Lecture # 1 : Introduction

September 12, 2023

What is HPC ?

High-Performance Computing (HPC)

Mixture of

1. Specialized (parallel) hardware
2. Adapted (parallel) algorithms
3. *Ad hoc* (parallel) programming techniques

+ Specially trained programmers !

Course Topics

1. Introduction
2. Distributed programming using MPI
3. Multi-thread programming with OpenMP
4. Vectorization
5. Understanding the hardware and exploiting its full capacity
6. Recurring algorithmic themes/problems

Next year: GPUs/accelerators

Fundamental Role of Numerical Simulations

Traditional scientific method

1. Formulation of a theory based on observations
2. Design of experiments to (in)validate the theory
3. Compare experimental results with observations

Iterate the process until convergence.

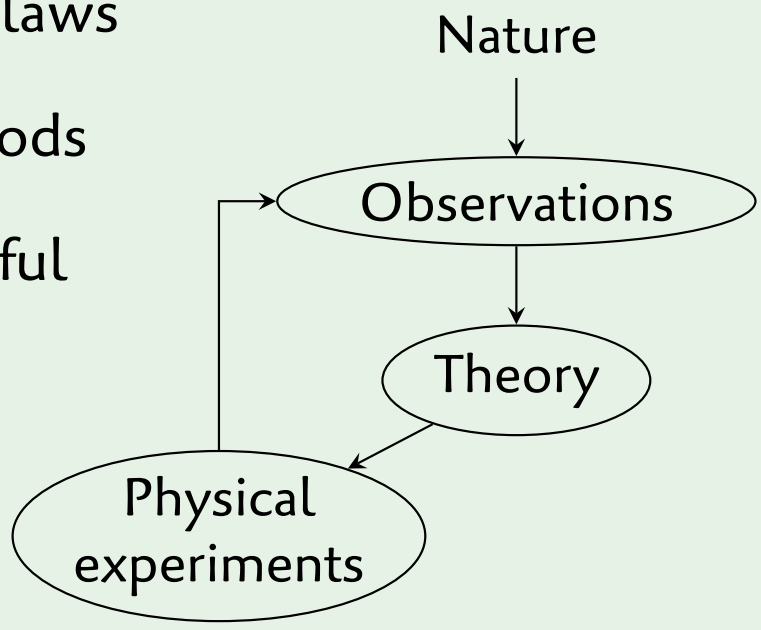
But experimenting can be too...

- Difficult (*Wind Tunnel* containing an Airbus A380?)
- Expensive (*crash test*, ...)
- Slow (climate evolution, galaxy dynamics, ...)
- Dangerous (clinical trials, ☣, ☢, ✺—, ...)

Numerical Simulations

Using computers to simulate and analyze a physical phenomenon

- Starting from physical laws
- Using numerical methods
- with ever more powerful computers

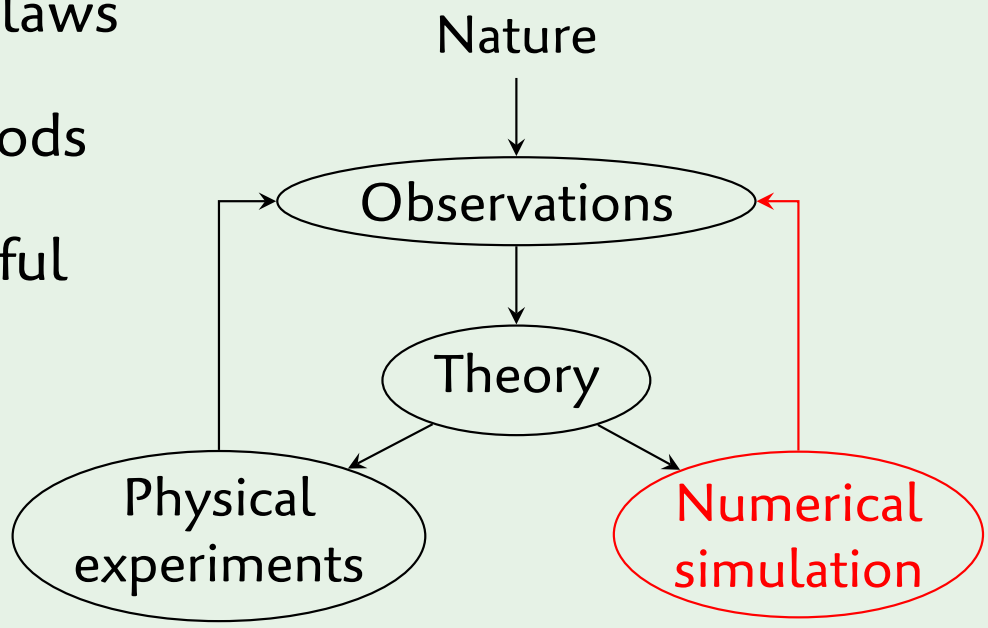


$\rightsquigarrow$ 3rd pillar of science?

Numerical Simulations

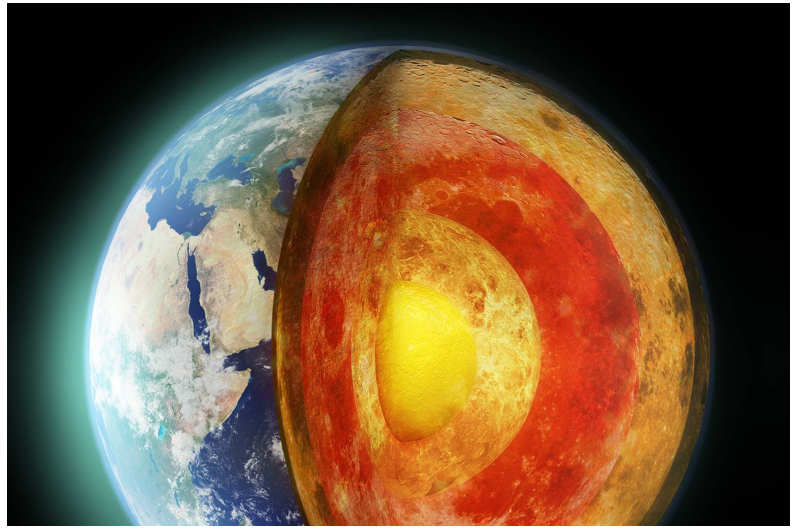
Using computers to simulate and analyze a physical phenomenon

- Starting from physical laws
- Using numerical methods
- with ever more powerful computers



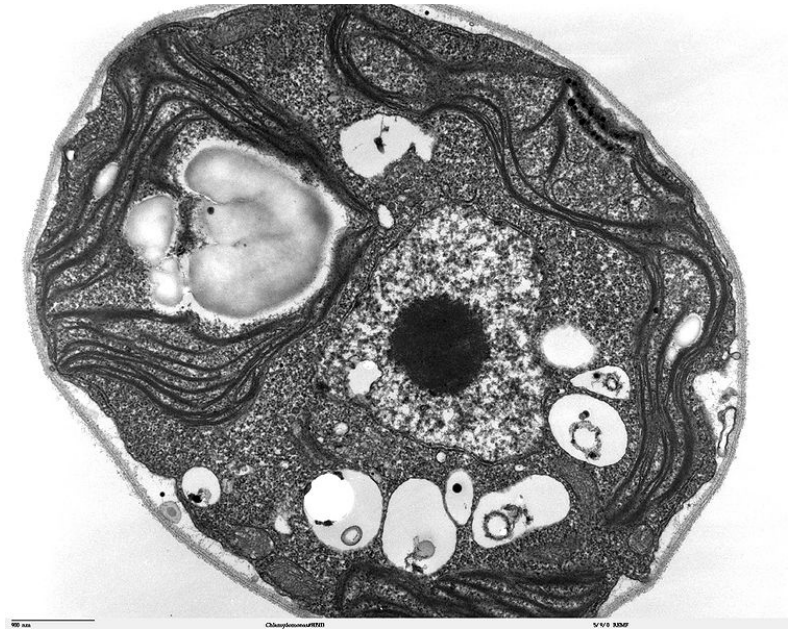
$\rightsquigarrow$ 3rd pillar of science?

Illustration of the Problem



- surface = 510 millions km^2, volume $\approx 10^{12}$ km^3
- One `double` per km^3 $\rightsquigarrow$ 8 TB of memory
- One `double` per m^2 $\rightsquigarrow$ 4 PB of memory

Illustration of the Problem (continued)



- Human cell. About 10^{14} atoms.
- $3 \times$ `double` (x, y, z) per atom $\rightsquigarrow$ 2.4 PB of memory

Not to mention...

- 10^{11}–10^{12} stars in out galaxy
- 10^{11}–10^{12} galaxies...

Realization 

- You know **a little** about how computers works
  - But not completely
  - Especially not **BIG** ones
- You have **not** been trained to :
  - Care about the **performance** of your code
  - Write **efficient** programs

Realization 

- You know **a little** about how computers works
  - But not completely
  - Especially not **BIG** ones
- You have **not** been trained to :
  - Care about the **performance** of your code
  - Write **efficient** programs

- **Time for a change**

Computing Power

performance measurement

Unit : FLOP (*Floating Point OPeration*), FLOP/s (or FLOPS)

- Giga ($10^9 \approx 2^{30}$)
- Tera ($10^{12} \approx 2^{40}$)
- Peta ($10^{15} \approx 2^{50}$)
- Exa ($10^{18} \approx 2^{60}$) ...

- Machines for large computations = **parallel computers**
- Parallelism is an **inexorable trend**

1990's PC

The only one you really know how to program

1 core, 25Mhz, 16KB cache, 4MB RAM

Contemporary Laptop

What you own, fortunately

4 cores, 4Ghz, 8MB cache, 16GB RAM (2 channels)

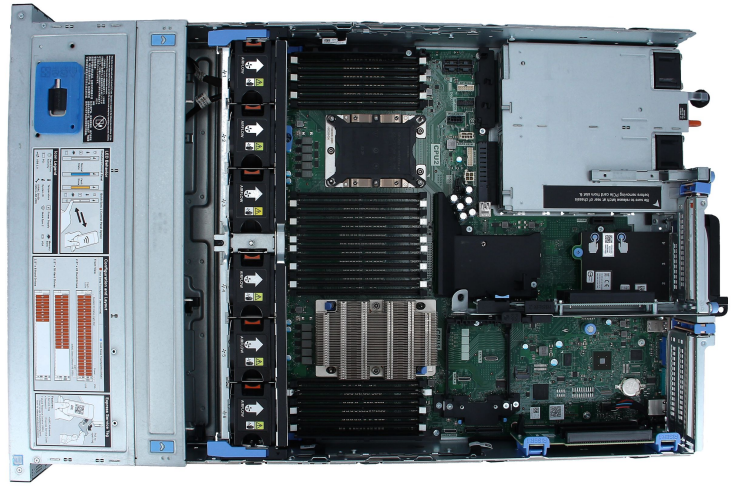
Gamer's PC

For smart-asses who think AI will replace actual science

10–16 cores, 4Ghz, 16MB cache, 32GB RAM (2 channels)

Compute Node

Now we're talking



2 CPUs, 32–128 cores, 2.5Ghz, 256GB RAM ($2\times$ 6–8 channels)

Supercomputer

Rare and Expensive

1000–10000000 cores. Cost between 10^6 and 10^9 euros

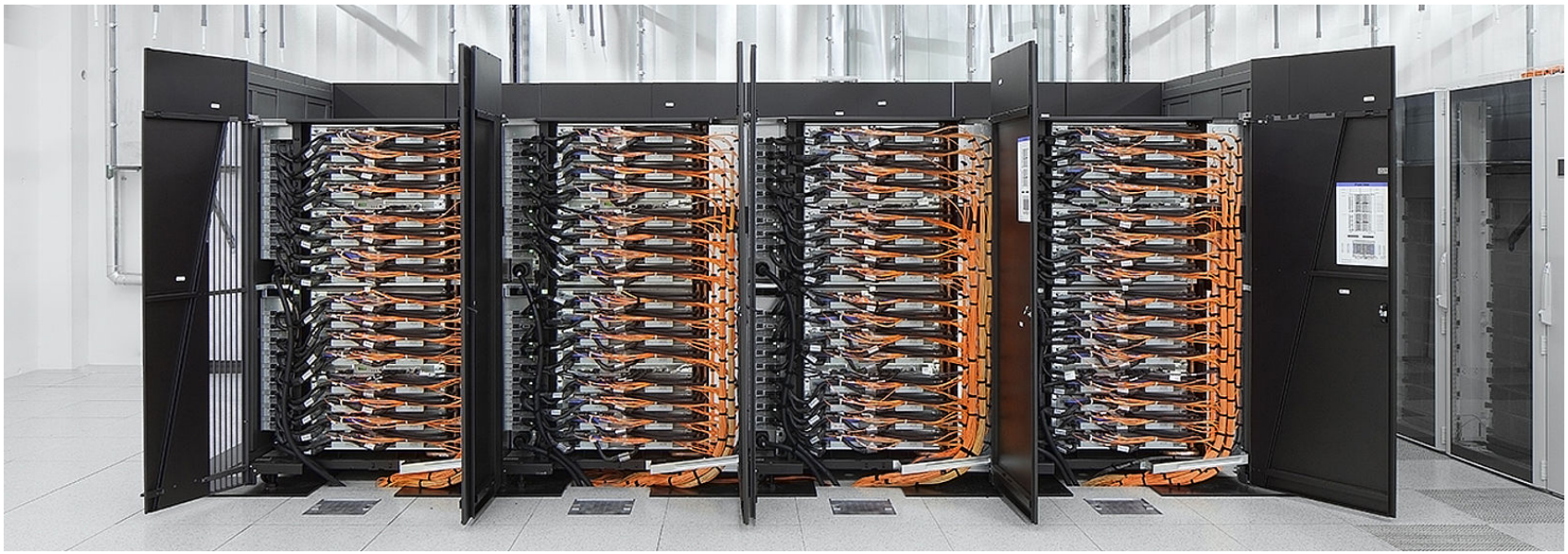
Description of an HPC machine: `Turing`

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

Description of an HPC machine: `Turing`

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

6 racks, 98 304 cores (“small”) $\rightsquigarrow$ `Sequoia` (USA) has 1 572 864



Description of an HPC machine: Turing

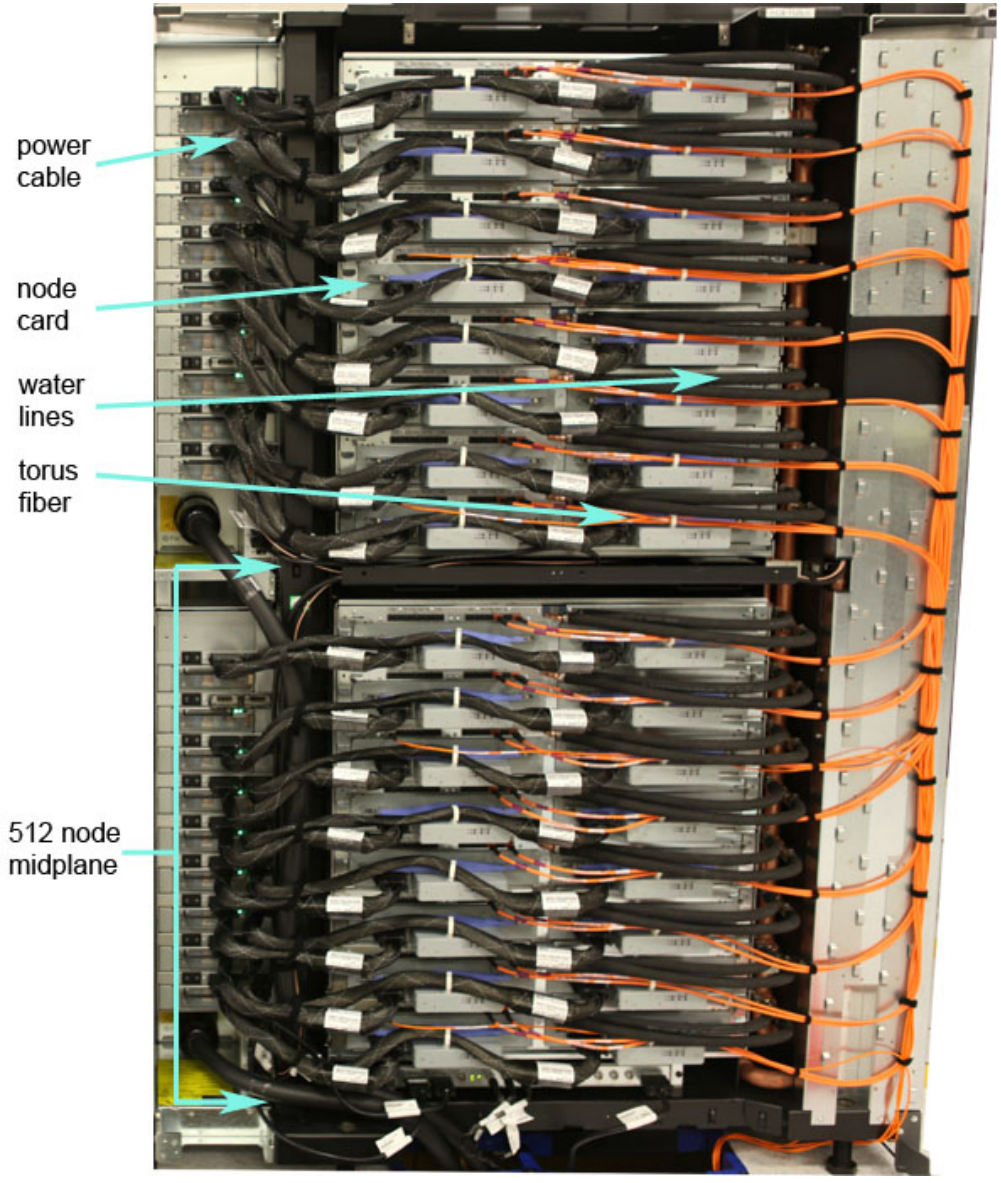
IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

1 rack =

- 16 *node cards*
- + power feeds
- + *water cooling*
- + I/O nodes

Total =

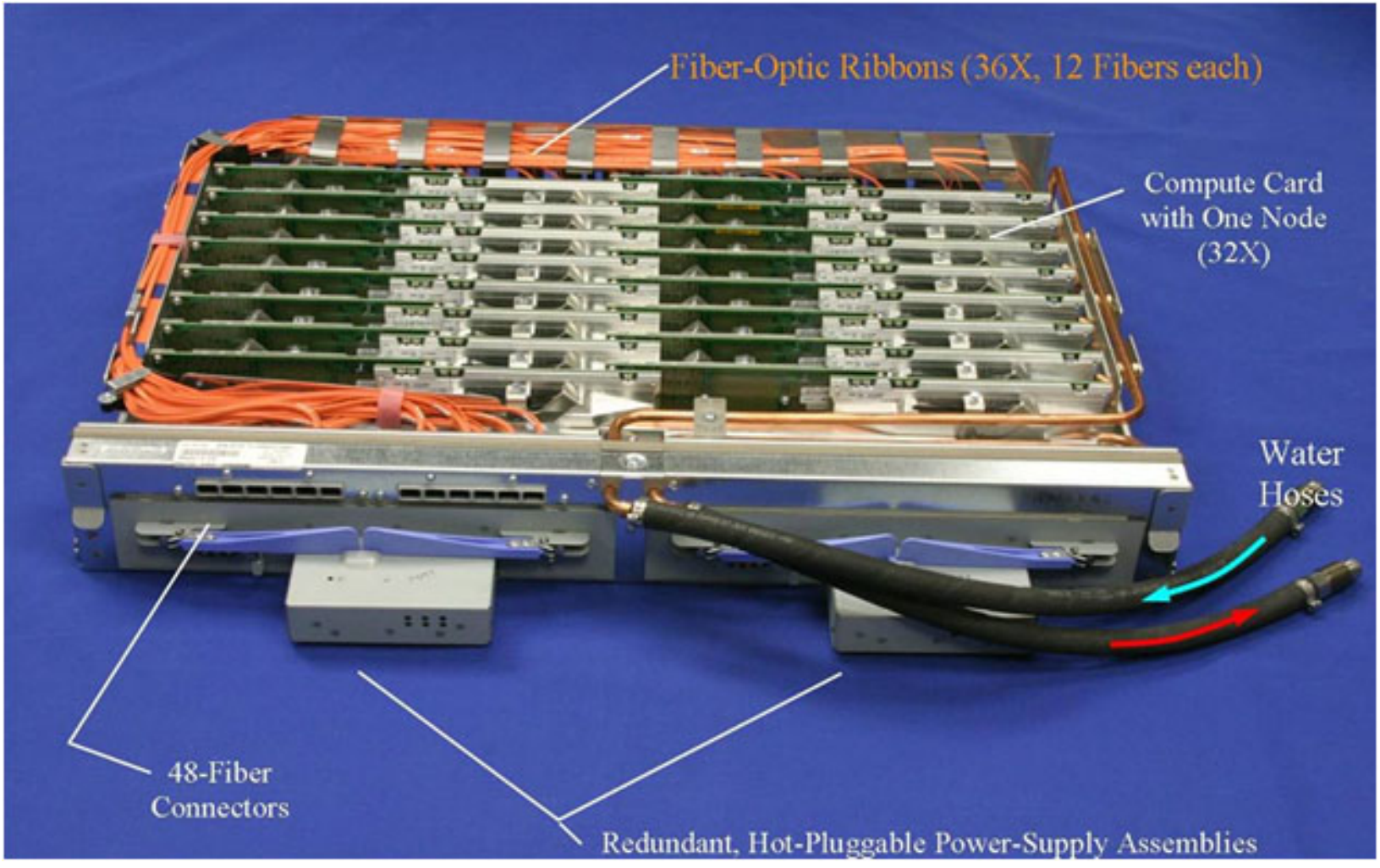
- 1024 nodes
- 16 384 cores
- 16 TB RAM



Description of an HPC machine: Turing

IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

1 *node card* = 32 *compute cards*



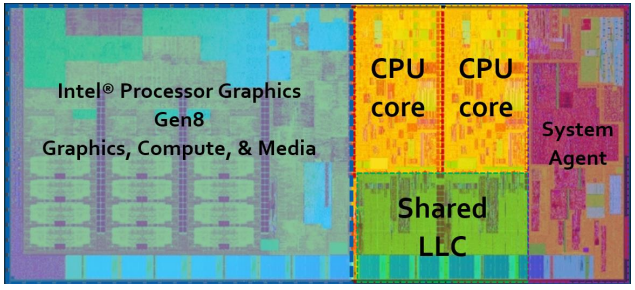
Description of an HPC machine: Turing

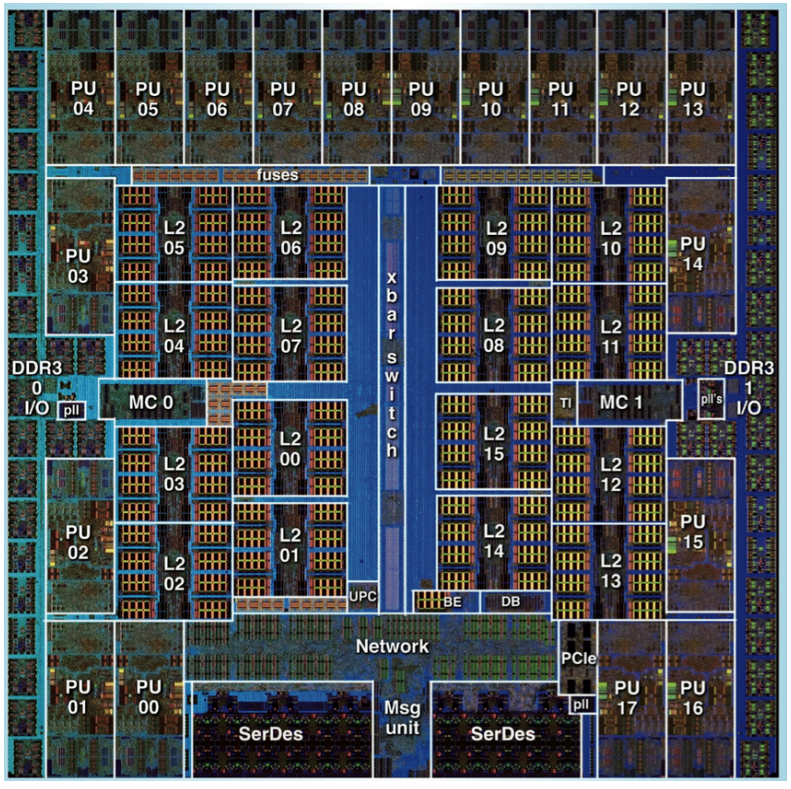
IBM BlueGene/Q (2012–2019) @ IDRIS (CNRS)

1 *compute card* =

- 1 PowerPC A2 CPU (16 cores @ 1.6Ghz, 32MB Cache)
- 16GB of ECC RAM (36 $\times$ 512MB chip)

Not “Normal” computers





A Few Examples (in France)

1. `Turing` (IDRIS / CNRS, 2012–2019), 1.2 PFLOPS
   - IBM BlueGene/Q ($\approx$ 20 millions €)
   - 6144 nodes, 98 304 cores, 98TB RAM
     - 16GB + 1 $\times$ IBM PowerPC A2, 16 cores @ 1.6 Ghz
     - 1 CPU = ??? € = 200 GFLOPS
   - Interconnect = "in-house" 5D-torus, 160 Gbit/s per node
   - 600kW (cold water cooling)

A Few Examples (in France)

1. `Turing` (IDRIS / CNRS, 2012–2019), 1.2 PFLOPS
   - IBM BlueGene/Q ($\approx$ 20 millions €)
   - 6144 nodes, 98 304 cores, 98TB RAM
     - 16GB + 1 $\times$ IBM PowerPC A2, 16 cores @ 1.6 Ghz
     - 1 CPU = ??? € = 200 GFLOPS
   - Interconnect = "in-house" 5D-torus, 160 Gbit/s per node
   - 600kW (cold water cooling)

2. `Jean-Zay` (IDRIS / CNRS, 2019–), 28 PFLOPS
   - HPE SGI 8600 (25 millions €)
   - 1528 "CPU" nodes $\rightarrow$ 61 120 cores, 4.9 PFLOPS
     - 192GB + 2 $\times$ Intel Xeon Gold 6248, 20 cores, 2.5Ghz
     - 1CPU = 3000€ = 1.6 TFLOPS
   - Interconnect = 100Gbit/s OmniPath
   - $\approx$ 1MW (warm water cooling)

A Few Examples (in France)

1. `Turing` (IDRIS / CNRS, 2012–2019), 1.2 PFLOPS
   - IBM BlueGene/Q ($\approx$ 20 millions €)
   - 6144 nodes, 98 304 cores, 98TB RAM
     - 16GB + 1 $\times$ IBM PowerPC A2, 16 cores @ 1.6 Ghz
     - 1 CPU = ??? € = 200 GFLOPS
   - Interconnect = "in-house" 5D-torus, 160 Gbit/s per node
   - 600kW (cold water cooling)

2. `Jean-Zay` (IDRIS / CNRS, 2019–), 28 PFLOPS
   - HPE SGI 8600 (25 millions €)
   - 1528 "CPU" nodes $\rightarrow$ 61 120 cores, 4.9 PFLOPS
     - 192GB + 2 $\times$ Intel Xeon Gold 6248, 20 cores, 2.5Ghz
     - 1CPU = 3000€ = 1.6 TFLOPS
   - 662 "GPU" nodes $\rightarrow$ 199 040 cores, 21.5 PFLOPS
     - idem + 4 $\times$ NVIDIA V100 SMX2 16/32GB, 80 cores
     - 1 GPU = 8500€ = 8 TFLOPS
   - Interconnect = 100Gbit/s OmniPath
   - $\approx$ 1MW (warm water cooling)

More Examples (TOP 500)

1. #5 = `Summit` (DoE, USA, 2018–), 200 PFLOPS
   - IBM ($\approx$ \$ 500 millions)
   - 4 608 nodes
     - $2\times$ IBM Power9, 22 cores, 3Ghz
     - $6\times$ NVIDIA V100 16GB
     - 512GB DDR4 + 96GB HBM + 1.6TB non-volatile memory
     - 1 node $\approx$ 120 000€ = 42 TFLOPS

   $\rightarrow$ 2.8PB RAM, 202 752 CPU cores, 2 211 840 GPU cores
   - Interconnect = 100Gb/s Infiniband
   - 13MW

More Examples (TOP 500)

1. #5 = `Summit` (DoE, USA, 2018–), 200 PFLOPS
   - IBM ($\approx$ \$ 500 millions)
   - 4 608 nodes
     - $2\times$ IBM Power9, 22 cores, 3Ghz
     - $6\times$ NVIDIA V100 16GB
     - 512GB DDR4 + 96GB HBM + 1.6TB non-volatile memory
     - 1 node $\approx$ 120 000€ = 42 TFLOPS

   $\rightarrow$ 2.8PB RAM, 202 752 CPU cores, 2 211 840 GPU cores
   - Interconnect = 100Gb/s Infiniband
   - 13MW

2. #21 = `Frontera` (Univ. Texas, 2019–), 38.7 PFLOPS
   - Dell ($\approx$ \$60 millions)
   - 8008 nodes
     - 192GB + 2 $\times$ Intel Xeon Platinum 8280, 28 cores @ 2.7Ghz
     - 1 CPU = 10 000€ = 2.4 TFLOPS

   $\rightarrow$ 1.5PB RAM, 448 448 cores
   - Interconnect = 100Gbit/s Infiniband

And of Course...

- #2 = Fugaku (RIKEN, Japon, 2021–), 550 PFLOPS
  - Fujitsu ($\approx$ 900 millions €)
  - $\geq$ 150 000 nodes
    - 32GB HBM2 + $1\times$ Fujitsu A64FX (arm), 48 cores @ 2Ghz
    - 1 CPU = ??? € = 2.7 TFLOPS

  $\rightarrow$ 4.8PB RAM, 7 200 000 cores
  - Interconnect = “in-house” 3D torus

And of Course...

- #2 = `Fugaku` (RIKEN, Japon, 2021–), 550 PFLOPS
  - Fujitsu ($\approx$ 900 millions €)
  - $\geq$ 150 000 nodes
    - 32GB HBM2 + $1\times$ Fujitsu A64FX (arm), 48 cores @ 2Ghz
    - 1 CPU = ??? € = 2.7 TFLOPS
  - $\rightarrow$ 4.8PB RAM, 7 200 000 cores
  - Interconnect = "in-house" 3D torus
- #1 = `Frontier` (DOE, USE, 2022–), 1.1 EFLOPS
  - HPE Cray ($\approx$ 600 millions €)
  - $\geq$ 9472 nodes
    - 512GB RAM + AMD Epyc 7A53, 64 cores @ 2Ghz
    - $4\times$ AMD Radeon Instinct MI250X GPUs (128GB)
  - $\rightarrow$ 606,208 CPU cores + 8,335,360 GPU cores + 9PB RAM
  - Interconnect = HPE Slingshot
  - 21MW

Your (Near) Future...

- #9999 = One computer lab room, ≈ 0.8 TFLOPS
  - 16 nodes
    - 4GB RAM + $1\times$ Core i5 CPU (4 cores)
    - 1 CPU = 50 GFLOPS

  $\rightarrow$ 64GB RAM, 64 cores
  - Interconnect = (gigabit ?) ethernet

Great Architectural Diversity

How to get computing power ?

- As-powerful-as-possible CPUs (`frontera`)
- Tons of weak CPUs (IBM BlueGene)
- Hardware accelerators (GPU) (`frontier`, `summit`, `jean-zay`)
- CPU-GPU hybrids (`fugaku`)

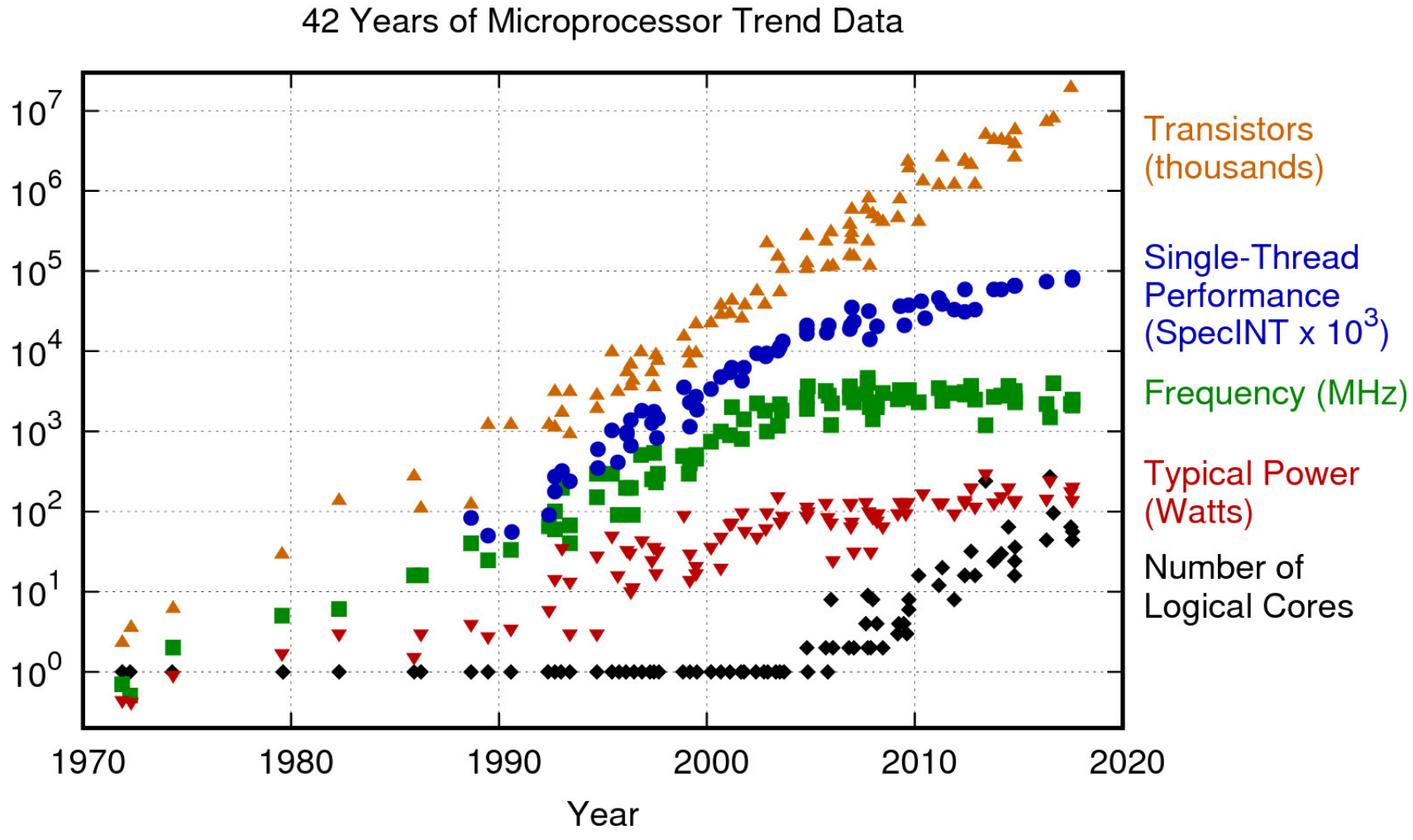
Porting difficulties

Some codes are sometimes written for **ONE** target machine

TOP500 History

Machine	year	nodes	M/node	FLOPS
Frontier	2022	9 472	1T	1100P
Fugaku	2020	158 976	32G	430P
Summit	2018	4 608	608G	150P
Sunway TaihuLight	2016	40 960	32G	93P
Tiahne-2A	2013	16 000	88G	62P
Titan	2012	18 688	38G	18P
Sequoia	2012	98 304	16G	17P
K computer	2011	82 944	16G	11P
Tiahne-1A	2010	7 168	32G	2 566T
Jaguar	2009	18 688	16G	1 941T
Roadrunner	2008	3 240	32G	1 105T
BlueGene/L	2004	106 496	512M	478T
Earth Simulator	2002	640	16G	39T
ASCI White	2000	512	16G	7226T
ASCI Red	1997	4 649	256M	2379G
CP-PACS/2048	1996	2 048	64M	368G
Numerical Wind Tunnel	1993	166	25M	124G
CM-5	1993	1 024	3M	60G

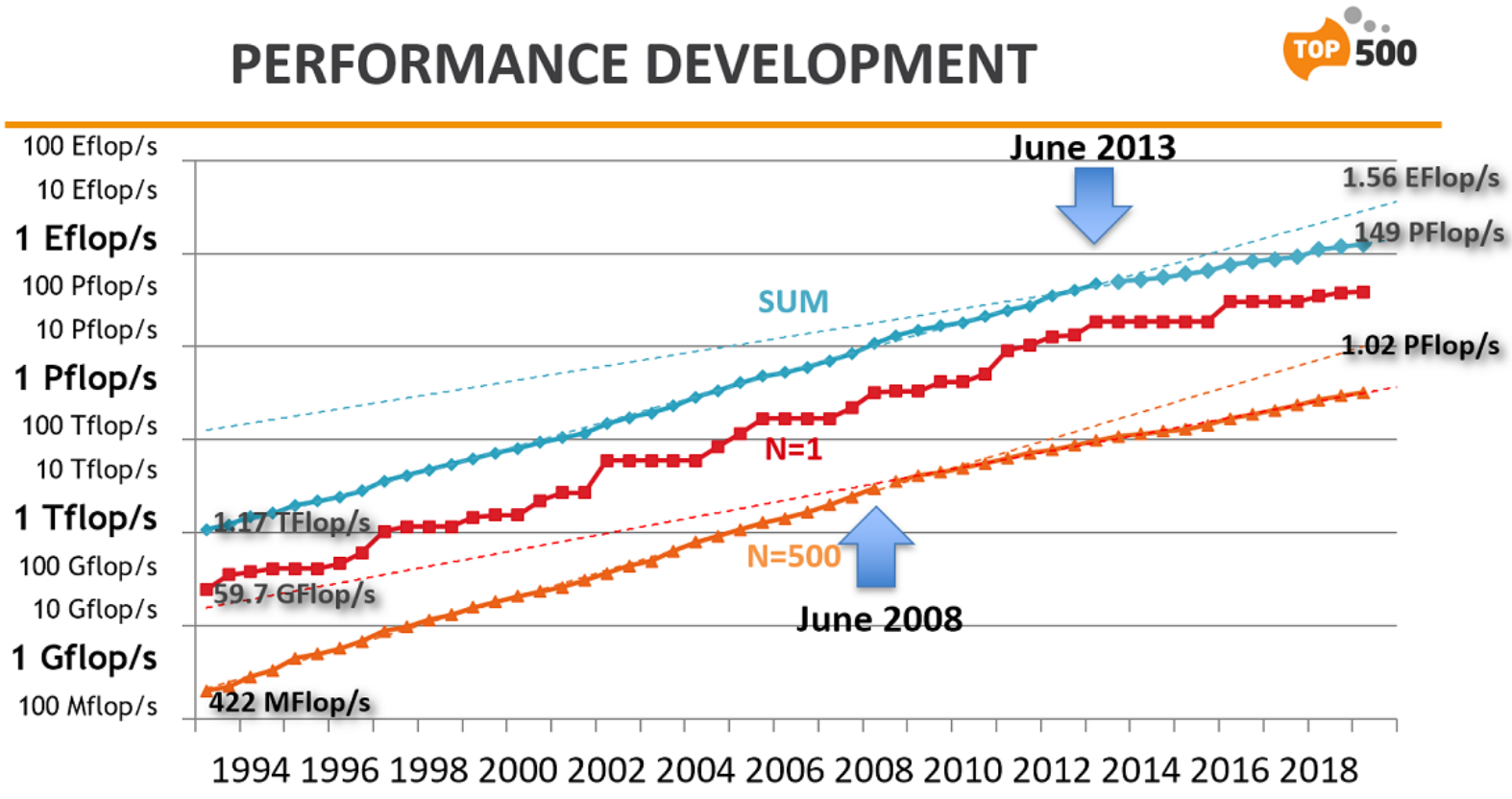
"Moore's Law" and the End of "*Dennard Scaling*"



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

"Moore's Law" and the End of "*Dennard Scaling*"

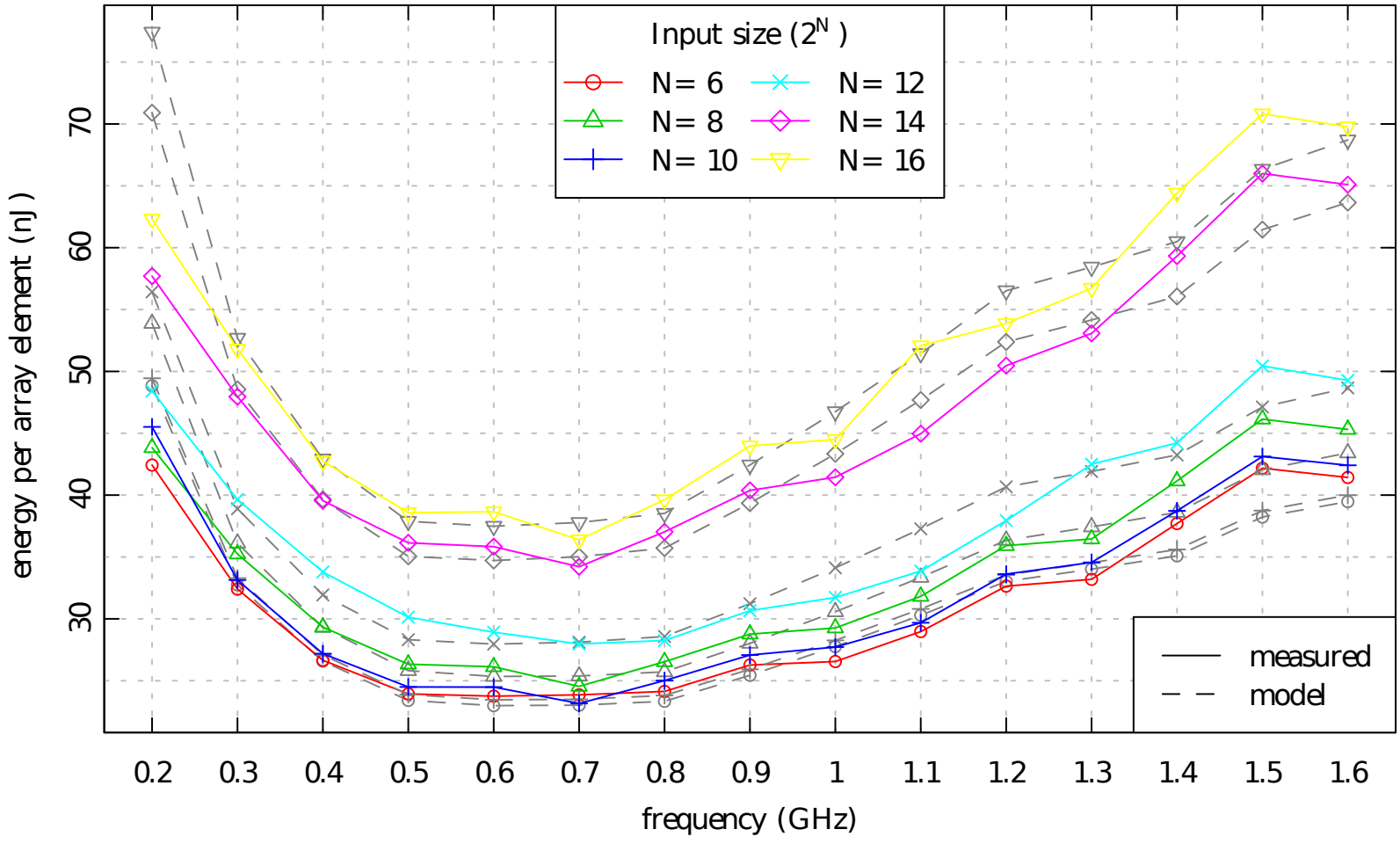
Consequence in the TOP500



Improving the Energy Efficiency (FLOP/Watt)

⇝ Reduce Frequency, increase # processors

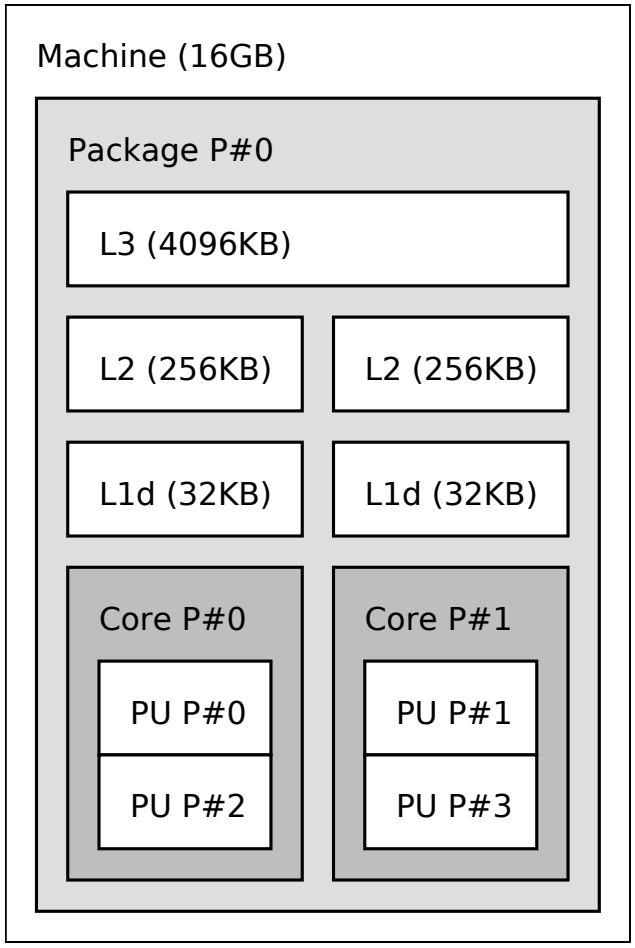
CPU = Cortex A9 (ARM, smartphone). Computation = (part of the) FFT.



Glossary

machine	all components
cluster	compute servers connected to a network
node	an independent “computer”
rack	frame or enclosure for mounting servers, switches...
SMP	node with several processors
processor	contains at least one core + cache, etc.
socket	connects a processor to the motherboard
core	circuit executing code autonomously
multicore	processor containing several cores
hardware thread	autonomous execution context in a core
SMT	core that hosts several hardware threads

Glossary — trouble with "*thread*"



`lstopo` output on a laptop

- hardware thread $\neq$ software thread (OS)
- "physical core" $\approx$ core
- "logical core" $\approx$ hardware thread

Parallel Machines — Classification

Flynn's taxonomy (1966) :

		Data stream	
		single	multiple
Instruction Stream	single	**SISD**	**SIMD**
	multiple	**MISD**	**MIMD**

Parallel Machines — Classification

Flynn's taxonomy (1966) :

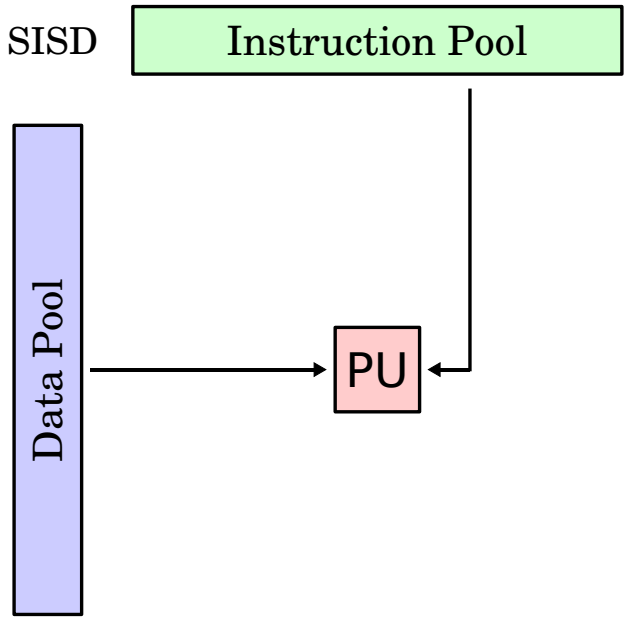
		Data stream	
		single	multiple
Instruction Stream	single	**SISD**	**SIMD**
	multiple	**MISD**	**MIMD**

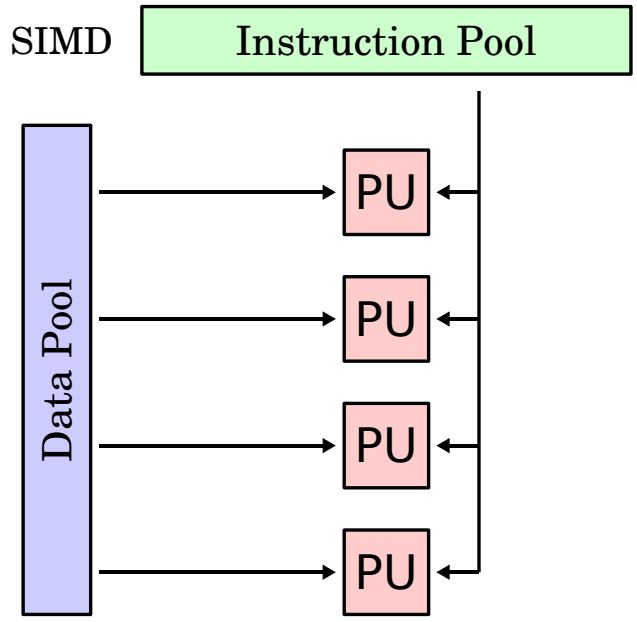
- **SISD** Usual sequential machines
- **MISD** Do not really exist

Flynn's Taxonomy (continued)

- **SIMD**
  Processing units simultaneously execute the same operation on their own data
  - Synchronous operations
  - Single centralized control unit
  - Examples :
    - Vector machines from the 1970–90 (CRAY , NEC ...)
    - Vector instructions in contemporary CPUs (SSE, AVX2, AVX-512, NEON, ...)
    - Graphics Processing Units (GPUs)

SIMD



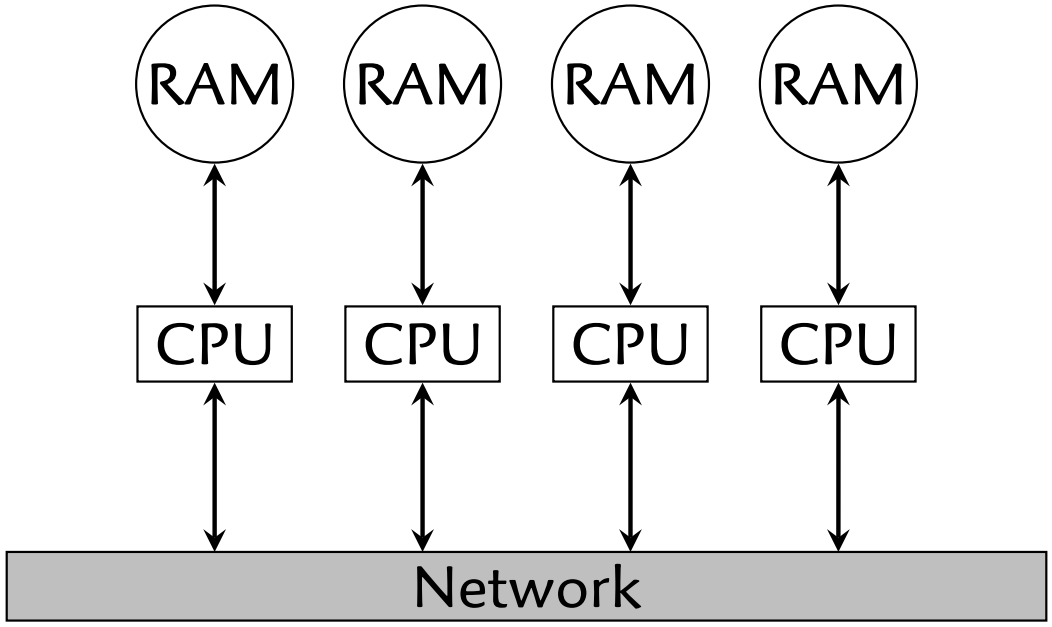


(image: Wikipedia)

Flynn's Taxonomy (continued)

- **SIMD**
  Processing units simultaneously execute the same operation on their own data
  - Synchronous operations
  - Single centralized control unit
  - Examples :
    - Vector machines from the 1970–90 (CRAY , NEC ...)
    - Vector instructions in contemporary CPUs (SSE, AVX2, AVX-512, NEON, ...)
    - Graphics Processing Units (GPUs)
- **MIMD**
  Independent processors can perform arbitrary operations on their own data simultaneously.
  - Asynchronous
  - Examples :
    - Multicore CPU
    - Clusters
    - All big HPC machines

Distributed Memory



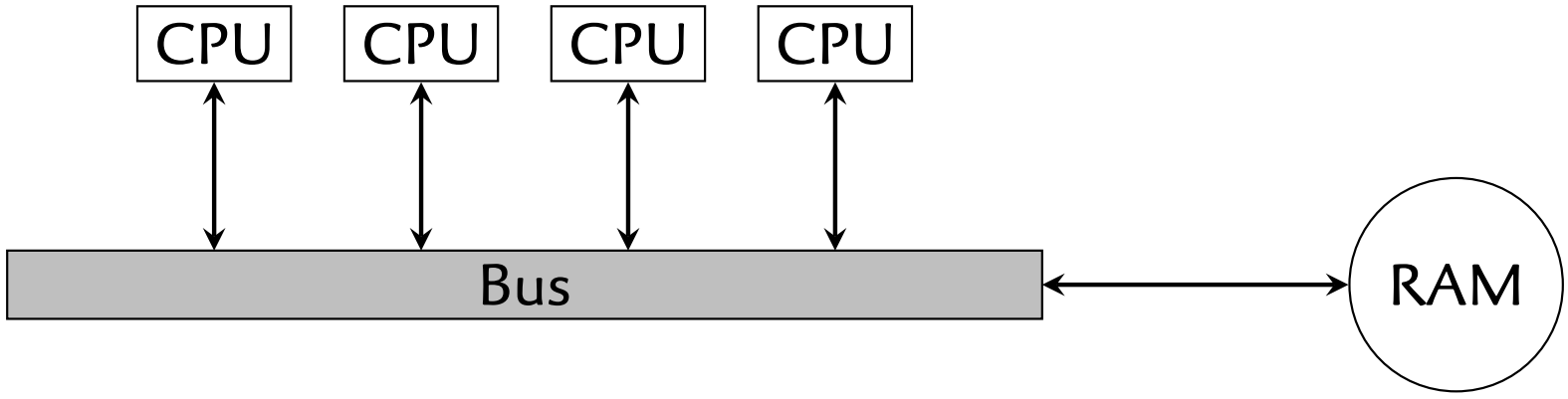
- Each processor has its own memory
- Communication = messages exchange on a network
- Network performance = **critical** aspect of the machine

Bad Idea

Tree-like Ethernet, Ordinary Switches

Bad performance, scalability problems

Shared Memory



- CPUs connected to the memory through a bus
- Each CPU may access the whole memory
- Communications $\leftrightarrow$ memory accesses

- Beware of race conditions!

Non-Uniform Memory Access

Shared memory: scalability problem

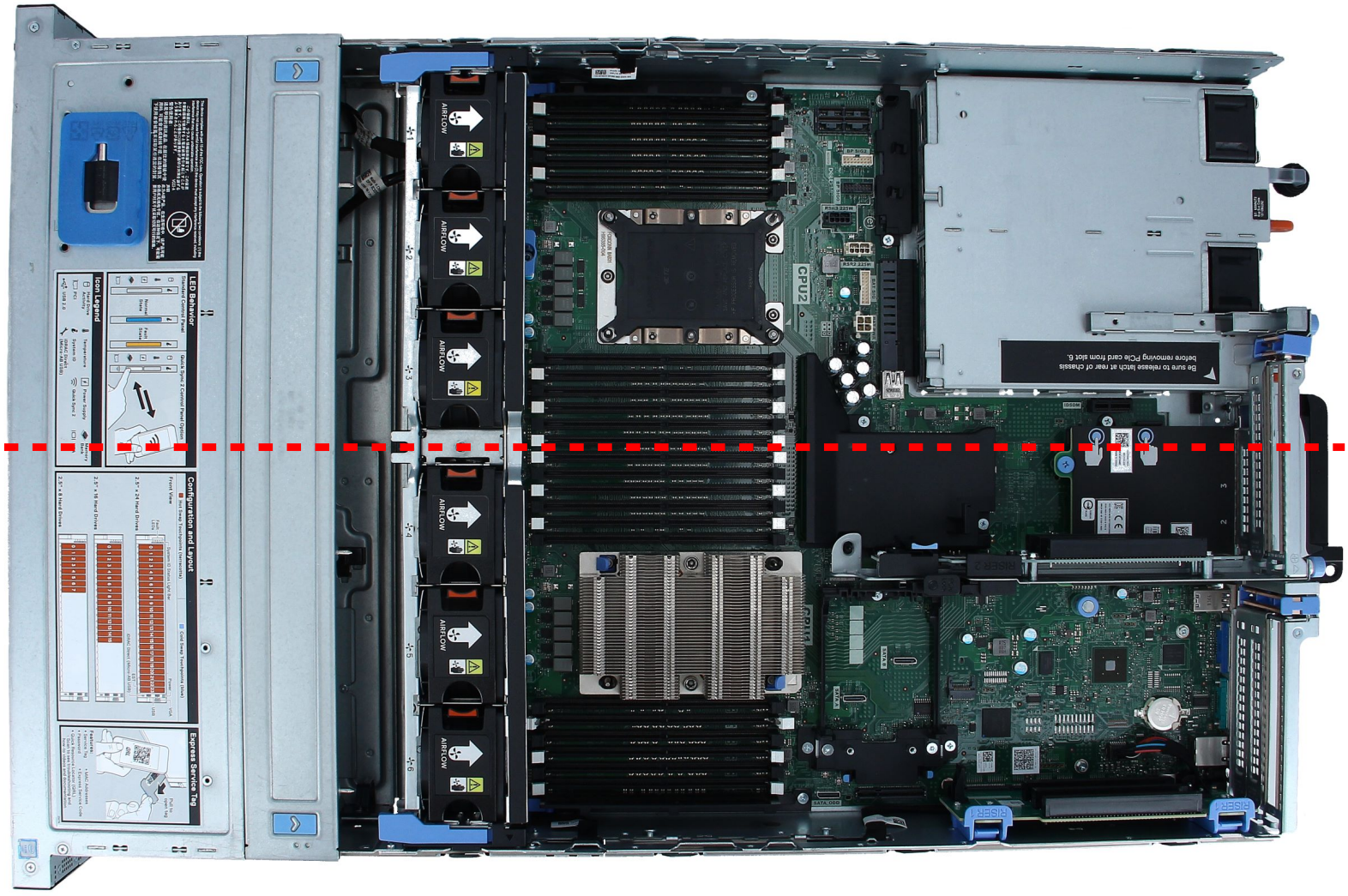
More CPUS $\rightsquigarrow$ contention on the bus

Solution

- Simulate a shared memory with a distributed memory
- Each CPU "owns" and control a share of the memory
  - *Directly* accesses its own share
- Accesses the rest by communicating with other CPUs

$\Rightarrow$ *ad hoc* network, very fast.

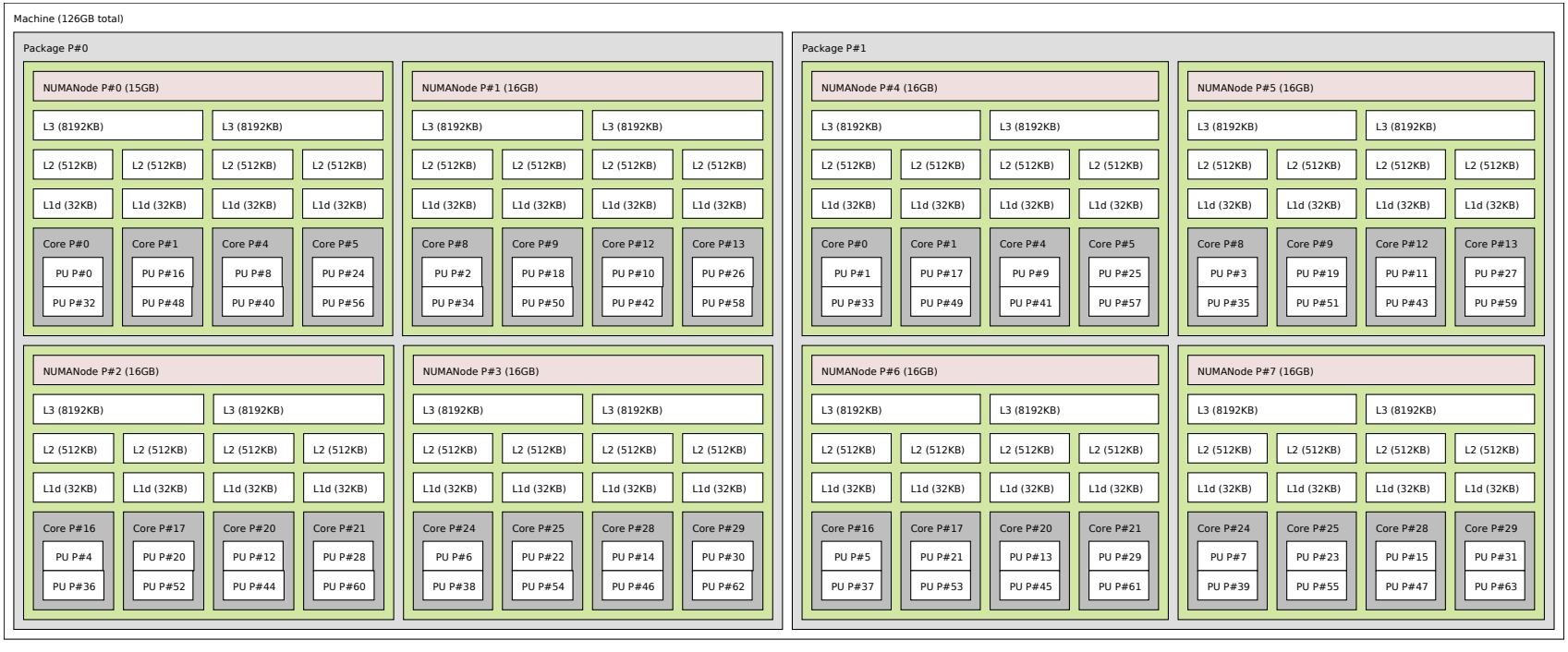
Non-Uniform Memory Access



Non-Uniform Memory Access

Real Example

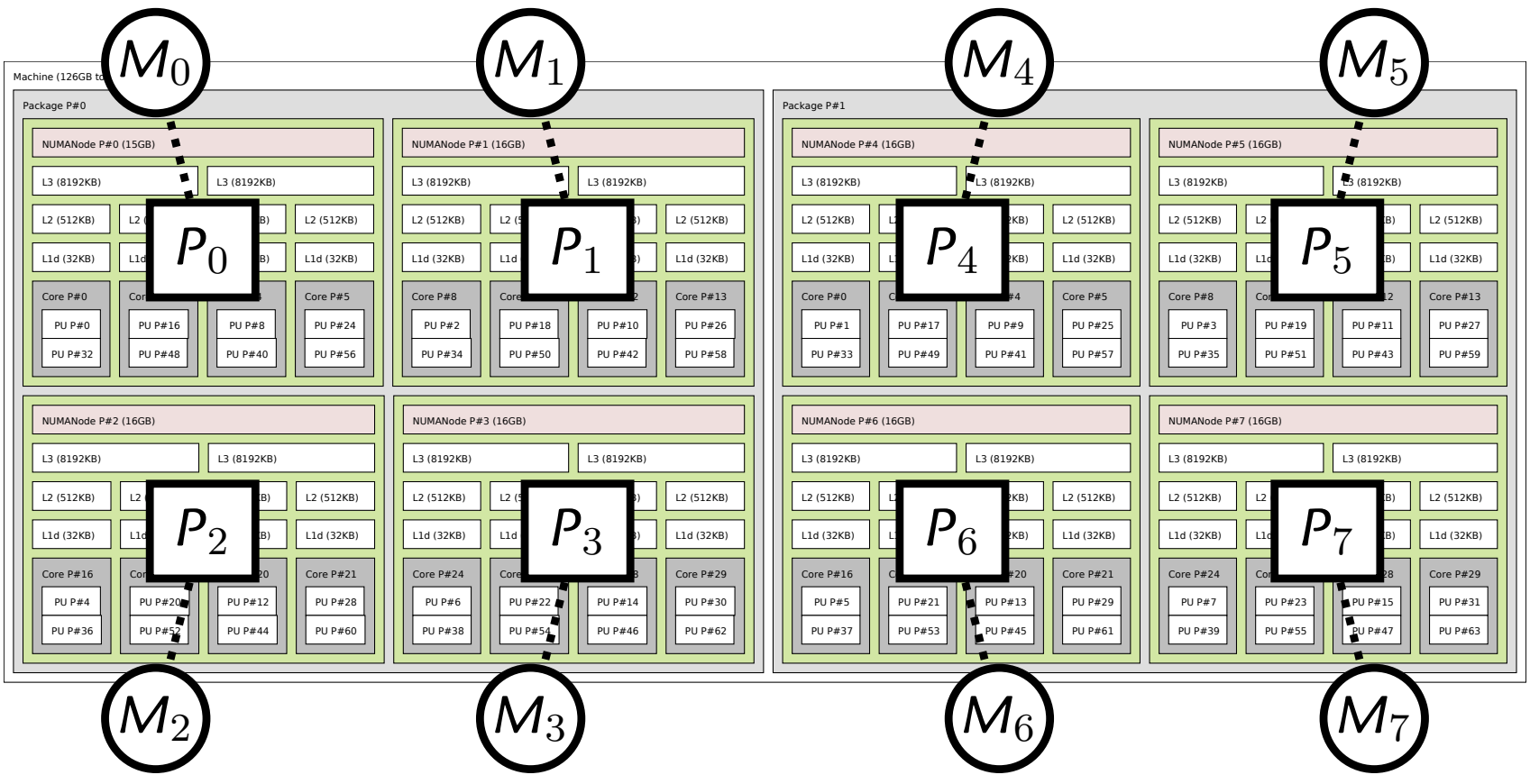
- SMP node with $2\times$ AMD EYPC "Naples" CPUs
- $4\times$ independent memory controller per CPU



Non-Uniform Memory Access

Real Example

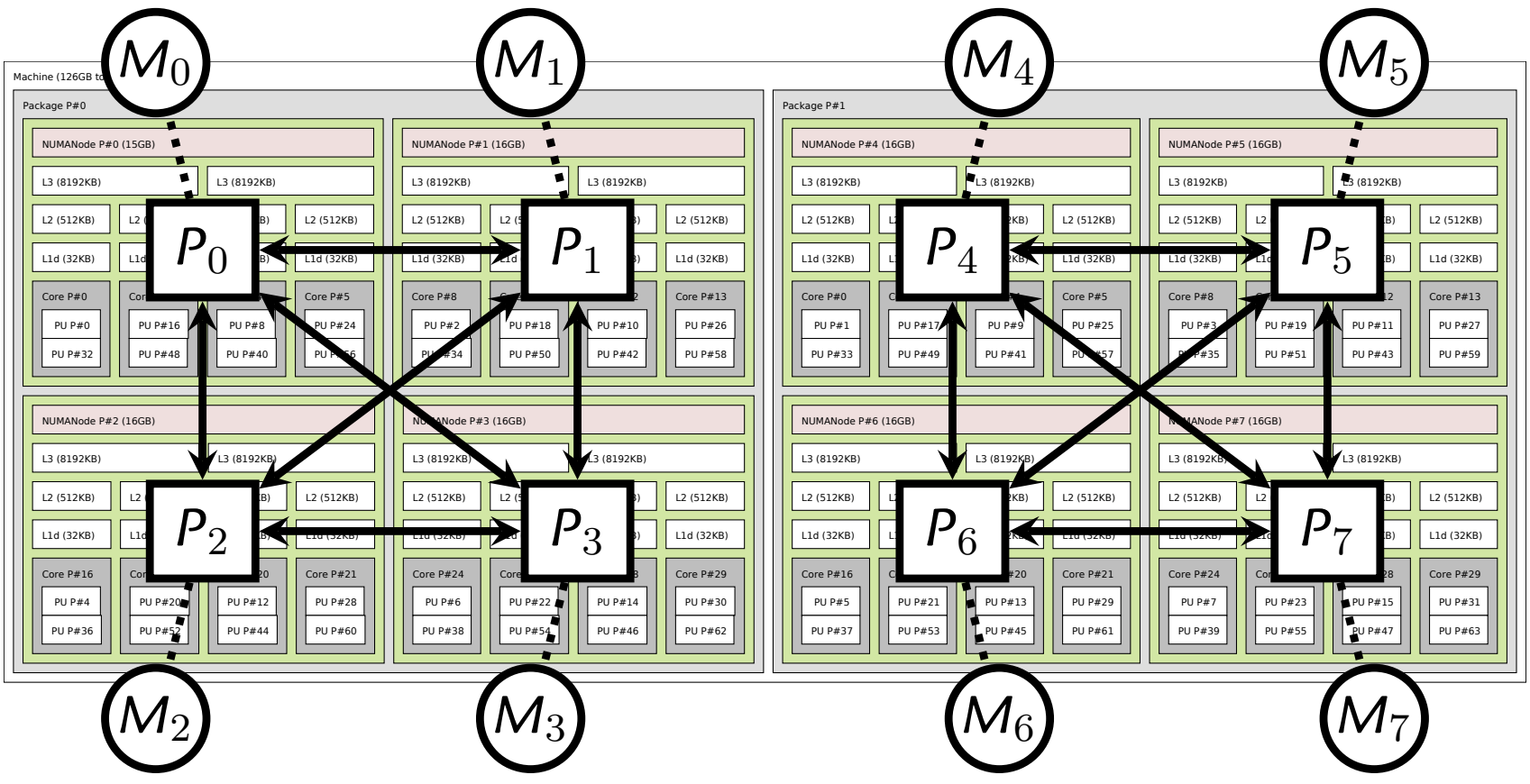
- SMP node with $2\times$ AMD EYPC "Naples" CPUs
- $4\times$ independent memory controller per CPU



Non-Uniform Memory Access

Real Example

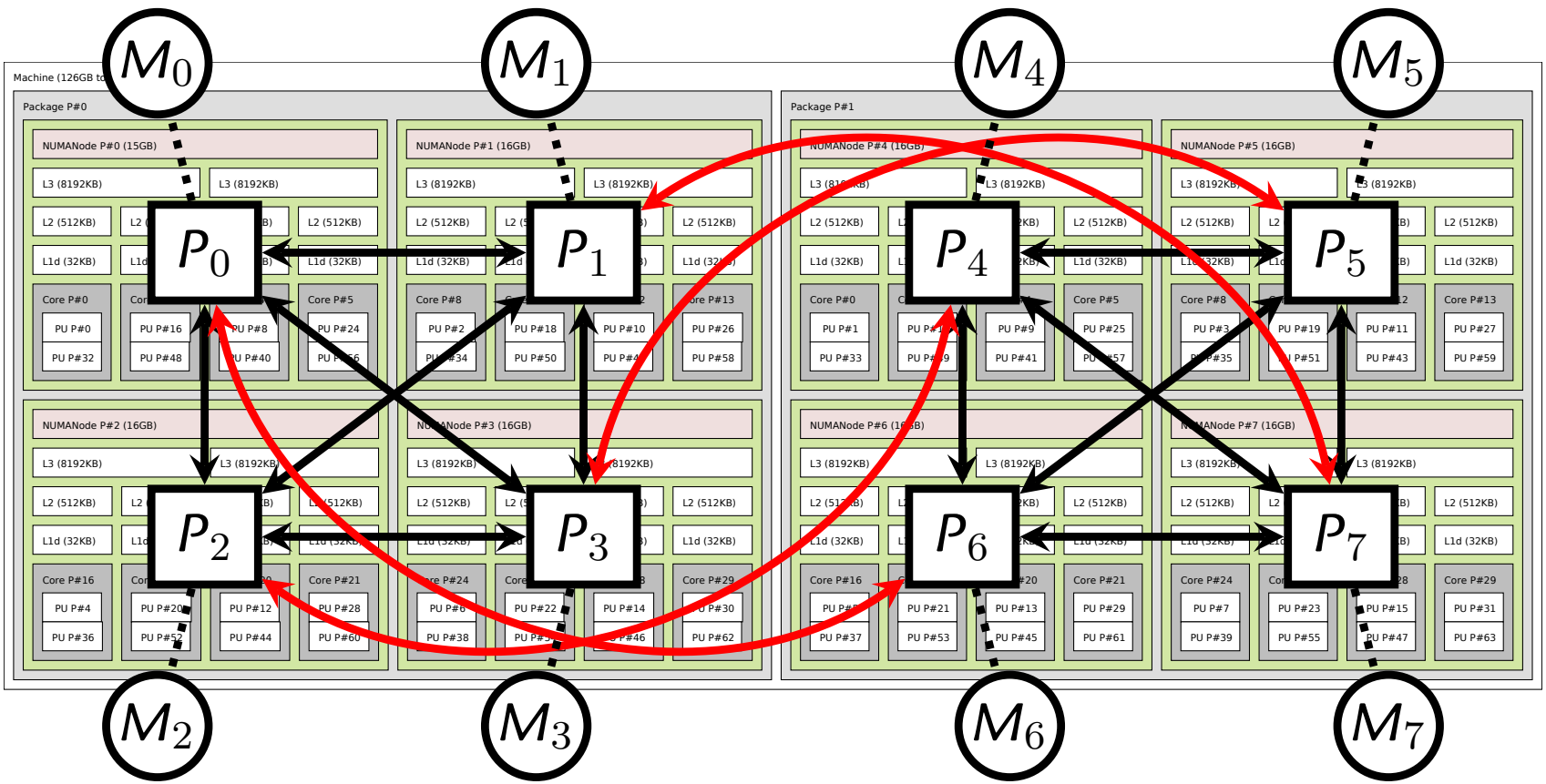
- SMP node with $2\times$ AMD EYPC "Naples" CPUs
- $4\times$ independent memory controller per CPU
- Complete interconnection inside of a CPU



Non-Uniform Memory Access

Real Example

- SMP node with $2\times$ AMD EYPC "Naples" CPUs
- $4\times$ independent memory controller per CPU
- Complete interconnection inside of a CPU
- Partial interconnection between the two sockets
- Red link = $\approx 2\times$ higher latency



In practice...

- machine = nodes connected to a network
  - MIMD / distributed memory
- One node = several processors (SMP)
  - Most likely NUMA
- Inside a node = all cores access the memory
  - Shared memory
- Inside a core = vector instructions
  - SIMD

Everything, everywhere, all the time.

Instruction-Level Parallelism (ILP)

```
slwi 10,9,3
add 8,11,10
lwzx 10,11,10
lwz 7,4(8)
or. 10,10,7
bne 0,.L146
addi 5,5,8
stw 3,0(8)
stw 4,4(8)
cmplw 7,6,5
bne 7,.L24
lwz 9,144(19)
li 10,1
stw 10,20704(31)
addi 9,9,1
```

code = *ordered* sequence of instructions

Parallelism inside a core

- *Pipeline(s)*
- *Superscalarity*
- *Out-of-order execution*
- *Simultaneous Multi-Threading*

$\Rightarrow$ Parallel implementation

$\Rightarrow$ Sequential semantics

How to Write Parallel Programs?

- Automatic parallelization

How to Write Parallel Programs?

- Automatic parallelization
- "Annotations" and compiler hints in C programs (OpenMP, OpenACC)

OpenMP

```
#pragma omp parallel for
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        for (int k = 0; k < n; k++)
            C[i][j] += A[i][k] * B[k][j];
```

How to Write Parallel Programs?

- Automatic parallelization
- "Annotations" and compiler hints in C programs (OpenMP, OpenACC)
- plain C + libraries (`pthread`, MPI)

Librairies

```
 for(int t=0; t<NUM_THREADS; t++)
       pthread_create(&threads[t], NULL, ThreadFunction, (void *) t);
...
pthread_exit();
```

How to Write Parallel Programs?

- Automatic parallelization
- "Annotations" and compiler hints in C programs (OpenMP, OpenACC)
- plain C + libraries (`pthread`, MPI)
- New languages?

Go

```
func main() {
    s := []int{7, 2, 8, -9, 4, 0}
    c := make(chan int)
    go sum(s[:len(s)/2], c)
    go sum(s[len(s)/2:], c)
    x, y := <-c, <-c
    fmt.Println(x, y, x+y)
}
```

Recycling Sequential Programming Languages

Single Program Multiple Data (SPMD)

- Same sequential code executed many times in parallel
- Special variable: `rank`
- `rank` $= i$ in the i-th copy of the program

```
int i = rank;                          /* data parallelism */
for (int j = 0; j < n; j++)
    for (int k = 0; k < n; k++)
        C[i][j] += A[i][k] * B[k][j];

if (rank == 0) {                       /* control parallelism */
  <<Purely sequential section>>
} else {
  <<Wait>>
}
```

Performance Evaluation

Justice of the peace : wall clock

- $T_1(n)$: time needed by the **best** sequential algorithm to process an instance of size n
- $T_p(n)$: time needed by the parallel algorithm to process an instance of size n on p processors

Speedup

$$S(n,p) = \frac{T_1(n)}{T_p(n)}$$

Efficiency

$$E(n,p) = \frac{S(n,p)}{p}$$

Strong scaling

Goal when Parallelizing?

- **Minimize time-to-solution for a fixed problem**

$\Rightarrow$ Study of the performance with fixed n and increasing p

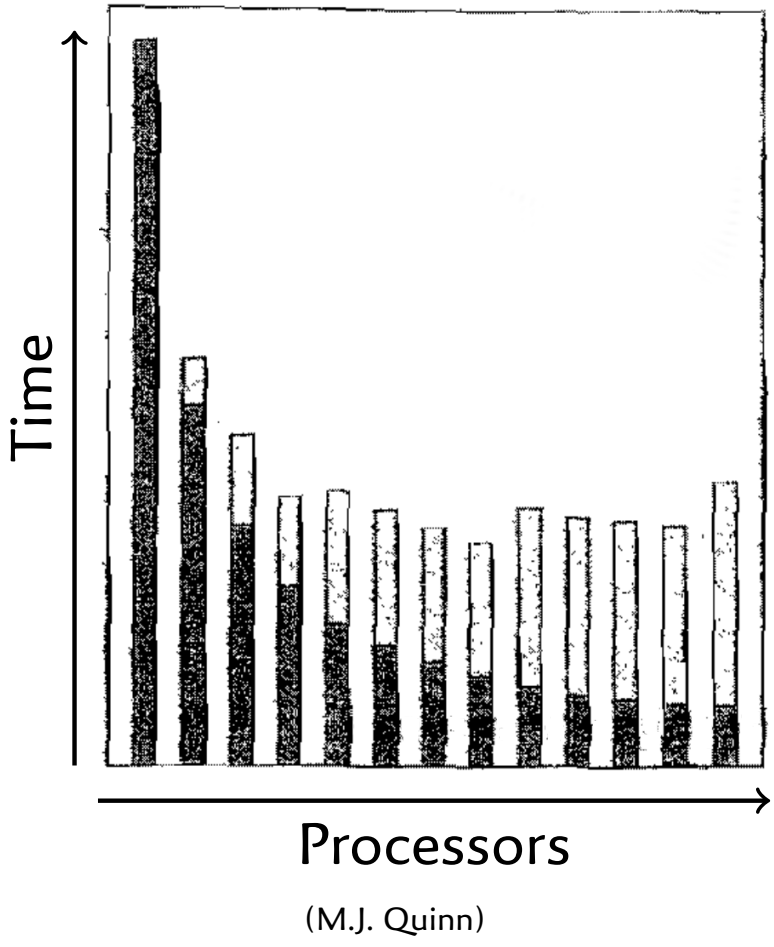
- **Linear speedup** ($\rightarrow$ **ideal**)**:** processors are 100% busy, no extra operations

$$S(n,p) = p \qquad E(n,p) = 1$$

- **Sublinear speedup:** processors are less than 100% busy, or extra operations
- **Superinear speedup:** hard to envision

Strong scaling

Real example

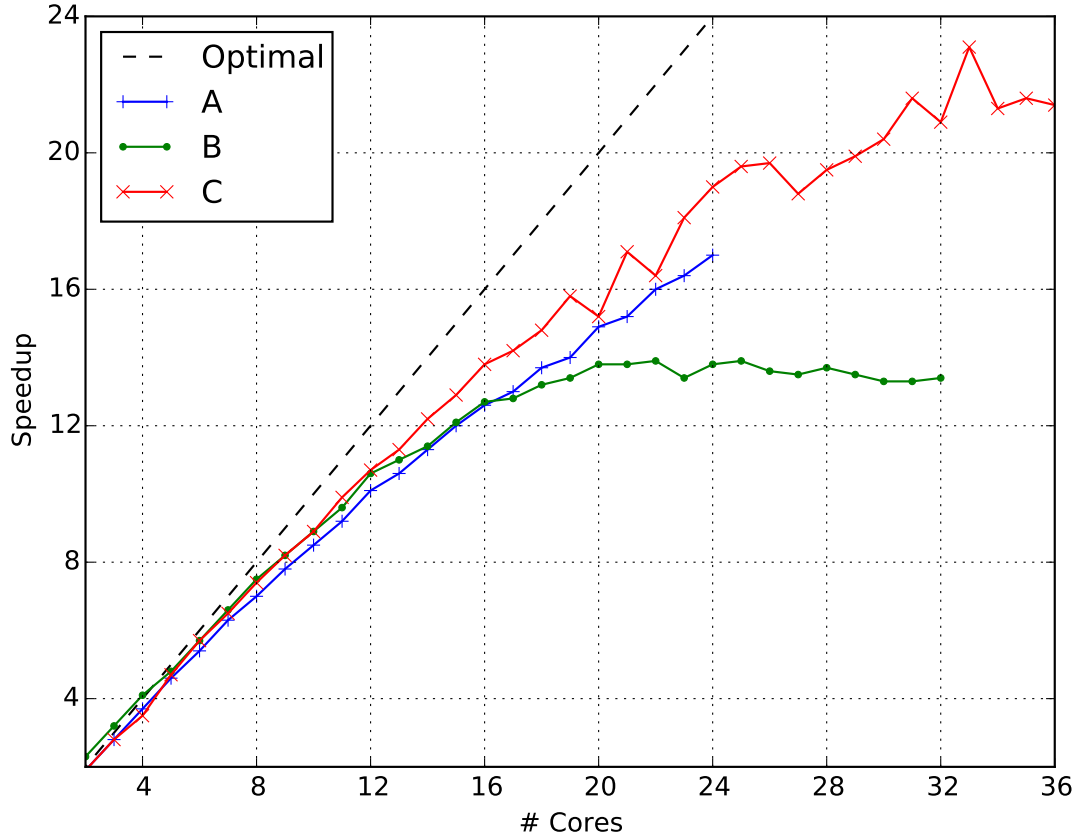


(M.J. Quinn)

- Gray: computation time
  - decreases (or stagnate) when p ↗
- White: overhead
  - Increases when p ↗
- For a given problem size (n), there is an optimal number of processors that can be efficiently exploited
  - Beyond this point, adding extra processors no longer brings any gain

Real Example (my code)

Compute Rank of Sparse Integer Matrices



Name	CPU Type	# CPU	cores/CPU	L3 Cache/CPU
A	Xeon E5-2670 v3	$2\times$	12	30MB
B	Xeon E5-4620	$4\times$	8	16MB
C	Xeon E5-2695 v4	$2\times$	18	45MB

Amdahl's Law

Sequential Portions Limit Parallel Speedup

Algorithm with a fraction f of operations that must be performed sequentially. Then

$$S(n,p) \leq \frac{1}{f + (1-f)/p}$$

$\Rightarrow$ 20% of an algorithm is sequential $\rightsquigarrow$ Speedup ≤ 5

Overhead caused by parallelism $(1 - E(n,p))$ caused by sequential parts and

- Communications
- Starting / Synchronizing tasks
- Load imbalance
- ...

Weak scaling

Goal when Parallelizing?

- **Solve *bigger* problems with more processors**

$\Rightarrow$ Study of the performance when both p and n increase

Amdahl's effect (empirical)

- When $n \nearrow$, computation time often increases *more* that communication time
- Often allows for greater speedups

Gustafson-Barsis's Law

A parallel computation is run on p processors; time spent in sequential operations is measured and represents a fraction s of the total.

Then :

$$S(n,p) \leq p - (p-1)s$$

Example

A parallel computation runs in 100s on 32 processors, including 5s in serial portions. Then $S(n,p) \leq 30.45$.