Introduction to HPC Programming (08.128.612)

Peter-Bernd Otte, Helmholtz-Institute Mainz, 16.10.2018

Why you should be here

- Alternative Title: Boost your analysis
  - But: There is no automated way to do this.

- Main purpose of this course
  - Give you the tools at hand to boost you analysis (less theory)
  - Focus on MSc and PhD students
  - What is so special about MSc/PhD?
    (no time, credit points irrelevant, already settled analysis program)
  - Hands-on during lecture

- Alternatives to this lecture:
  - “HPC” by Bertil Schmidt, 08.079.090:
    - standard lecture with problem classes (credit points included), no hands-on
    - not physics-analysis orientated
- Motivation:
  - There are some people saying, they found the use of C pointers more difficult that parallel programming with “MPI”.

Organisational issues

- Lecture times: Tuesdays, 14:15-15:45, Conference Room 1, HIM Building
- Optimised for target audience:
  - No home work or problem classes
  - No compulsory attendance (please leave me a note, when you are unable to come)
  - Learning Blocks (eg OpenMP, MPI) stretch over several lectures. Please recap the content before next lecture if you missed it.
  - OpenMP / MPI can be used at any HPC computer. Special lectures (later in this lecture) might become special cases.

- Lecture nodes get published on course webpage
  - https://www.hi-mainz.de/research/computing/lectures/

- Practical work:
  - 2 people share one computer (Linux, macOS, Windows)
  - Please form clusters
- Homework:
  - Set up a Computer for next week‘s session with the following software: X-Server for macOS and Linux, MobaXterm for Windows (Alternatively Putty with Xming)
  - Log into Himster 2 Headnode once (as soon as it is online again). Details: https://mogonwiki.zdv.uni-mainz.de/dokuwiki/access
  - Help any time: email or stop by my office.

Lecture Overview

- Typical lecture: 30’ talk + 60’ hands-on

1. Introduction: (1 lecture, today)
    1. Why HPC?
    2. Setup of a HPC (in general terms / Mogon2 / HIMster 2 / Clover)
2. Programming:
    1. Shared memory programming: OpenMP (3 lectures + hands on)
    2. Distributed memory programming: MPI starter (3 lectures + hands on) + advanced (2lectures + hands on)
3. Debugging:
    1. MPI parallel debugging with TotalView (1 lecture + hands on)
    2. OpenMP verification with Intel Inspector XE (1 lecture + hands on)
4. Survey dependent:
    - 5 remaining lectures
      (Hybrid Programming MPI+OpenMP, optimising I/O pattern, usage of “non HPC programs”, special programming languages, …)

Literature

- An Introduction to Parallel Programming, Peter Pacheco, 2011, 978-0-12-374260-5
  - Ebook freely available
- MPI-3.1: A Message-Passing Interface Standard
  - The standard is really readable! https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report-book.pdf
- Introduction to High Performance Computer for Scientists and Engineers, Hager and Wellein, 978-1-4398-1192-4

- courses at HLRS Stuttgart (and more at PRACE) (https://www.hlrs.de/training/)
  - Parallel Programming (*)
  - Performance Optimization and Debugging (*)
  - Computational Fluid Dynamics and Parallelization
  - Scientific Visualization

Basic Requirements

- Linux basics needed:
  - Bash: launch a program with different parameters, write script
  - SSH: generate a key and log into HIMster2
  - modules: list and load different modules
  - gcc: compile with different optimisations
  - (versioning with git)
- Very basic C programming skills

- If not present, familiarise with it OR find the right team mate!

Lecture Today

- Organisational Points (10’)
- Motivation for Cluster Computing (10’)
- Survey (10’)
- Short break (5’)

- Cluster building blocks and our HIMster2 (30’)
- TOP 500 / Green 500
- Linux basics (5’)
- Guided Tour HIMster 2 (20’)

Why HPC?

Why did we gather together?

- Why HPC?

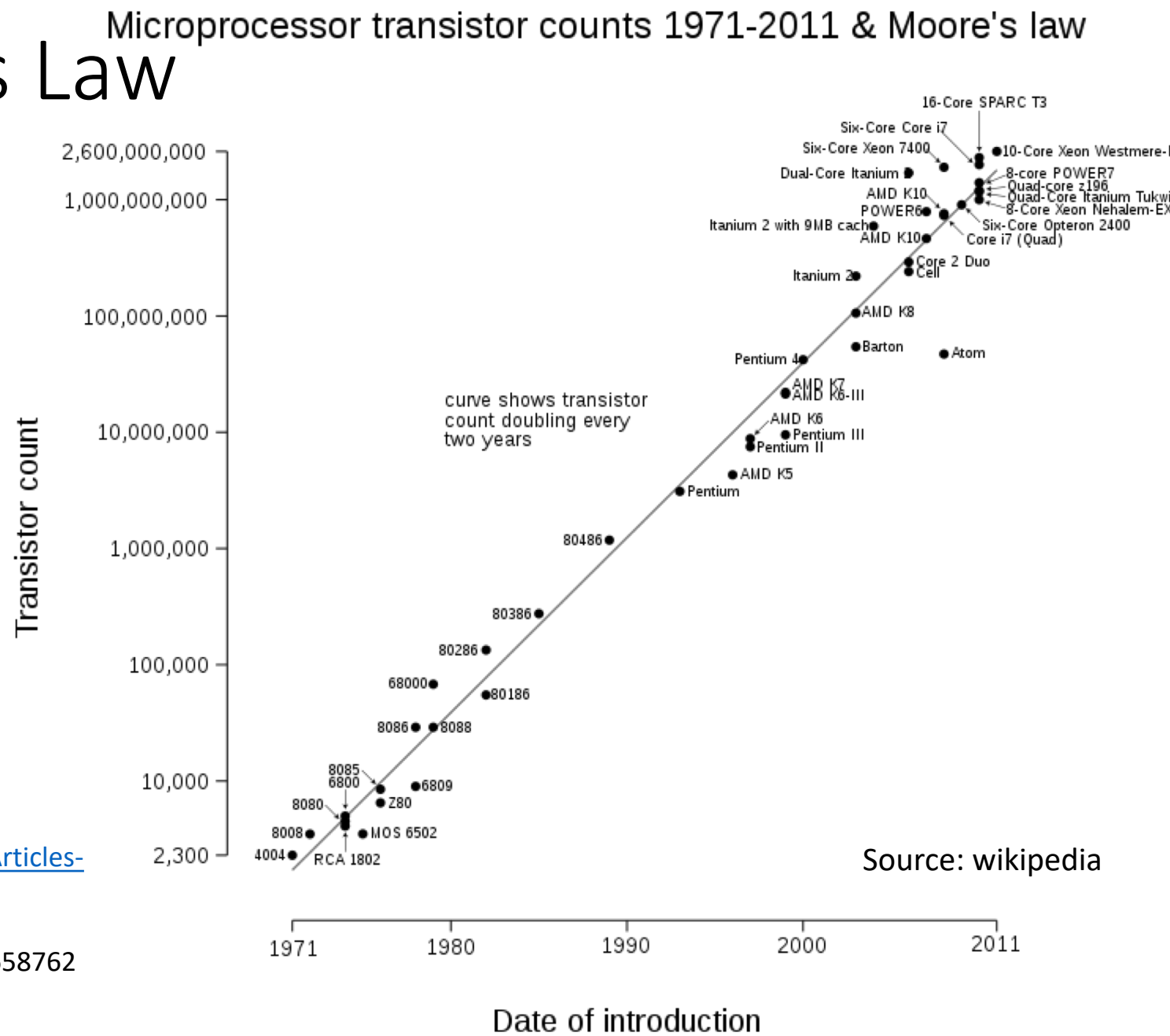
- Intense computational problem → single desktop computer not capable enough

- Run on a “super computer“
  1. <2002: fast single core super computer
  2. Since 2002: parallel systems as super computers

  → Why parallel systems?

The Era of Moore‘s Law

- Moore's law (1965) = observation number of transistors in a IC doubles every ~2a.
- Still valid, no natural law.



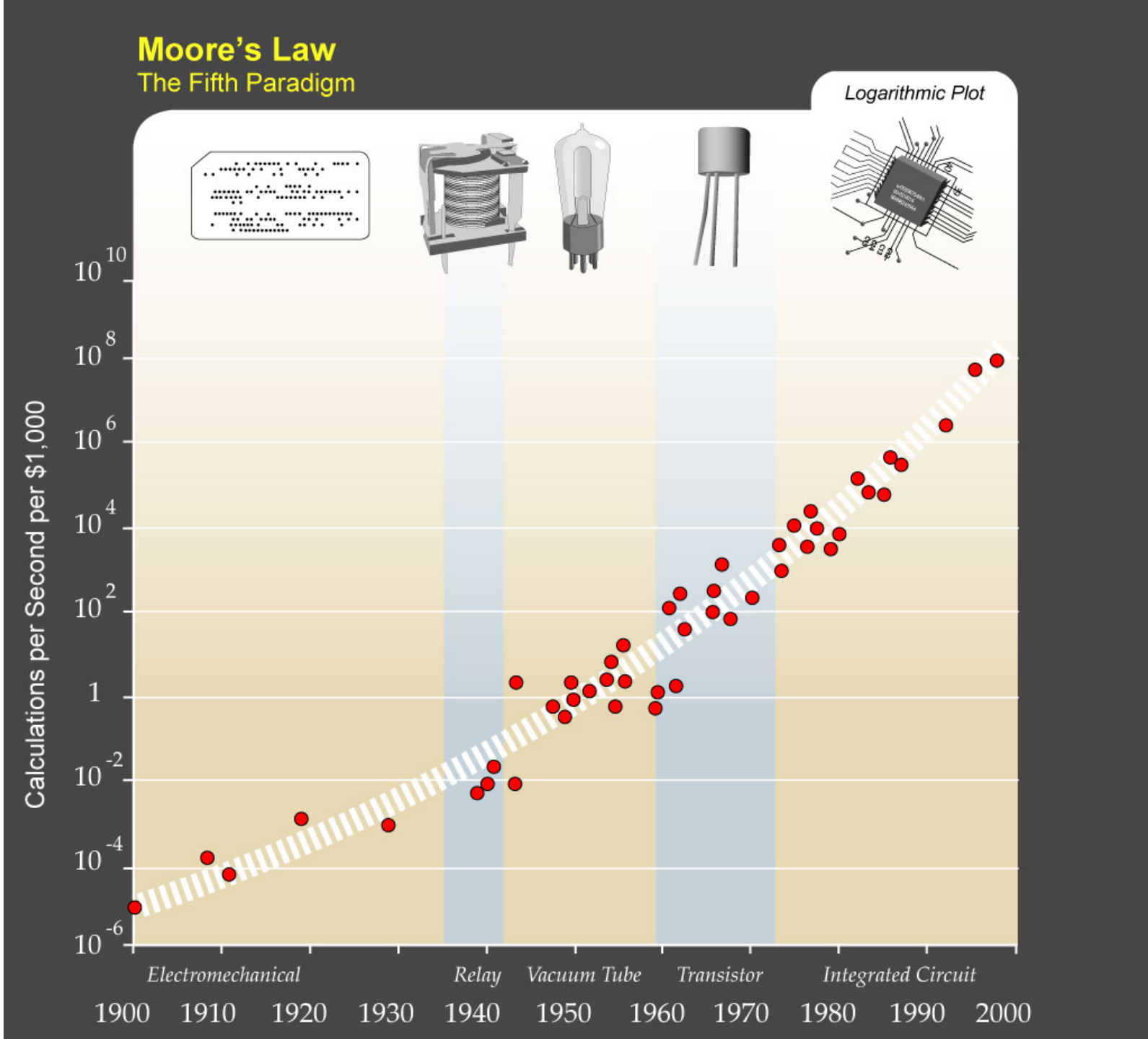
Source: wikipedia

Cramming More Components onto IC (1965): ftp://download.intel.com/sites/channel/museum/Moores_Law/Articles-Press_Releases/Gordon_Moore_1965_Article.pdf or

https://ieeexplore.ieee.org/document/658762?tp=&arnumber=658762

The Era of Moore's Law

- 1900-2000
- source: Wikipedia



Single-Core Performance

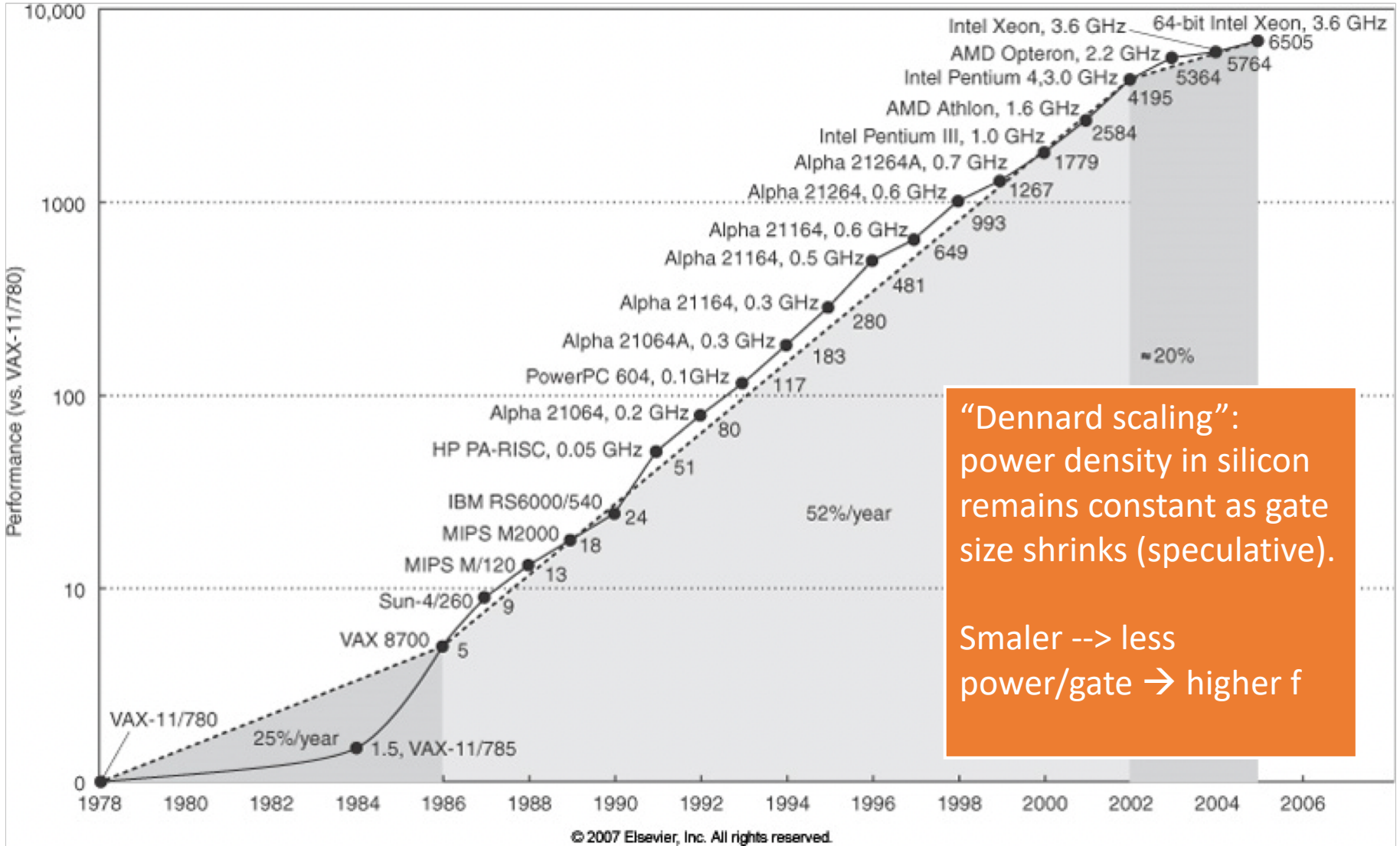
The single core-performance increased by

- <2002: 50%/a
- >2002: 20%/a

Speedup after 10a:

- <2002: ~6000%
- >2002: ~600%

Simply wait for the next CPU release is not enough any longer.

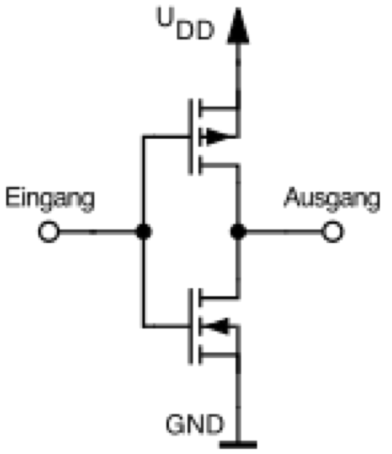


"Dennard scaling": power density in silicon remains constant as gate size shrinks (speculative).

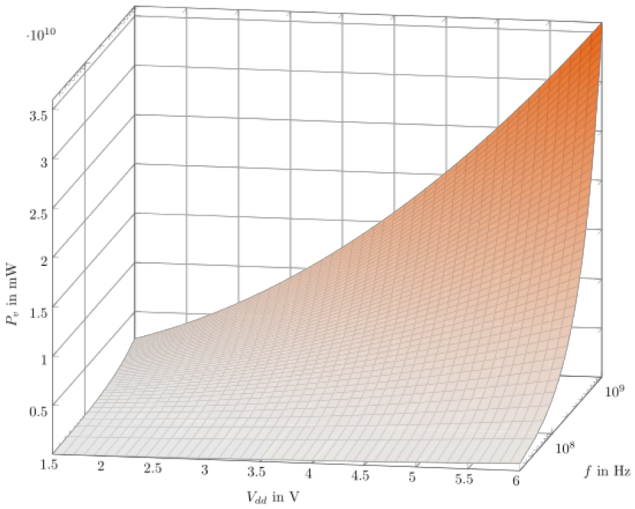
Smaler --> less power/gate → higher f

Why not increasing frequency?

- Core speeds topped out at 2-4 GHz
  - World record standard CPU: 8722.78 MHz with liquid nitrogen cooling (http://hwbot.org/benchmark/cpu_frequency/)
- Problem #1: cooling the chip
- Finding: “Dennard scaling” (constant power density) no longer valid
  - No longer (since 2000’s) true since 90nm gate sizes (leakage current!)
- The two things that consume energy (CMOS gate):
  1. switching state (1 ⇔ 0) (10µW/MHz, prop with f^1.75)
  2. leakage current (10nW / CMOS-Gate, anti-prop with Vdd and gate size)
- Increasing f: increase in power on same area → compensate this: shrinking gate sizes and lower Vdd
  - But: smaller gates have higher leakage current. → New innovations needed. → multi-cores at fixed f to gain performance

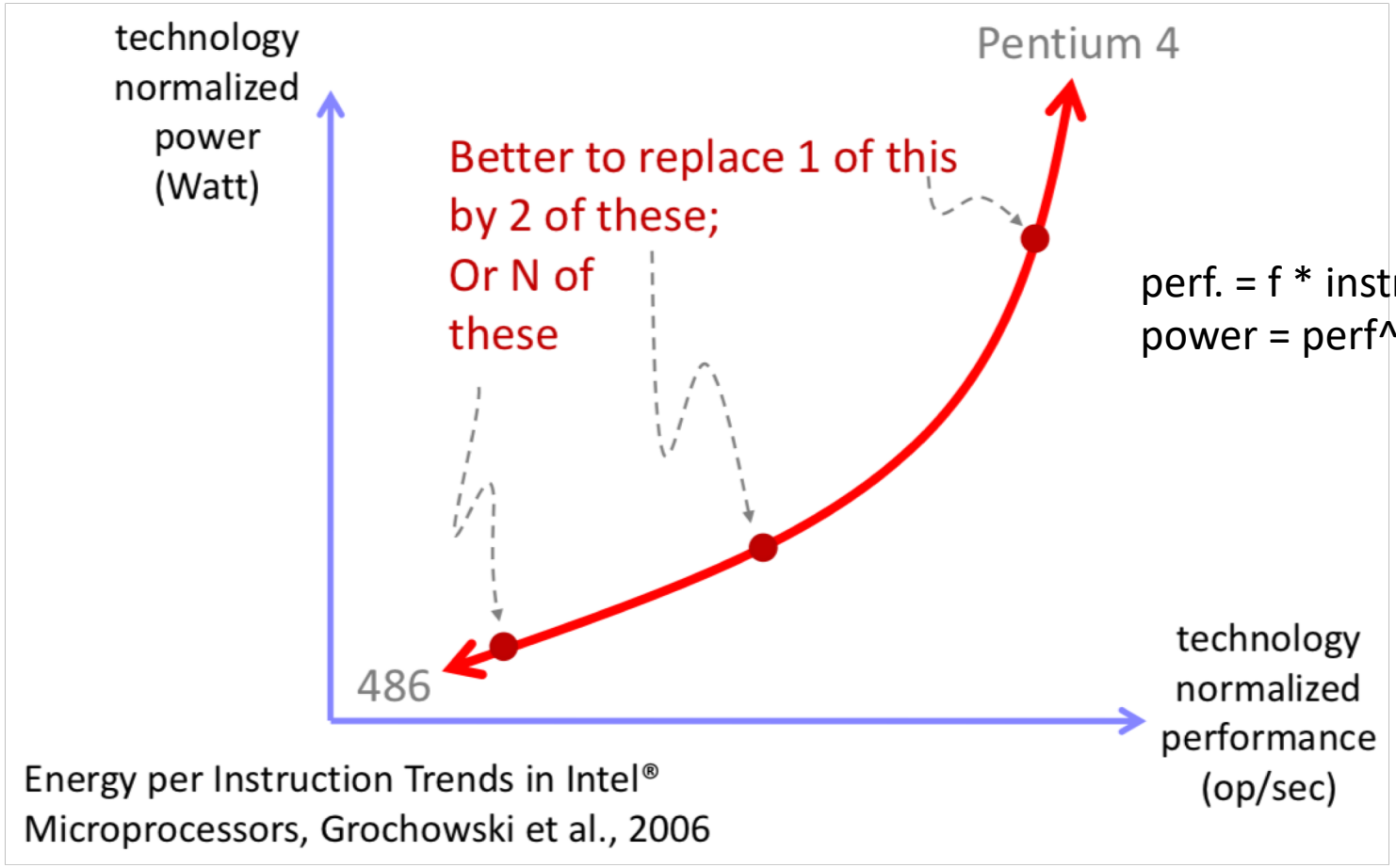


not gate



Power=P(Vdd, f)

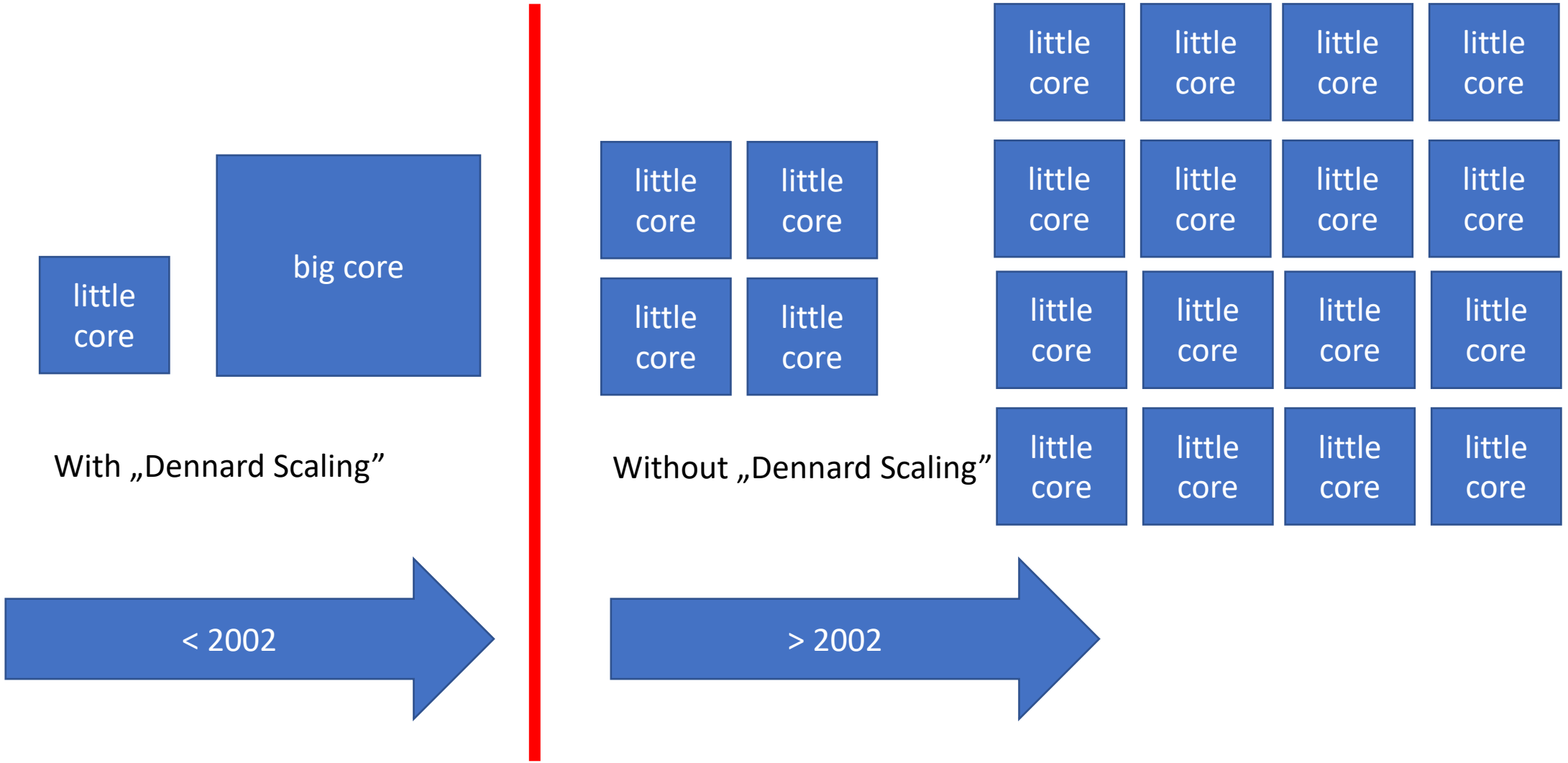
Answer: multicores



Energy per Instruction Trends in Intel®
Microprocessors, Grochowski et al., 2006

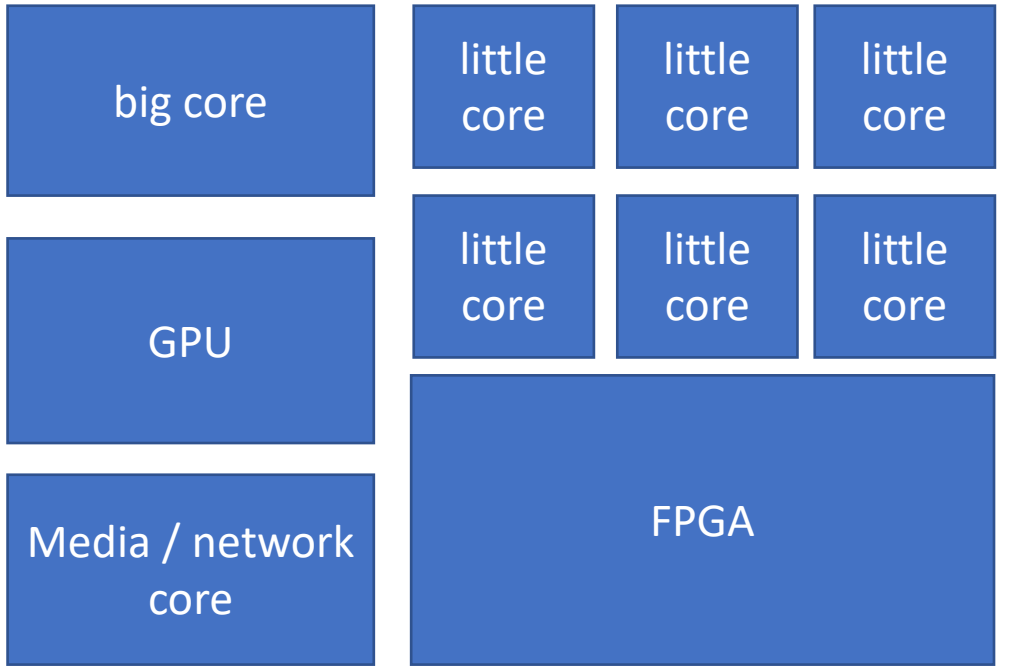
perf. = f * instr./clock
power = perf^1.75 + const.

Moore‘s Law scaling with cores



Future

- All about performance/watt and performance/€



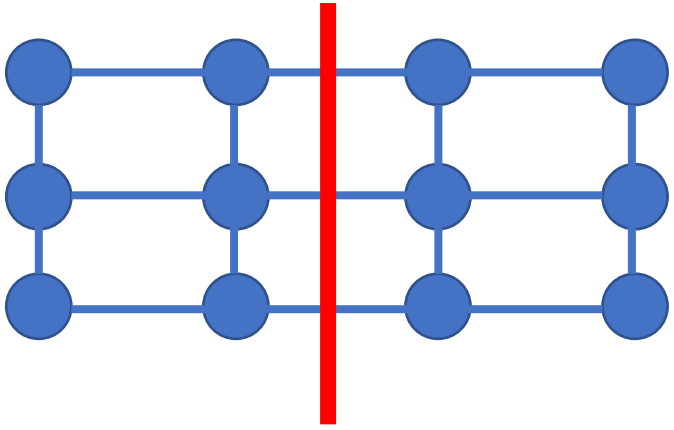
Finally: Why specialised?

Comparison with distributed office computers at HIM

- FLOPS / computer (floating-point operation per second):
  - FLOPS = $f \times N_{cores} \times N_{instr\ per\ cycle}$
  - Intel E5-2670 (2,6 GHz, 8 cores): 2,6GHz × 8 × 8 = 166,4 GFLOPS
- $N_{computers}$: 25 offices / floor, 4 floors, 2 people / office, 1 computer / person = 200.
- 33TFLOPS (Clover = 106TFLOPS, HIMster2/Mogon2: 2801TFLOPS) cluster “for free”

Drawbacks:

- OS: Windows (20%), MacOS (20%), Linux (50%) other (10%) – all on a different version level
- Temperature in office rooms, closed window, 15th July: 0W = 29°C, with 400W = 50°C
- Network: 1GBit/s, Backbone 10GBit/s (HIMster2: 100GBit/s)
  - 10GBit/s / 200 computers / 8 cores = 780kByte/s
  - Compare bisection bandwidth (minimal accumulated bandwidth between any bisections of the network): fat tree ⇔ binary tree
- Storage?
- No node checks, difficult to maintain, reduced availability



bisection bandwidth

Basic concepts

Parallel Programs: Worked out example (1)

Large task: grading 100 exams, 5 questions each

- Approaches:
  - task-parallelism (grading exam questions: one core = one task = one question)
  - data-parallelism (grading exam: one core = one exam = data)

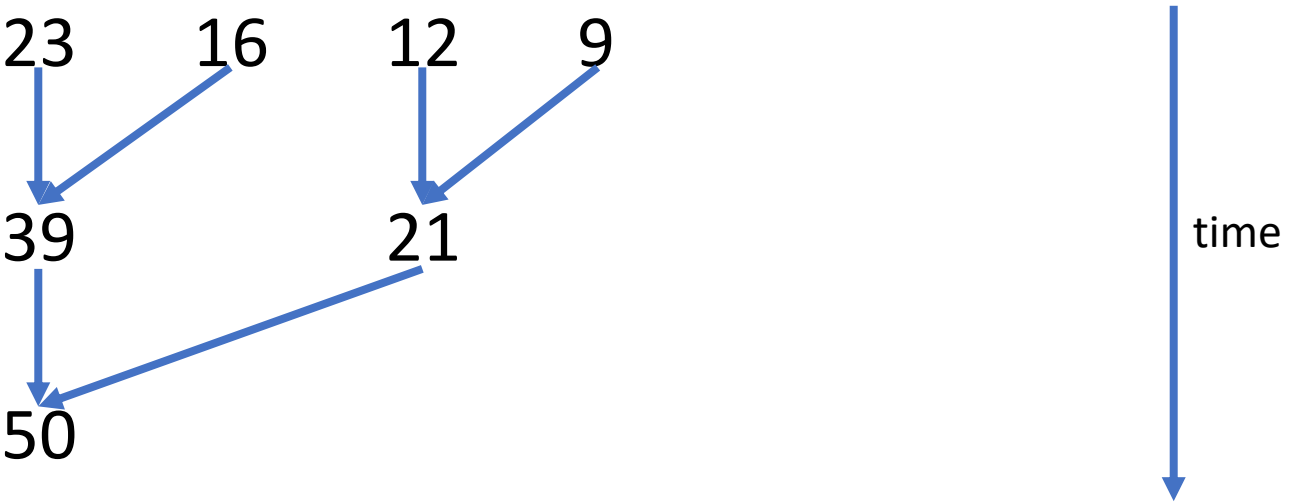
- Check, Coordination of work:
  - Load balancing (assign all cores with equal load) (⇔ your brain while coding)
  - Communication (⇔ Storage, Interconnect)
  - Regular synchronisation (waits!) (⇔ MPI, OpenMP)

- We will write explicitly parallel programs:
  - C language with extensions (OpenMP and MPI)

Parallel Programms: Worked out example (2)

- Task: calculate a large sum of numbers (eg for integration)

- 6,8,9 core 0 | 3,5,8 core 1 | 9,1,2 core 2 | 2,3,4 core 3
- local sums: 23 | 16 | 12 | 9
- collection: 39 | 21
- final sum: 50

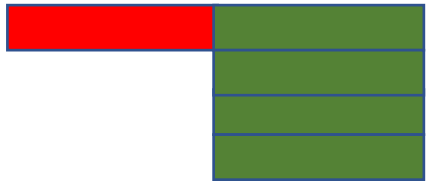


Always check the scaling of your program.

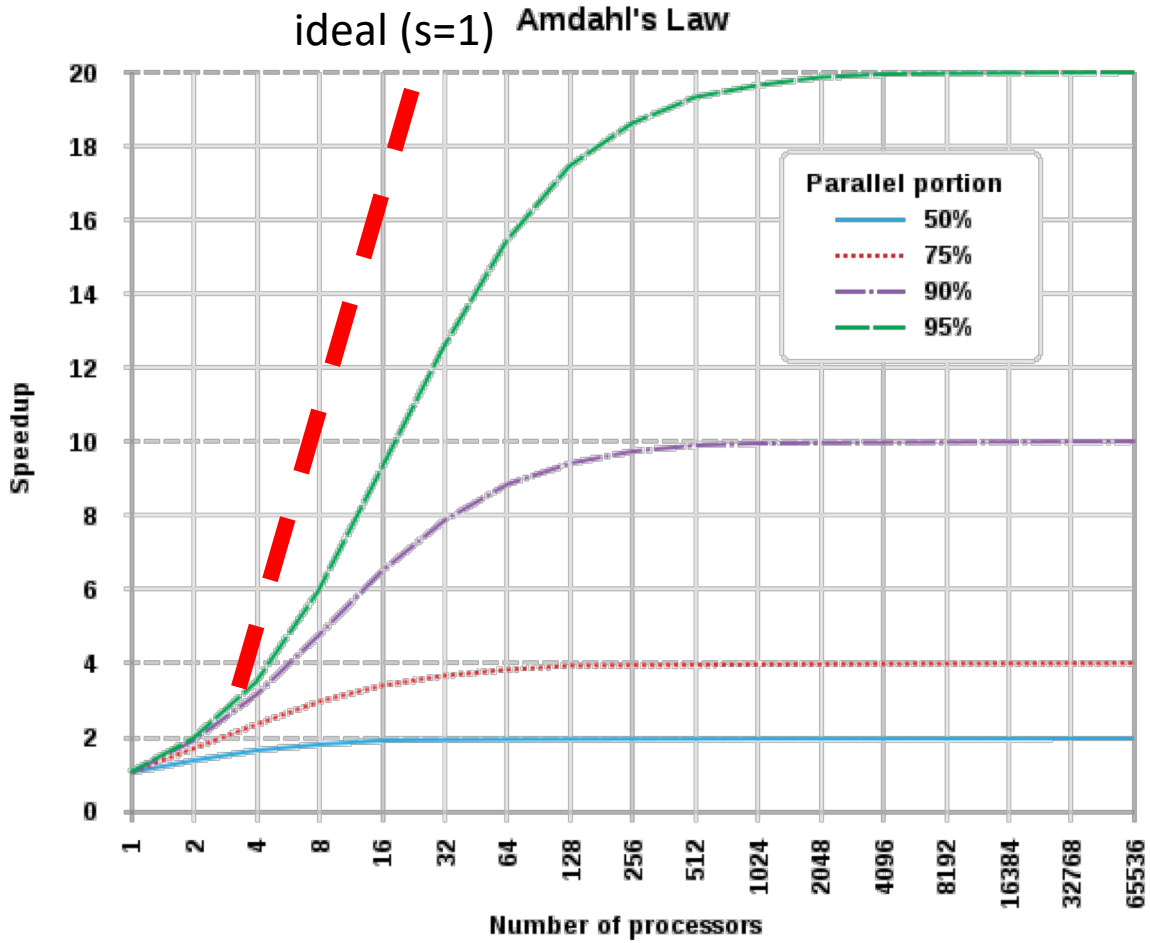
Amdahl's Law

- Given a program consisting of a non-parallelisable and a perfectly parallelisable part

- Fraction s of the non-parallelisable part:
  $T(p)=T_{seq} + T_{par}(p) = T(1) * s + T(1) * (1-s)/p$



- Speed-up: $S(p) = (1+(1-s)/p)^{-1}$
  - p → inf: S(p) = 1/s
  - If S(p) > 1/s → "super-scaler speedup", problem fit's into CPU cache.



More Conventions

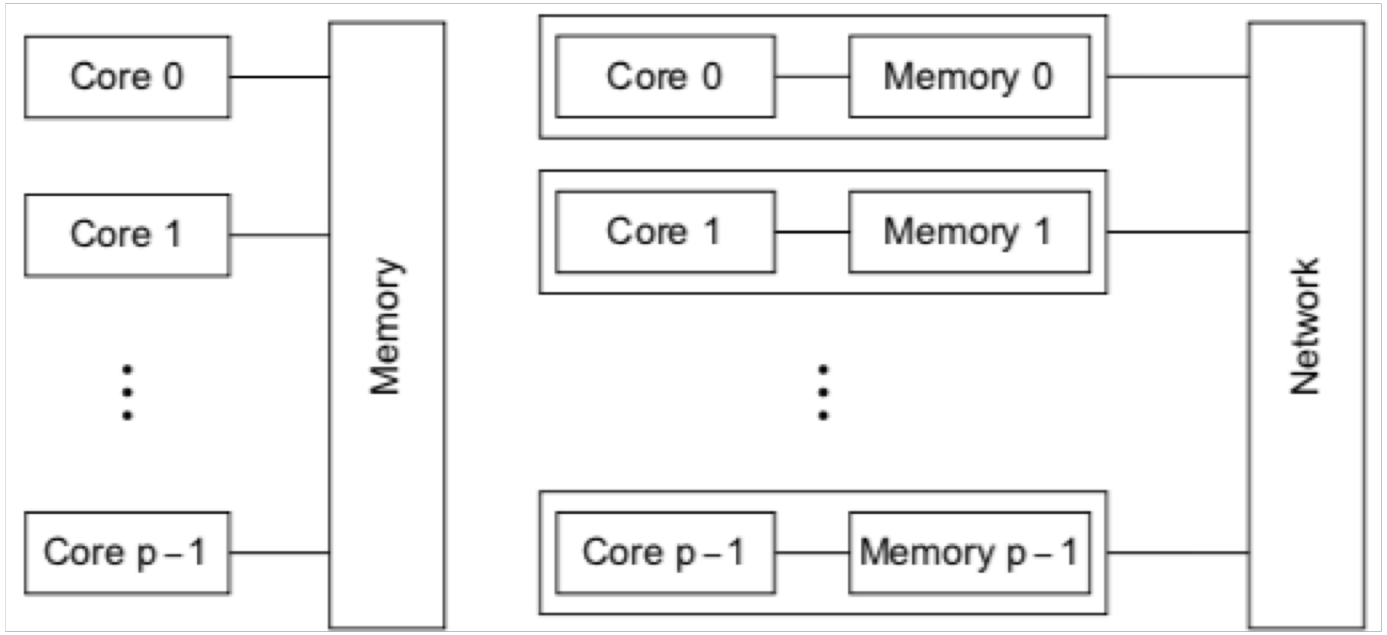
- Concurrent computing
  - Single program, inside: multiple tasks can be *in progress* at any time.
- Parallel computing
  - Single program, inside: multiple tasks *cooperate closely*
  - tasks run on cores with a very high-speed interconnect, like inside a CPU
- Distributed computing
  - Many programs. These may need to cooperate with each other
  - “loosely coupled”, but in reality: None of these programs may crash.

Shared System: Why two C extensions?

(a) Shared-Memory system:
- Each core can read/write each memory location
- Coordination of cores via shared-memory locations
- Use OpenMP
- Small projects. HIMster2: up to 32 cores/node

(b) Distributed-Memory system:
- Each core has private memory
- Cores explicitly sending messages for data exchange and coordination
- MPI
- Several nodes of a cluster

- Hybrid-Programming:
  - OpenMP+MPI



(a) (b)

Trivial vs full usage of HPC

- Trivial parallelisation:
  - Run your analysis several times (with different parameters)
  - Out of the box with any non-interactively linux program
  - Outcome / speedup unclear, but works very good for 10-100 jobs in parallel

- Full usage:
  - No automated process to convert a single-core to a multi-core program
  - Write parallel code: This is the content of this lecture starting next week.
  - Use parallel programs: We can discuss such cases.

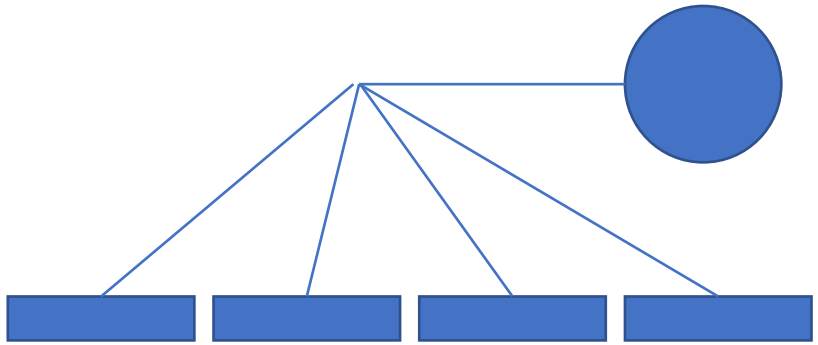
Survey

- Your Name:
- Your Uni-Mainz account name (to get access to the lecture reservation):
- Short description of your analysis:

- Embedded in lager collaboration? (Panda, BES, A1, A2, Atlas, etc.)
- Programming language:
- Operating system:
  - Or Vendor (if closed source):
- Typical amount of data storage (MB, GB, TB?):
- Typical RAM usage per call:
- Single thread / multi threated program? Yes/No
- Typical runtime per call:
- Typical number of calls per analysis (or per day):
- Boundaries? (RAM/CPU/Storage)

What is High Performance Computing (HPC)

- Basic building blocks are:
  1. compute nodes
  2. fast interconnect
  3. parallel file system



- Usage remotely, non interactively


HIMster compute nodes
8 racks


Cooling power
for up to 750kW


Power and OmniPath Interconnect

HlMster II Specs

- 320 Compute Nodes (216 theory, 100 experiment, 4 dev) in 8 racks
  - dual socket Intel 6130 @ 2.1GHz (à 16 cores)
  - 3GB RAM /core
  - OmniPath 100 Gbit/s interconnect
  - 400 GB local SSD scratch
  - https://mogonwiki.zdv.uni-mainz.de/dokuwiki/nodes
- Parallel File System: 747TB Lustre volume
- Software
  1. organized in modules
     - eg: module avail; module load devel/MariaDB/5.5.52-clientonly
     - See: https://mogonwiki.zdv.uni-mainz.de/dokuwiki/setting_up_environment_modules
  2. More via nfs mount: /cluster (no cvmfs)

HIMster II

- HIMster II, Mogon Iia and Mogon IIb form a compound state
  - share login nodes, maintenance servers
  - interconnect: OmniPath (100GBit/s)
- situated in the institute's basement computing room, 660kW
- 2PFlops Linpack (20% contributes HIMster II)
- although calculation on all clusters is possible, use HIMster II
- account registration via PI of HIM or it@him.uni-mainz.de.
  University of Mainz account is mandatory (→ HIM Admin will contact you).
- ssh pbotte@miil01-miil04 (only ssh-key login possible, login Mogon I first via password)
- home directory: Shared with Mogon I, quota 300 GB
- More info: https://mogonwiki.zdv.uni-mainz.de/dokuwiki/ssh_from_outside
- Rules apply: https://www.en-zdv.uni-mainz.de/regulations-for-use-of-the-data- center/

Comparison with its predecessor / do's

- Per core memory bandwidth
  - Clover, HIMster II = 5.6 GByte/sec
  - HIMster = 3.8 GByte/sec
- More memory per core
- HIMsterII has Skylake CPUs (eg AVX512 avail.)
- Storage / Parallel File system:
  - NO BACKUP of data
  - Try to use large files: Source code should be in /home/
  - Try not to put too many files into one directory (less than 1k)
  - Try to avoid too much metadata load:
    - DO NOT DO ls –l unless you really need it
    - In your scripts avoid excessive tests of file existence (put in a sleep statement between two tests say 30 secs)
    - Use lfs find rather than GNU tools like find
    - Use `O_RDONLY | O_NOATIME` (readonly and no update of access time)

Batch System: SLURM

- Introduction and docu:
  - https://mogonwiki.zdv.uni-mainz.de/dokuwiki/slurm_submit
  - https://slurm.schedmd.com/tutorials.html

- account to use: m2_himkurs
- Reservation: himkurs
- Submit into partition: parallel
  - srun --pty -p parallel -A m2_himkurs --reservation kurstest bash -i
- Check what is running: squeue -h | grep pbotte
  - 1184615_79 parallel N203r001 **pbotte** R 1:00:40 52 z[0367-0386,0403-0413,0430-0450]
  - SSH login into your occupied nodes possible: eg ssh z0367