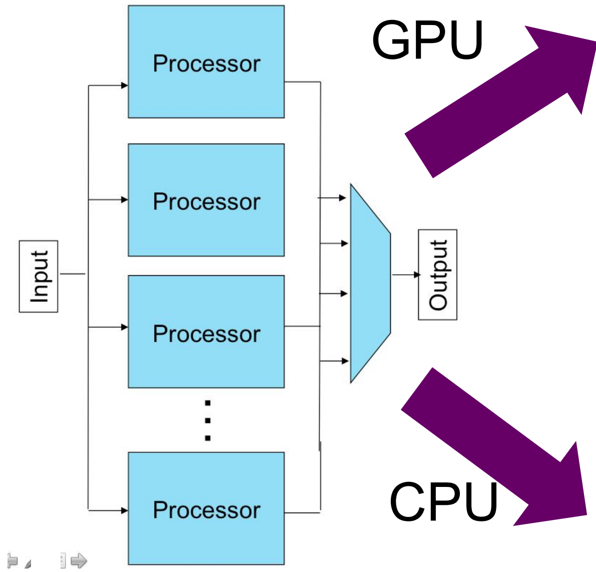
f * time

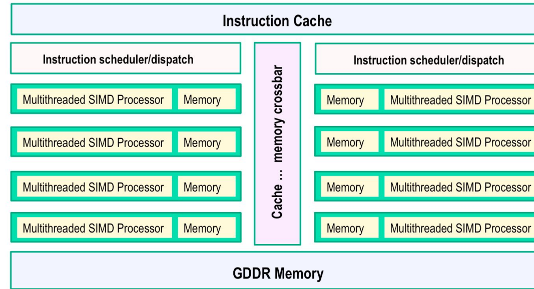
Capacitance = 2.2C
Voltage = 0.6V
Frequency = 0.5f
Power = $0.396CV^2f$

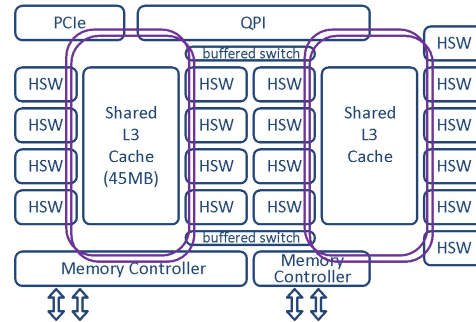
Chandrakasan, A.P.; Potkonjak, M.; Mehra, R.; Rabaey, J.; Brodersen, R.W., "Optimizing power using transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, vol.14, no.1, pp.12-31, Jan 1995

Source: Vishwani Agrawa

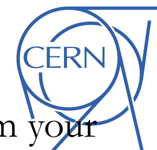
… Many core: we are all doing it



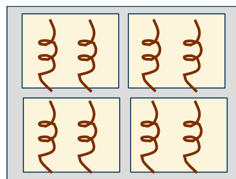




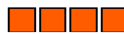
For hardware … parallelism is the path to performance



All hardware vendors are in the game … parallelism is ubiquitous so if you care about getting the most from your hardware, you will need to create parallel software.

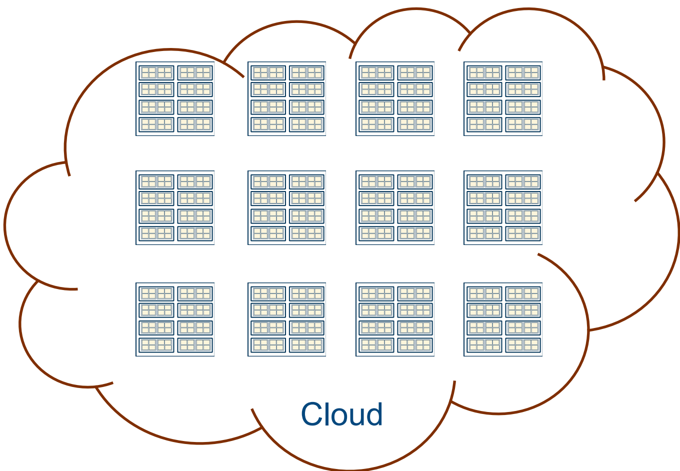


CPU

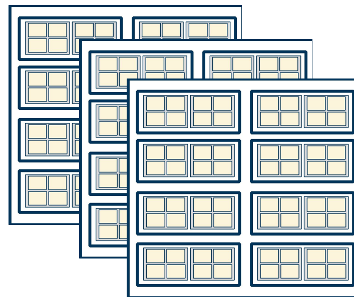


SIMD/Vector

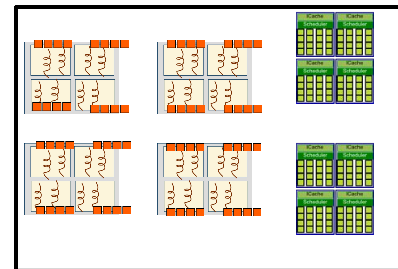
GPU



Cloud



Cluster



Heterogeneous node

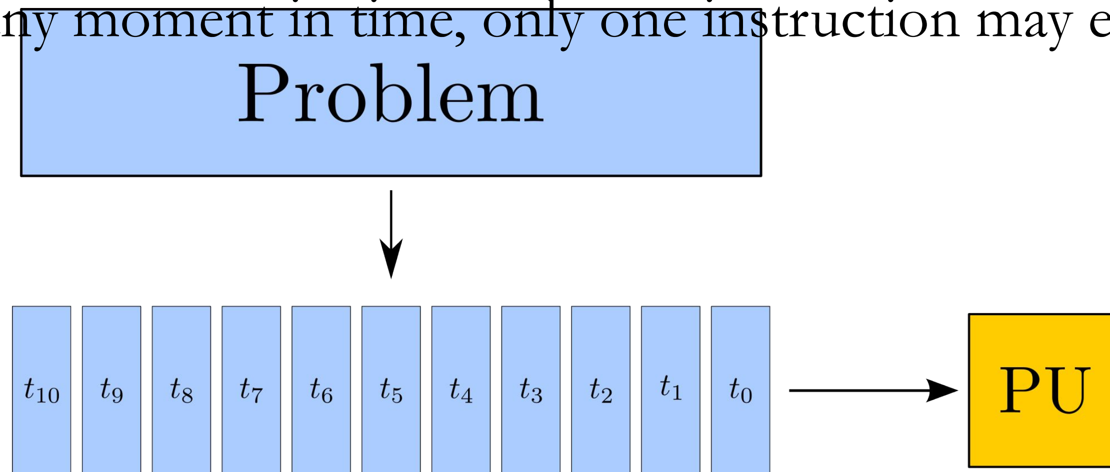
Evolution of system architecture

CERN

- Increased number of Processing Units
- More complex control
  - Pipelining
  - hardware threading
  - out-of-order execution
  - instruction-level parallelism
- Deeper memory hierarchy
- Accelerators
- Interconnects

Serial computation

- Software traditionally written for serial computation:
- the sequence of instructions that forms the problem is executed by one Processing Unit (PU)
- every instruction has to wait for the previous one to be completed before its execution can start
- at any moment in time, only one instruction may execute



Parallel computing

CERN

Parallel computing requires that

- The problem can be decomposed into sub-problems that can be safely solved at the same time
- The programmer structures the code and data to solve these sub-problems concurrently

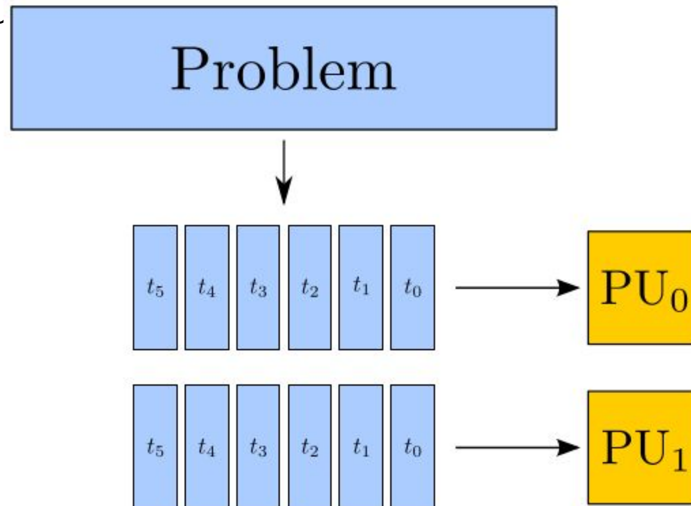
The goals of parallel computing are

- To solve problems in less time (strong scaling), and/or
- To solve bigger problems (weak scaling), and/or
- To achieve better solutions (advancing science)

The problems must be large enough to justify parallel computing and to exhibit exploitable concurrency

Parallel computation

- In parallel computation, if two instructions have no data dependency, they can be executed in parallel, at the same time, by two



Pizza Wall

- How many cooks does a pizzeria need to achieve the best production rate possible?
- If all the ingredients are in the same fridge and there is only one oven? Maybe 1, 2, 64, i

Mitigating the Pizza Wall

CERN

- Reuse of ingredients and tools which are used often: put them on a small table close to you
- Increase the frequency of travels to the fridge
- Increase the amount of ingredients you transfer from the fridge
- If ingredients are located all in the same box in the fridge, you can carry more of them with a single transfer
- Better organization of order of instructions, keeping cooks busy

Memory Wall

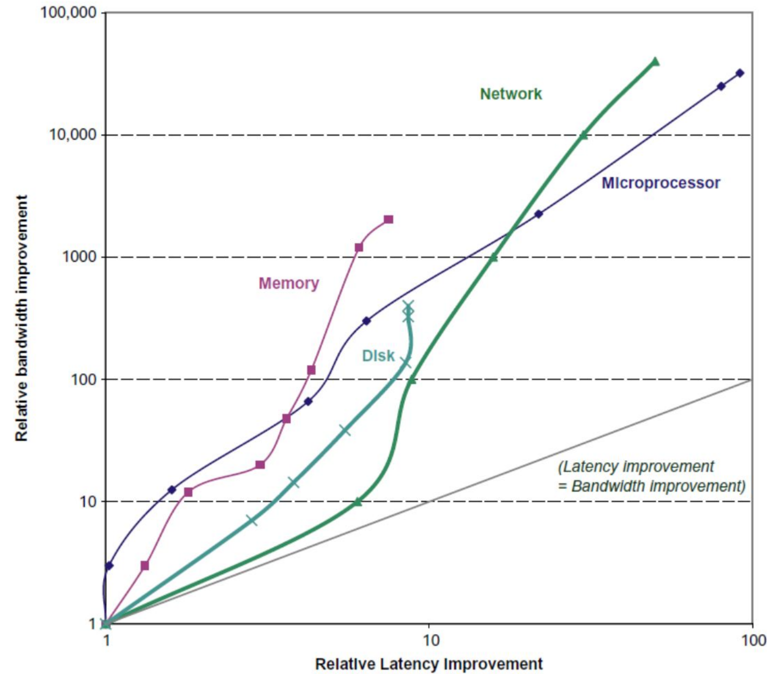
CERN

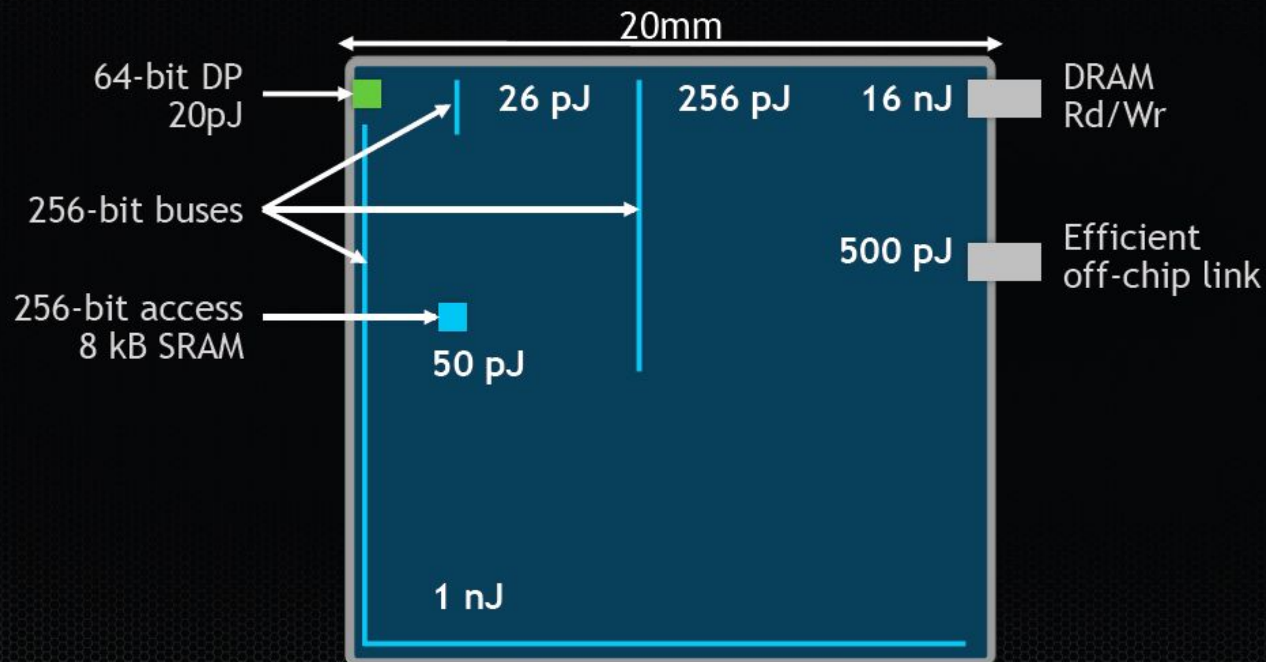
- How many PUs does a program need to achieve the best performance possible?

Mitigating the Memory Wall

- Reuse data and instructions: data and instructions which are used often are stored in a on-chip memory called cache.
- Increase the memory transfer speed: this can be done by increasing frequency, which is limited by the power wall.
- Increase the amount of data to transfer: memory transfers have overheads, which can become negligible if more memory is transferred in one instruction.
- Improve the access pattern to memory: if more processing units are reading adjacent memory locations, they can all be fed by a single memory transfer.
- Better organization of order of instructions, keeping PU busy
- Smarter prefetching

Latency vs Bandwidth





Three orders of magnitude difference between the energy consumption of a double

Conclusion

CERN

- Knowing what's going underneath your code helps you write better high level code
- Parallel computing key to achieve efficient hardware utilization with reasonable power budget

Think parallel and enjoy the school!!!