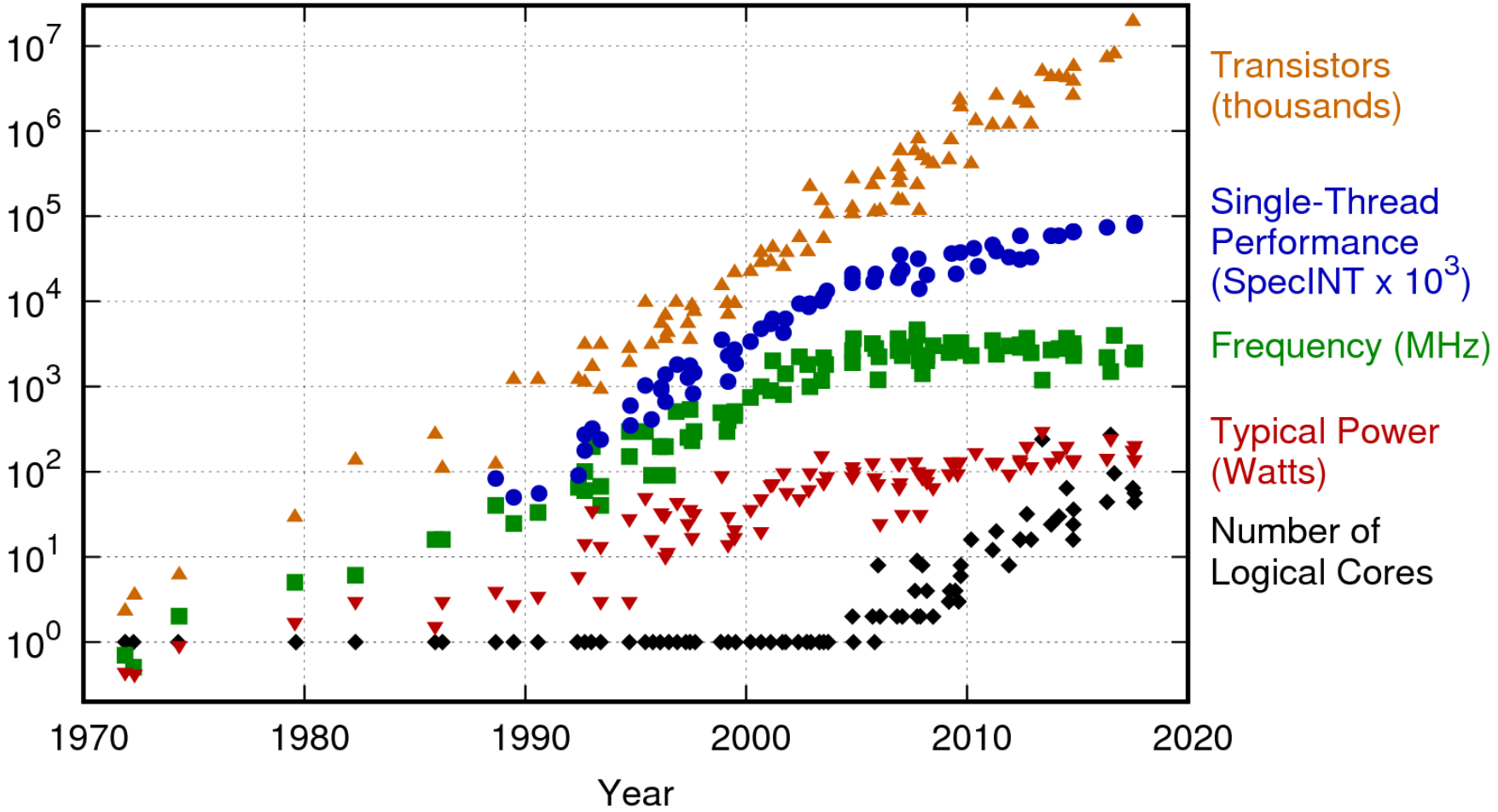
Programming in R and Python

Lecture 11: Parallel computing

Adam Gudyś
Silesian University of Technology
2020

Microprocessors over years



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 2010-2017 by K. Rupp

Microprocessors over years

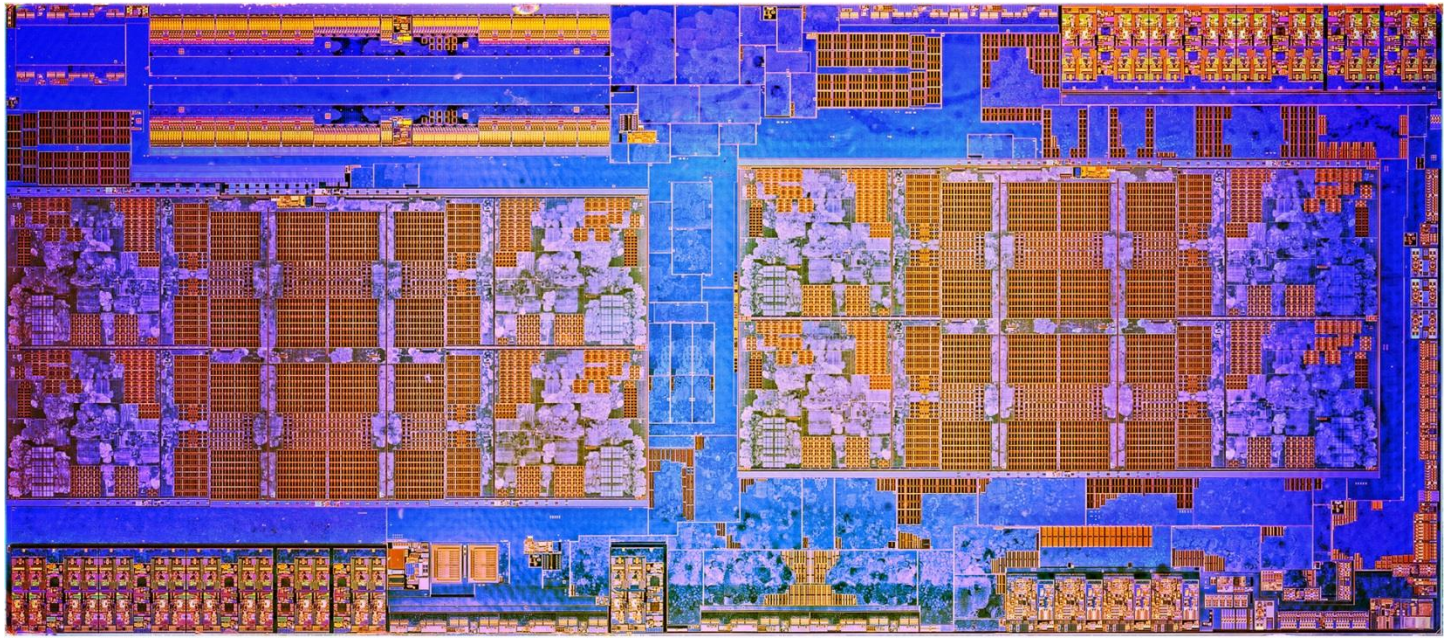
Up to mid 00's:

- CPUs were single core,
- performance gain due to frequency increase,

From mid 00's (Intel Core2Duo):

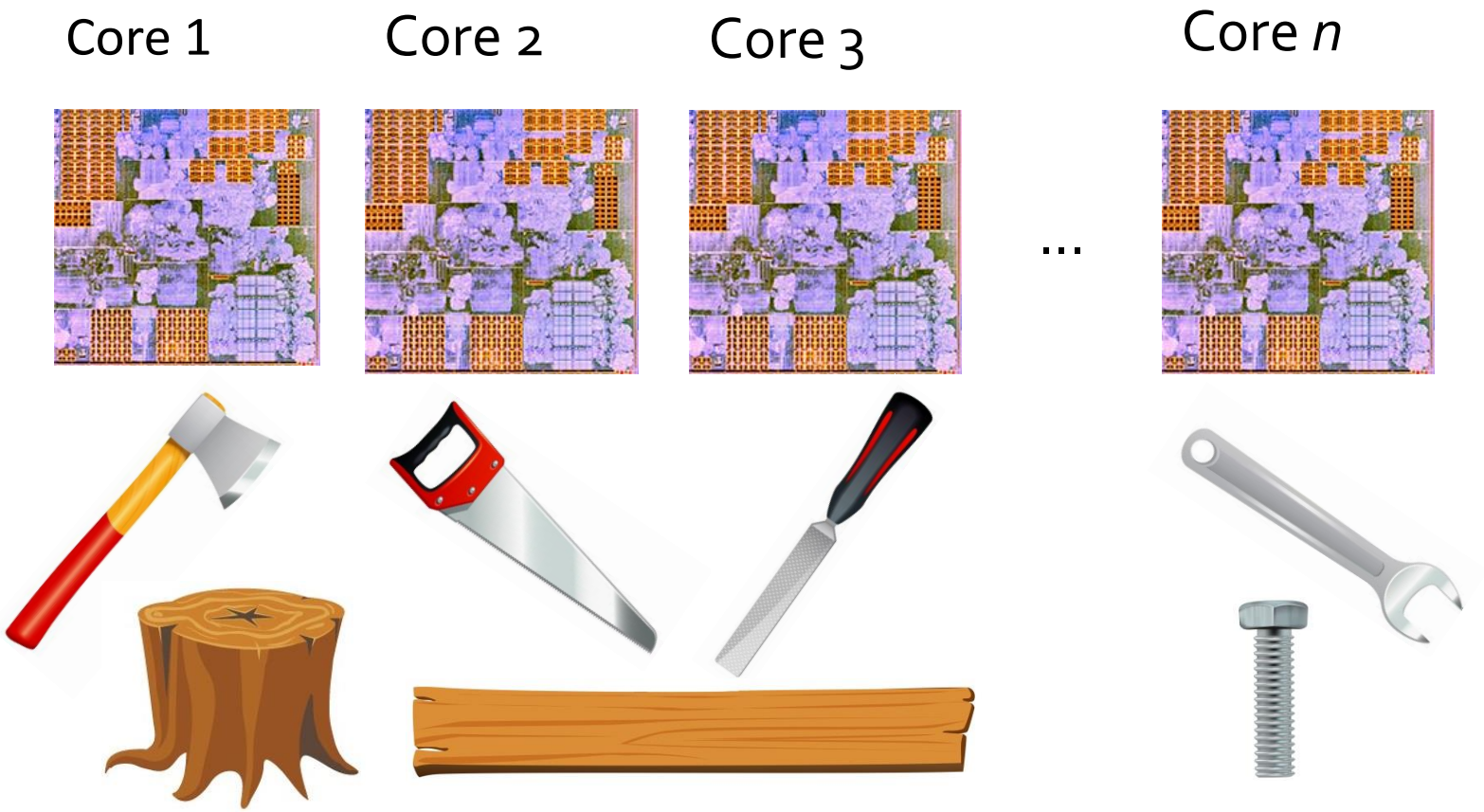
- frequency limited by physical bounds,
- miniaturization continued
  (65nm in 2006 to 7nm in 2018),
- power consumption per transistor still lowered,
- multiple cores on a single die.

AMD Ryzen 2700 (April 2018)



cores: 8×3.6 GHz (16 threads)
cache: 8×32 KB L1, 8×256 KB L2, 2×8 MB
price: 260$

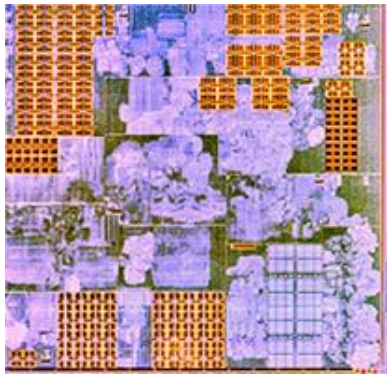
Task parallelism



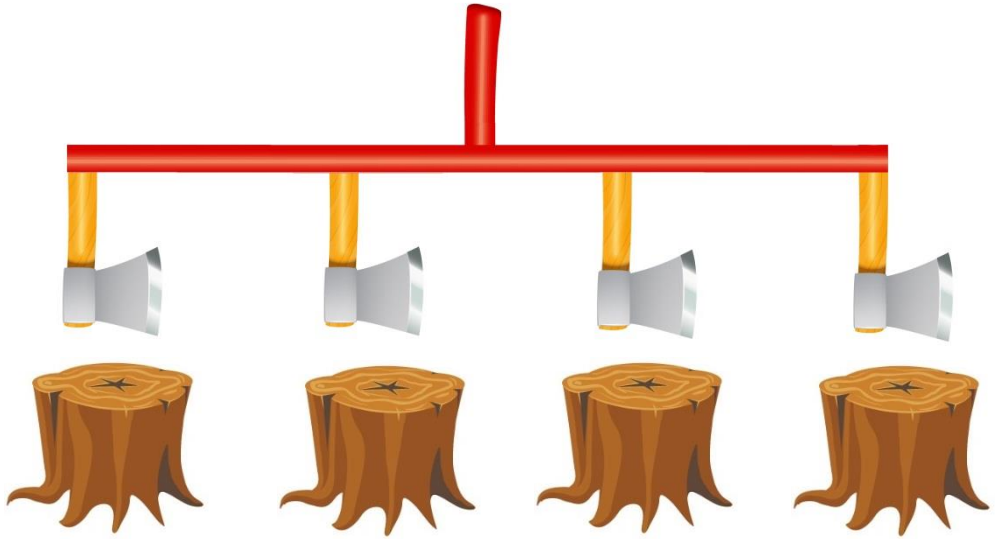
Data parallelism

Modern CPU cores are vector units:

- single instruction on multiple data (SIMD),
- SIMD extensions: SSE, SSE2, SSE4, AVX, AVX2, AVX-512.



=



Doing things in parallel

Task parallelism:

- easy to obtain by invoking processes or threads,
- supported by many languages (inc. Python and R),
- CPU computational resources assigned by OS.

Data parallelism:

- code must be compiled to use SIMD instructions,
- direct support in few languages (C++, Fortran),
- some Python/R packages employ SIMD (eg. Intel Math Kernel Library).

Code time

Things to remember

- determine bottlenecks (Amdahl's law),
- check if the algorithm is arithmetic-bound,
- balance workload evenly with minimum no. of tasks,
- avoid synchronization,
- manage cache memory carefully,
- remember the overheads,

Thank you for your attention!