Parallel programming
Introduction

PRINCETON
UNIVERSITY

Parallel Computing

- Moore’s law
- Parallel processing
- Hardware evolution
- Parallel computing languages
- Open Multi Processing (OpenMP)
- Message Passing Interface (MPI)
- Graphical Processing Units (GPU)

Moore’s law

Gordon Moore: founder of Intel.

Statement of the law:

The number of transistors that can be placed on an integrated circuit at a reasonable cost doubles every two years.

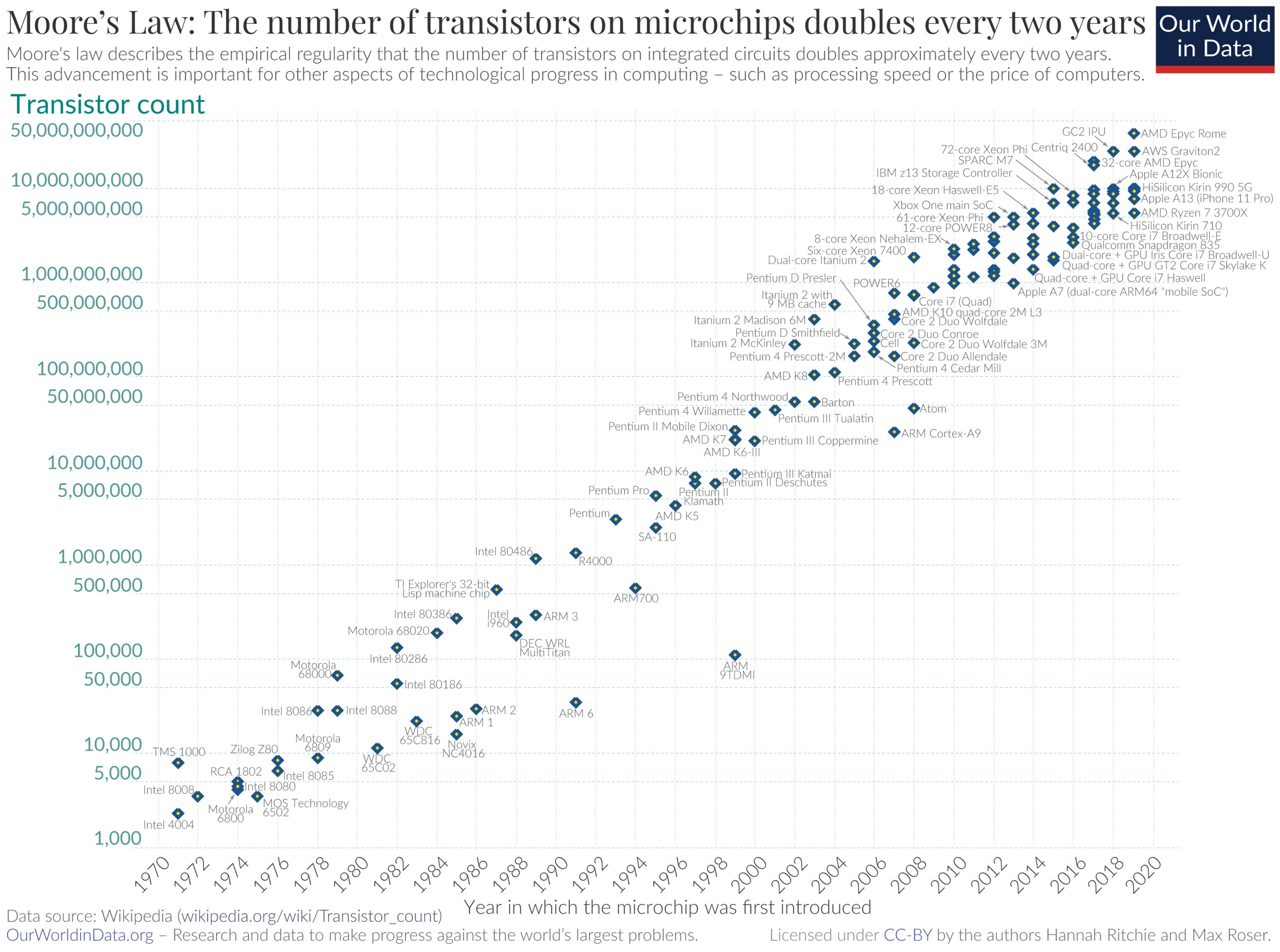
Hardware evolution:

- Processor frequency reached a plateau around 3GHz since 2002-2004 but Moore’s law is still true
- More cores per chip (many cores architectures, GPU...)

Energy consumption and cost:

- Dissipated electric power scales at clock frequency to the cube.
- Dissipated power by square cm is limited by cooling

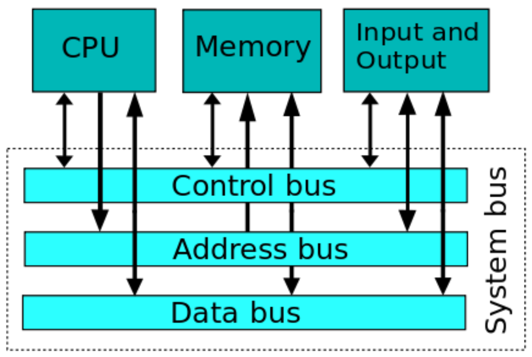
Moore’s law


Moore’s Law: The number of transistors on microchips doubles every two years
Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.
Our World in Data
Transistor count
50,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
Year in which the microchip was first introduced
GC2 IPU
AMD Epyc Rome
72-core Xeon Phi
Centriq 2400
AWS Graviton2
SPARC M7
32-core AMD Epyc
IBM z13 Storage Controller
Apple A12X Bionic
18-core Xeon Haswell-E5
HiSilicon Kirin 990 5G
Apple A13 (iPhone 11 Pro)
Xbox One main SoC
AMD Ryzen 7 3700X
61-core Xeon Phi
HiSilicon Kirin 710
12-core POWER8
10-core Core i7 Broadwell-E
8-core Xeon Nehalem-EX
Qualcomm Snapdragon 835
Six-core Xeon 7400
Dual-core + GPU Iris Core i7 Broadwell-U
Dual-core Itanium 2
Quad-core + GPU GT2 Core i7 Skylake K
Pentium D Presler
Quad-core + GPU Core i7 Haswell
POWER6
Apple A7 (dual-core ARM64 "mobile SoC")
Itanium 2 with 9 MB cache
Core i7 (Quad)
AMD K10 quad-core 2M L3
Itanium 2 Madison 6M
Core 2 Duo Wolfdale
Pentium D Smithfield
Core 2 Duo Conroe
Itanium 2 McKinley
Cell
Core 2 Duo Wolfdale 3M
Pentium 4 Prescott-2M
Core 2 Duo Allendale
Pentium 4 Cedar Mill
AMD K8
Pentium 4 Prescott
Pentium 4 Northwood
Barton
Pentium 4 Willamette
Atom
Pentium III Tualatin
Pentium II Mobile Dixon
AMD K7
Pentium III Coppermine
ARM Cortex-A9
AMD K6-III
AMD K6
Pentium III Katmai
Pentium II Deschutes
Pentium Pro
Pentium II Klamath
Pentium
AMD K5
SA-110
Intel 80486
R4000
TI Explorer's 32-bit Lisp machine chip
ARM700
Intel 80386
Intel i960
ARM 3
Motorola 68020
DEC WRL MultiTitan
Intel 80286
ARM 9TDMI
Motorola 68000
Intel 80186
Intel 8086
Intel 8088
ARM 2
ARM 6
WDC 65C816
ARM 1
Motorola 6809
Novix NC4016
TMS 1000
Zilog Z80
WDC 65C02
RCA 1802
Intel 8085
Intel 8008
Intel 8080
MOS Technology 6502
Motorola 6800
Intel 4004
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world’s largest problems.
Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

The Memory Wall

Von Neumann Architecture

Programming instructions (stored-program) and data share the memory unit. They are loaded to the Central Processing Unit (CPU) for execution. Results are loaded back to the memory unit.



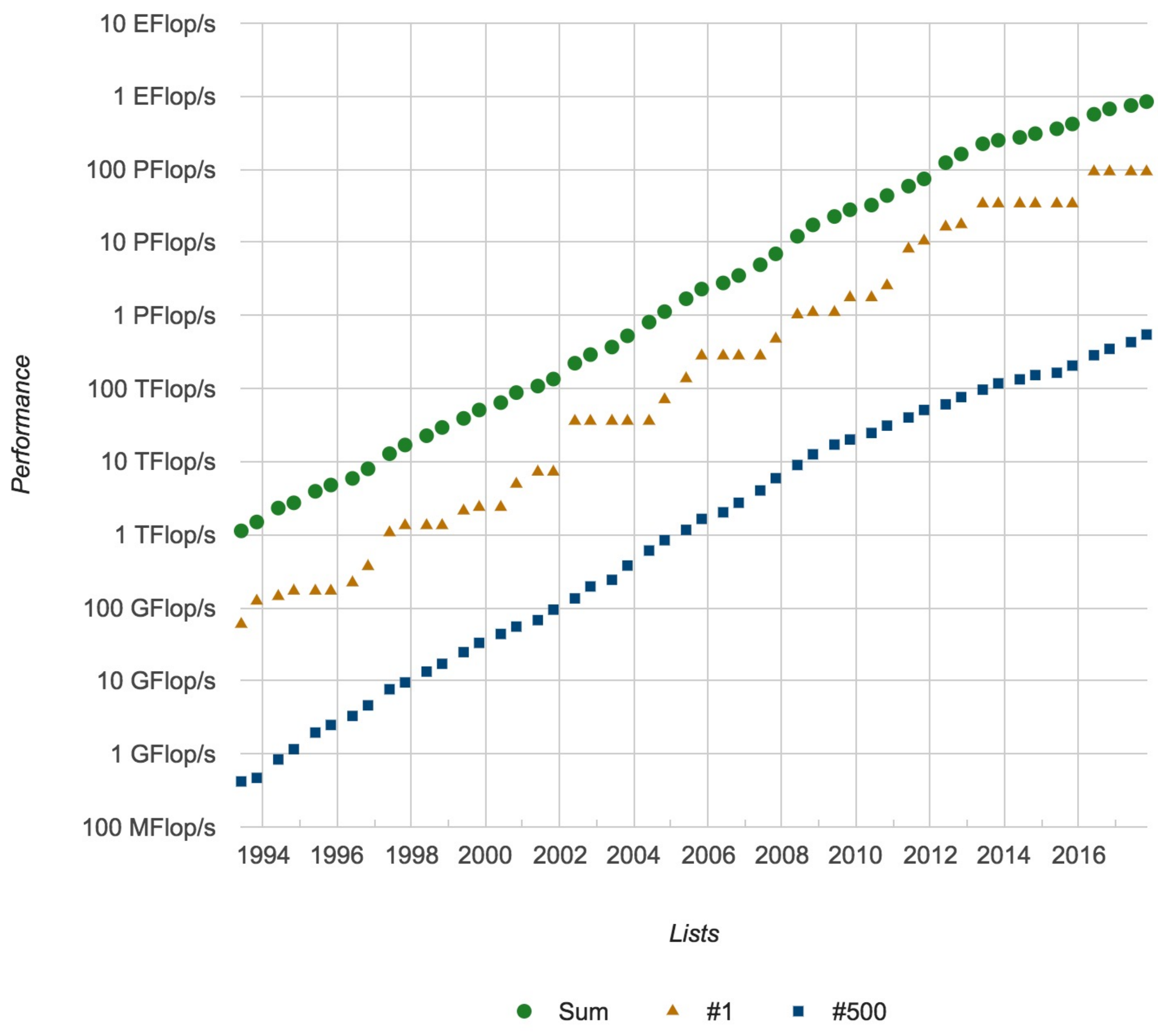
The Von Neumann Bottleneck:

- Memory bandwidth is not increasing as quickly as processor computing power
- Memory latency is decreasing very slowly
- Number of computing cores per memory units is increasing

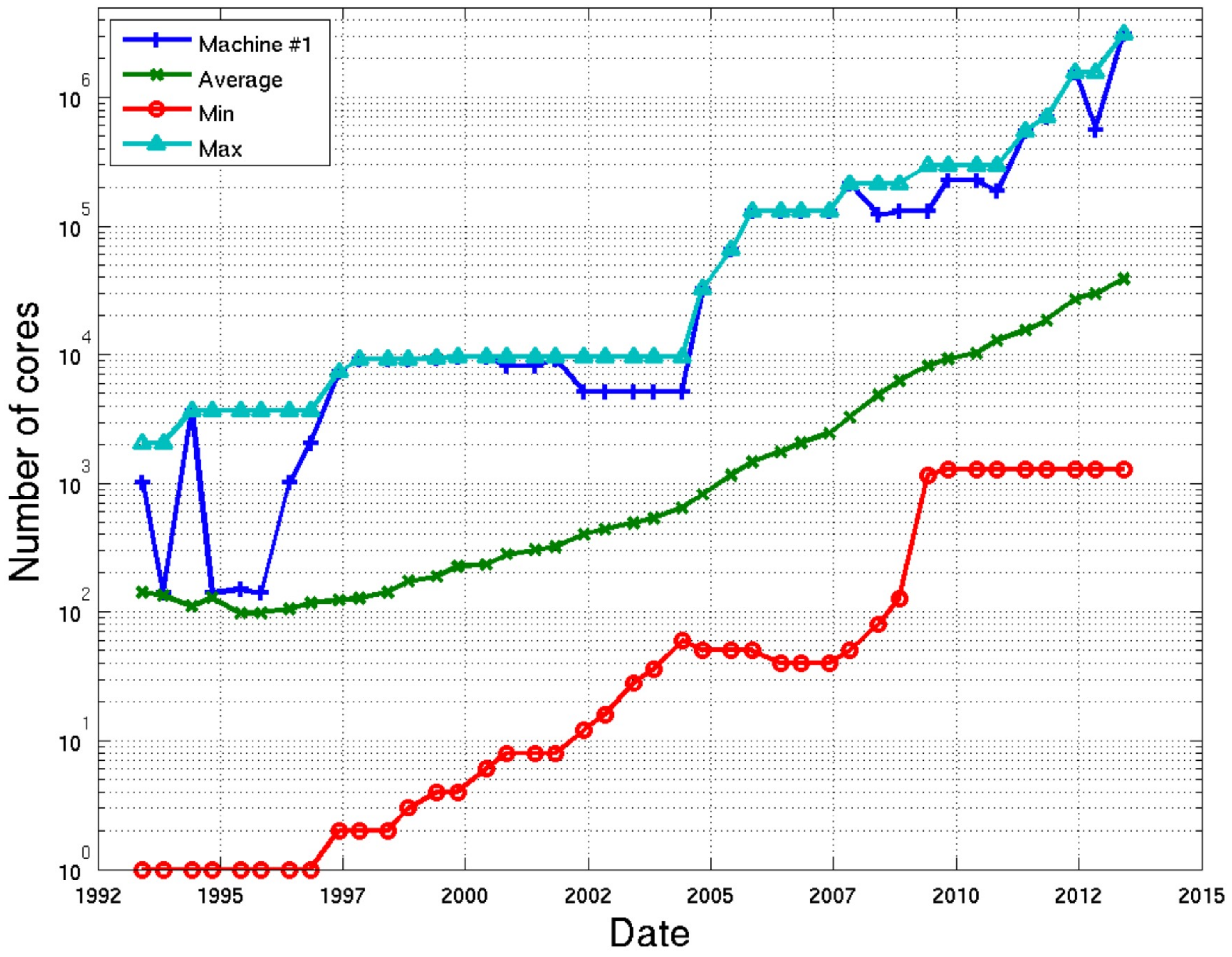
Consequences and solutions:

- The CPU wastes cycles while waiting for data.
- Introduction of cache memory (L1, L2, L3)
- Parallel access to memory (vector architectures, AVX)

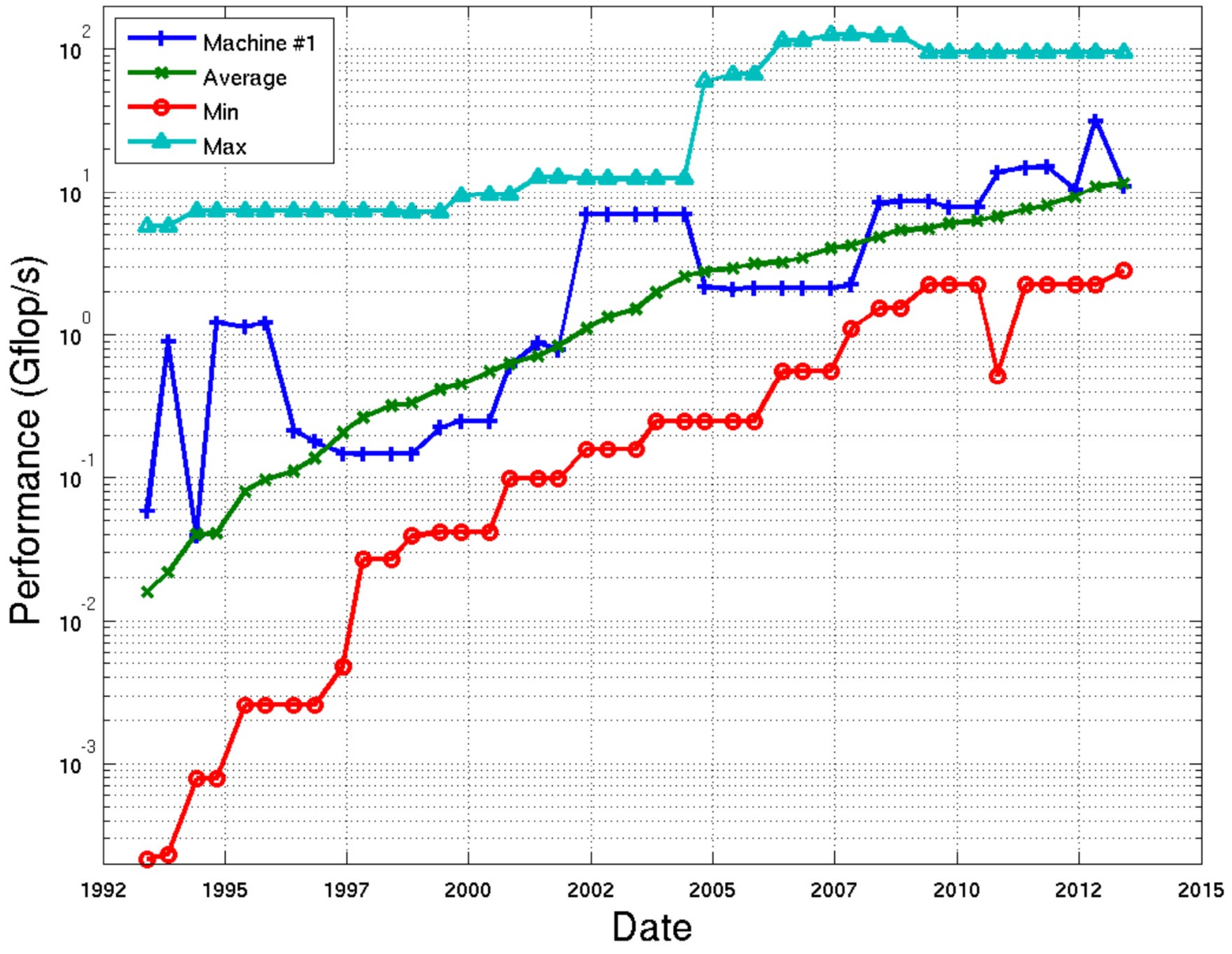
Performance in the Top 500



Number of cores in the Top 500



Performance per core in the Top 500



Parallel Processing

Gene Amdahl (1967): Amdahl's Law

Statement of the law:
The theoretical maximum speedup obtained by parallelizing a code ideally, for a given problem with a fixed size:

$$\text{Speedup}\,(N) = \frac{T_s}{T_p\,(N)} = \frac{1}{\alpha + \frac{1-\alpha}{N}}$$

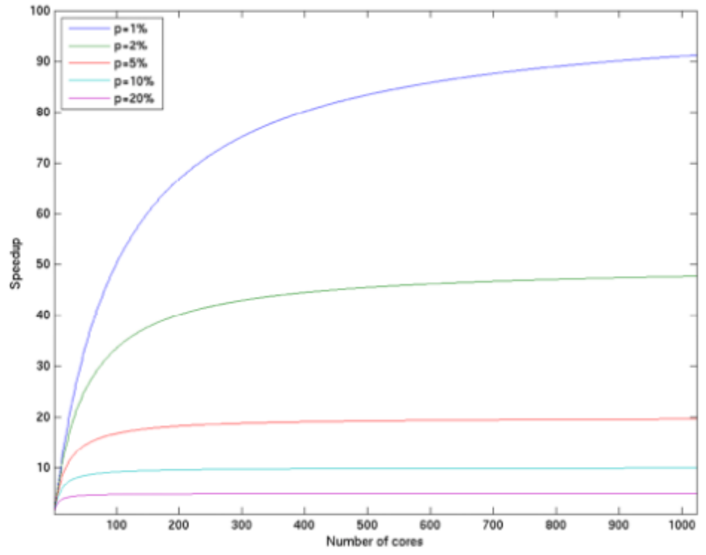
T_s : execution time of the serial code
T_p : execution time of the parallel code
α : fraction of the code that is not parallel
N : number of processors

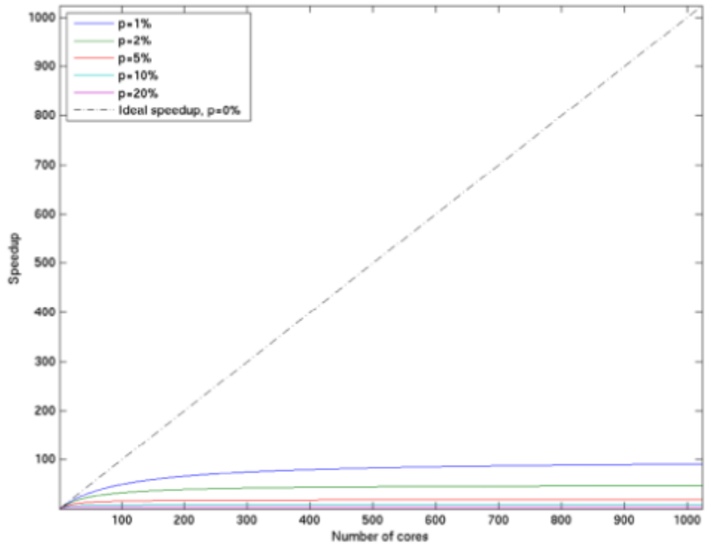
$$\text{Speedup}\,(N) \to \frac{1}{\alpha} \text{ for } N \to +\infty$$

Strong Scaling

Theoretical Maximum Speedup

Cores	α (%)								
	0	0.01	0.1	1	2	5	10	25	50
10	10	9.99	9.91	9.17	8.47	6.90	5.26	3.08	1.82
100	100	99.0	91.0	50.2	33.6	16.8	9.17	3.88	1.98
1000	1000	909	500	91	47.7	19.6	9.91	3.99	1.998
10000	10000	5000	909	99.0	49.8	19.96	9.99	3.99	2
100000	100000	9091	990	99.9	49.9	19.99	10	4	2
∞	∞	10000	1000	100	50	20	10	4	2





Weak Scaling

John Gustafson and Edwin Barsis (1988): Gustafson's Law

Statement of the law:
The theoretical maximum speedup obtained by parallelizing a code ideally *for a problem of constant size per core*:

$$\mathrm{Speedup}\,(N) = \frac{T_s\,(N)}{T_p\,(N)} = \alpha + (1-\alpha)N$$

Assuming that the execution time of the non-parallel part of the code remains constant, one has:

$$T_s\,(N) = \alpha T_0 + N(1-\alpha)T_0$$
$$T_p\,(N) = \alpha T_0 + (1-\alpha)T_0 = T_0$$

This law is more optimistic than Amdahl's law:

$$\mathrm{Speedup}\,(N) \to (1-\alpha)N \ \text{ for } \ N \to +\infty$$

In both laws, everything is determined by α, the non-parallel fraction of the code. Reducing α is called "parallel code optimization".

Hardware Evolution

Technical trends:

- Computing power is doubling every year
- Massively parallel and many-cores architectures are dominant
- Since 2010, Graphical Processing Units become a key player
- Increasing hardware complexity (hybrid system, multiple cache layers)
- Memory per core is plateauing or decreasing
- Performance per core is plateauing
- Disk Input/Output bandwidth is increasing very slowly

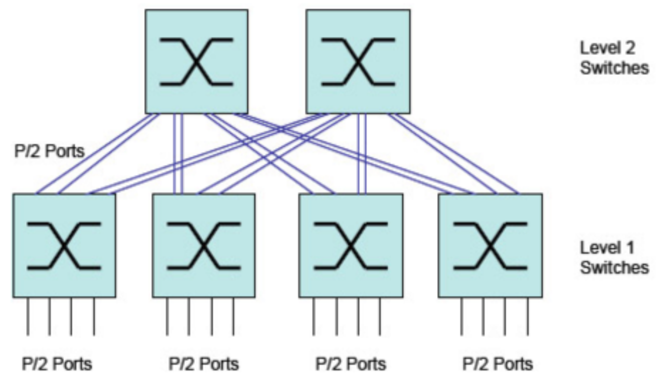
Cray XC50 with GPU

Consequences:

- It is necessary to exploit many (relatively slow) cores
- The memory per core is constant or even decreasing (memory limited)
- Raw performance of individual core not increasing anymore
- Higher level of parallelism and I/O bottlenecks
- More complex architecture (cache levels, accelerators, latency…)
- Multi-disciplinary approach, concept of ”co-design”

Communication Network

Connecting millions of processors requires a high -performance network
Most existing systems use “Infiniband” or variations.
Need a network switch to interconnect all the nodes
Need high quality fiber cables to connect each node to the switch



Fat-tree network: non-blocking network topology invented by Charles Clos (1953)
Depending on how many switches you can afford, you might choose blocking configurations.
http://www.mellanox.com/clusterconfig/
http://clusterdesign.org/fat-trees/

Key network parameters (switches and cables) are the **latency** and the **bandwidth**
Infiniband network: latency 0.5-1 microsec, bandwidth 5-10 GB/sec

Parallel Programming Languages

Evolution of programming methods:

- MPI is still the dominant programming technique
- Hybrid OpenMP/MPI approach most effective on supercomputers
- GPU programming develops quickly
  - CUDA
  - OpenACC and OpenMP
- Message Passing directly within the GPU
- New specific parallel programming languages are developed:
  - Co-array Fortran, PGAS, X10, Chapel…
- New runtime systems to handle task-based parallelism:
  - Charm++, HPX, Kokkos

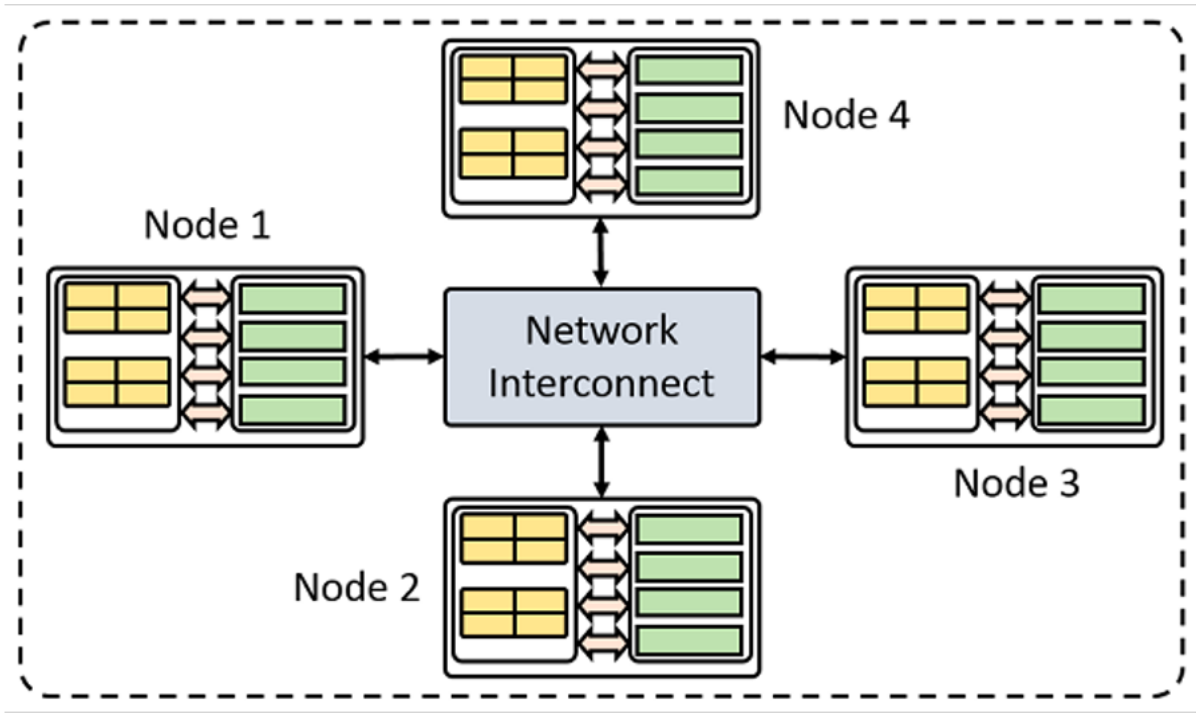
Distributed memory versus shared memory

MPI uses a distributed memory paradigm

- Data are transferred explicitly between nodes through the network

OpenMP uses a shared memory paradigm

- Data are shared implicitly within the node through the Random-Access Memory.



Node 4
Node 1
Network Interconnect
Node 3
Node 2