Algorithmics&Parallel programming *// programming*

Françoise Baude

Université Côte d’Azur

Polytech Nice Sophia

baude@unice.fr

web site: https://lms.univ-cotedazur.fr/course/view.php?id=269001

Feb 2023

Chapter 1 : General Introduction, PRAM models

Organization

- 9h C in total
- Approx. 20h TD or labs in total

- One individual project in C/OpenMP
- One Lab on GPGPU programming
- Perhaps One lab in C/MPI
- Final individual exam (50%), around mid April 2023
  - Personal notes on an A4 sheet of paper

- Bibliography on this (old) topic is huge: check slides links, LMS, and many more !

Plan

1. **Motivation**

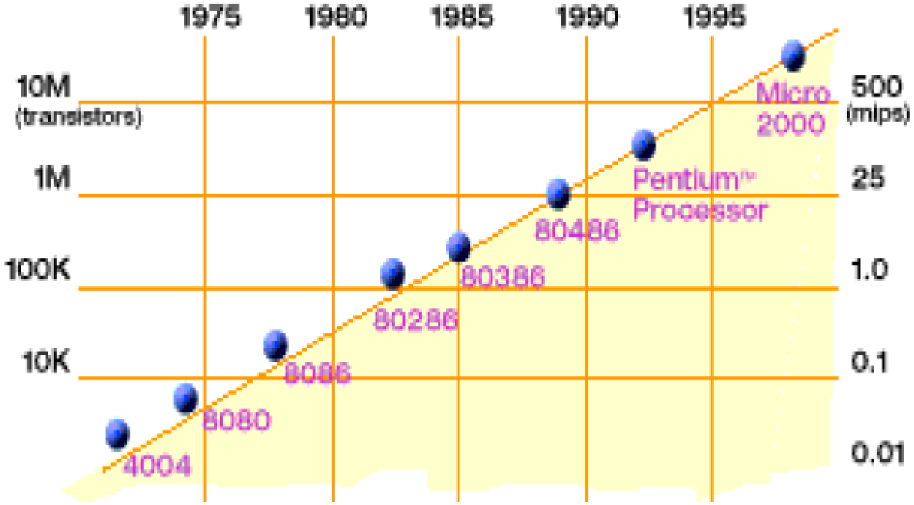
Which includes A very brief glimpse of parallel machines

2. Introduction to the theoretical PRAM model and typical algorithms

Why parallel computing ?

- Solve a problem faster
  - Rely on machines that are more powerful
    - Increase chip performances (Moore law)
    - Or Number of processing elements >> 1
- Solve problems whose size is too high to be handled by one single machine
  - Cut the problem into sub-problems
    - Solve sub-problems in parallel on several machines

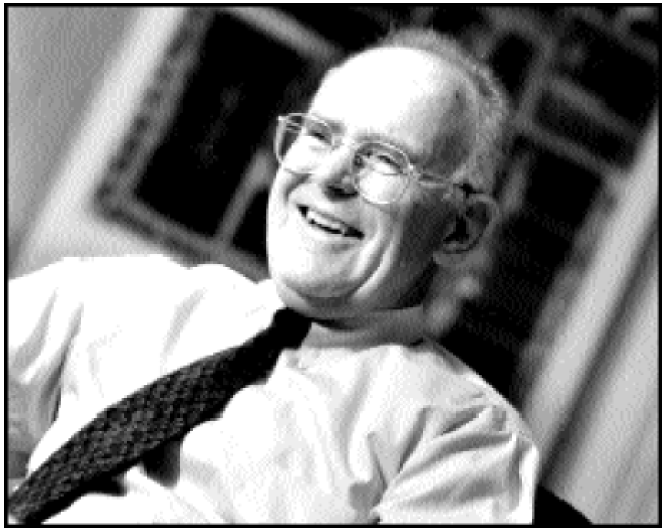
Chip performance: Moore Law …



2X transistors/Chip Every 1.5 years

Called "Moore's Law"

Microprocessors have become smaller, denser, and more powerful.



Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

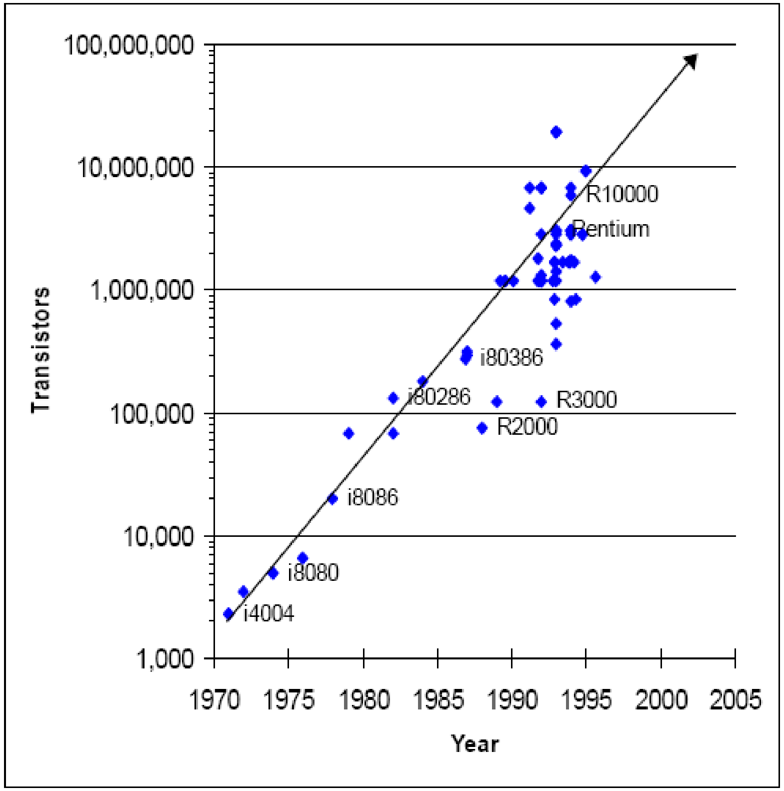
Source, cours CS194 U.Berkeley

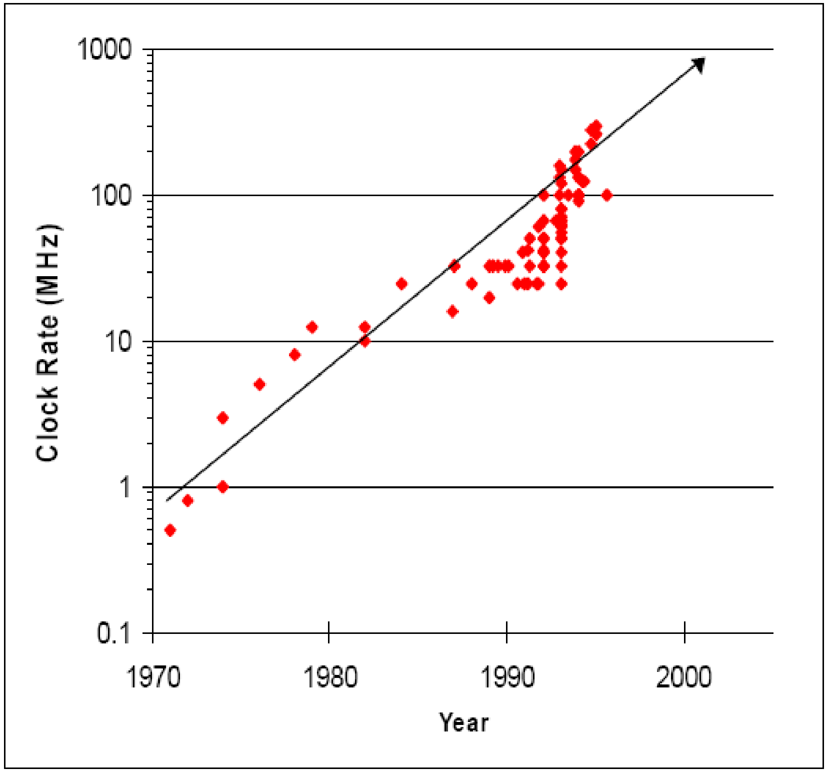
To compute faster :

Moore law (around the 2000's)

Growth in transistors per chip

Increase in clock rate

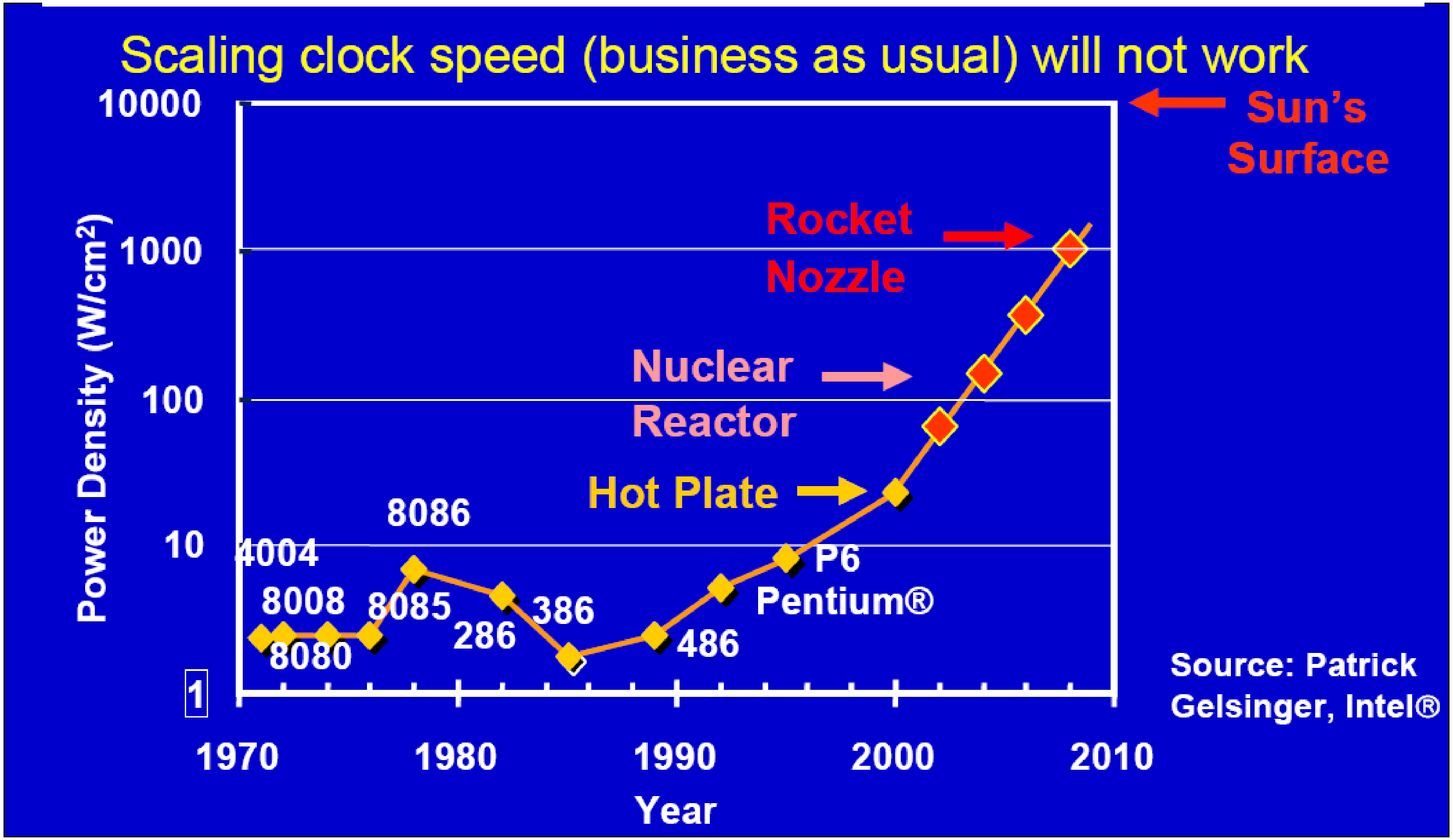




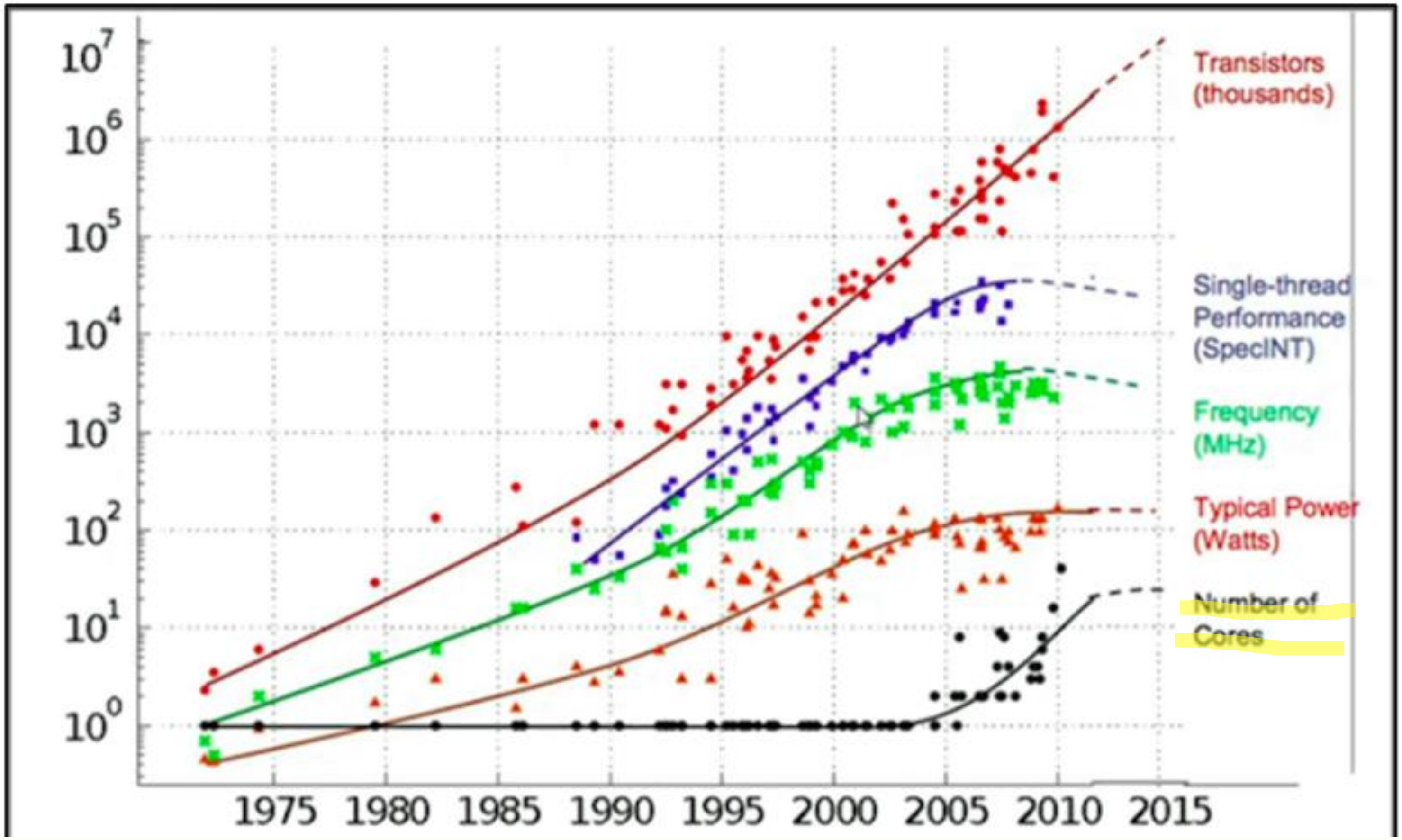
Why bother with parallel programming? Just wait a year or two…

Source, cours CS194 U.Berkeley

Moore’s law limit

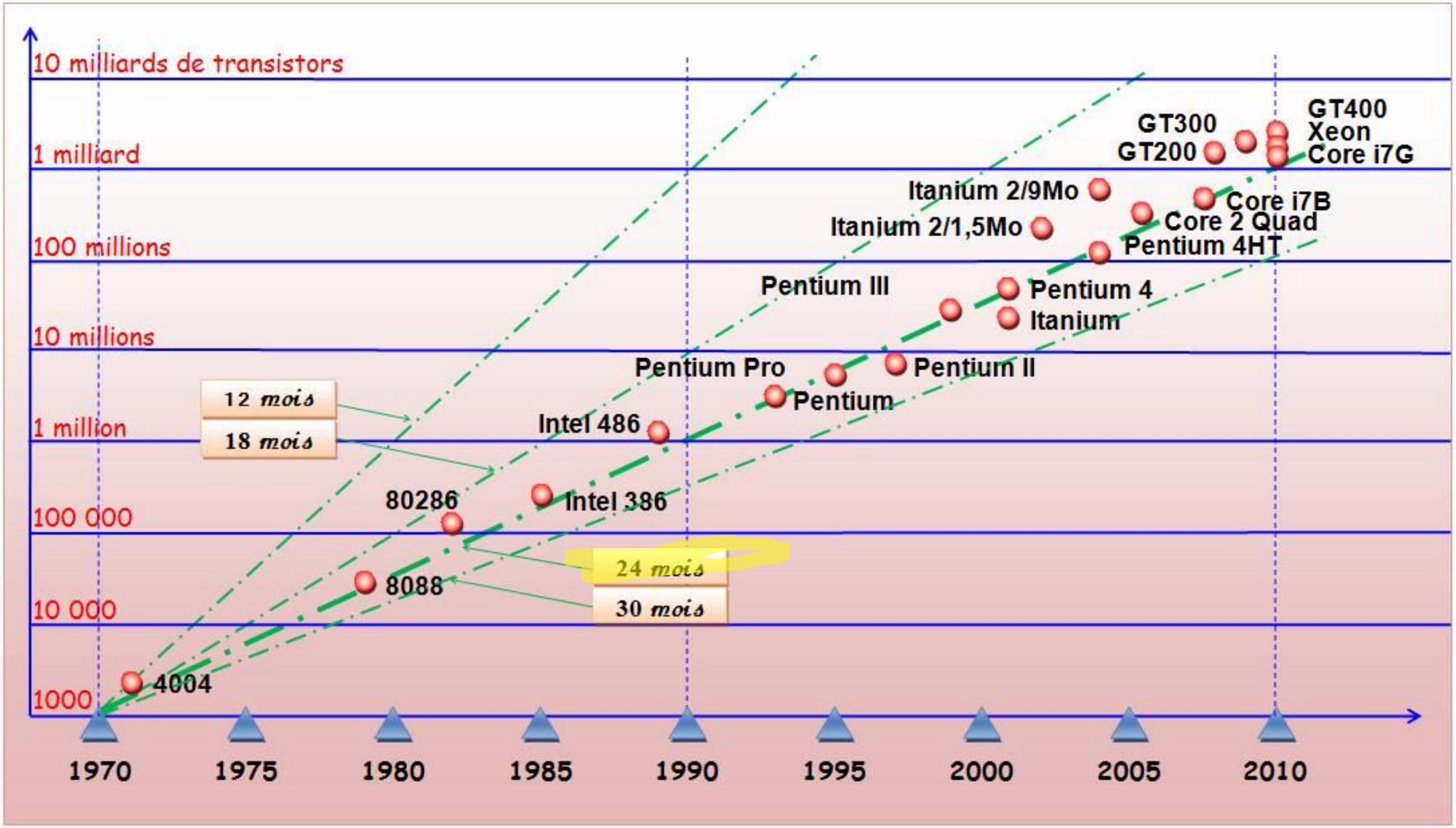


To compute faster given the Moore’s law limit: multi core



The Free Lunch Is Over
A Fundamental Turn Toward Concurrency in Software
By Herb Sutter, March 2005

Moore law



http://villemin.gerard.free.fr/Multimed/Moore.htm

Modern chips: Tendencies to mitigate the end of Moore' law

- Even more specialized units (as FPUs are):
  - GPUs, Computational storage devices, Tensor processing units,.. => offloading workloads from CPUs
- 3D transistors
- Non silicon & nano transistors : Molecular, DNA, …
- Quantum computers ?
- And… going back to more code optimization !
  - Also to decrease energy consumption, target better performance per watt, instead of pure performance

(Explicit) Multicore programming is needed: basic concepts

- Programmers must think on how to benefit from the multiple cores, even on a single chip
  - Can be helped by general purpose/high level patterns
    - ForkJoin, bag of tasks, map&reduce functions, //workflows …
- Each core runs its own thread(s) of computation
  - Exhibit threads in the computation
    - Could be through compilation mechanisms, dedicated APIs or language constructs
  - Hopefully, runtime+O.S. will schedule threads on the various cores to run in parallel
  - Collaboration between threads through the shared memory of the chip
    - Needs (classic) concurrency control mechanisms

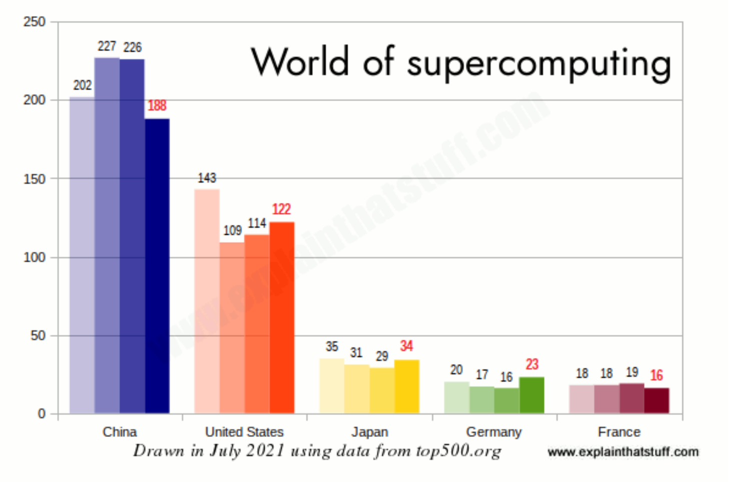
Why parallel computing ?

- Solve a problem faster
  - Rely on machines that are more powerful
    - Increase chip performances (Moore law)
    - Or Number of processing elements >> 1
- Solve problems whose size is too high to be handled by one single machine
  - 90% of data volume created in the past 2years alone!
  - Cut the problem into sub-problems
    - Solve sub-problems in parallel on **several** machines
    - Interactions between sub-computations ?
      - Message passing on a (high tech) network (eg infiniband)
      - And/or Through a physical or virtual shared memory
        - NUMA: Non Uniform Memory Access
        - typical of GPU numerous memory / cache levels

Massively parallel computing / HPC Supercomputers

- Many-core to super computing
  - Many chips, interconnected
  - Target Exascale: 10^{18} FLOPS
    en.wikipedia.org/wiki/Exascale_computing

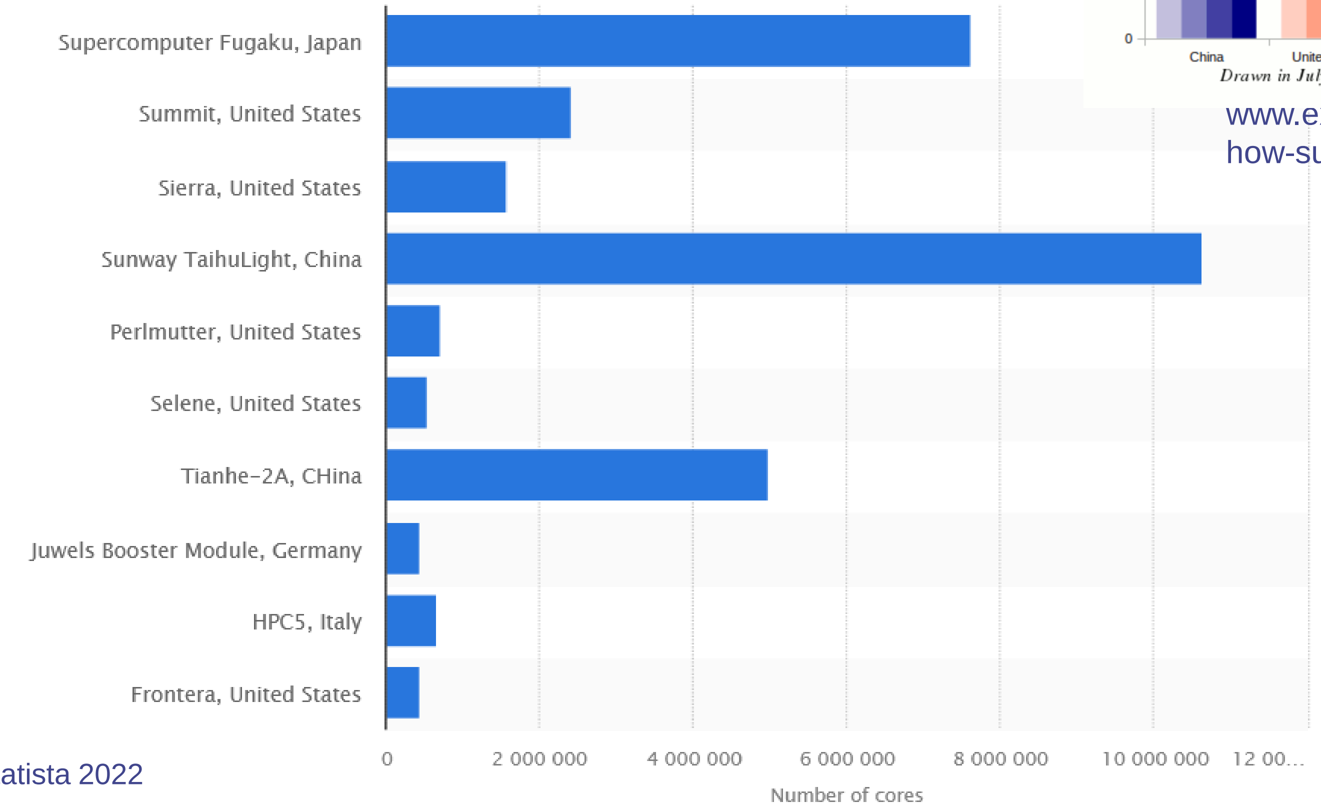
Who has the most supercomputers? 2017-2020



Drawn in July 2021 using data from top500.org

www.explainthatstuff.com

www.explainthatstuff.com/
how-supercomputers-work.html



© Statista 2022

Number of computer cores in the 10 fastest supercomputers in the world in 2020

Top'500

- International ranking of supercomputers
  - https://top500.org/lists/top500/2022/11/ latest list
  - Based on their performance in FLOPS
    - Floating Point Operations Per Second (eg 10FLOPS=pocket calc)

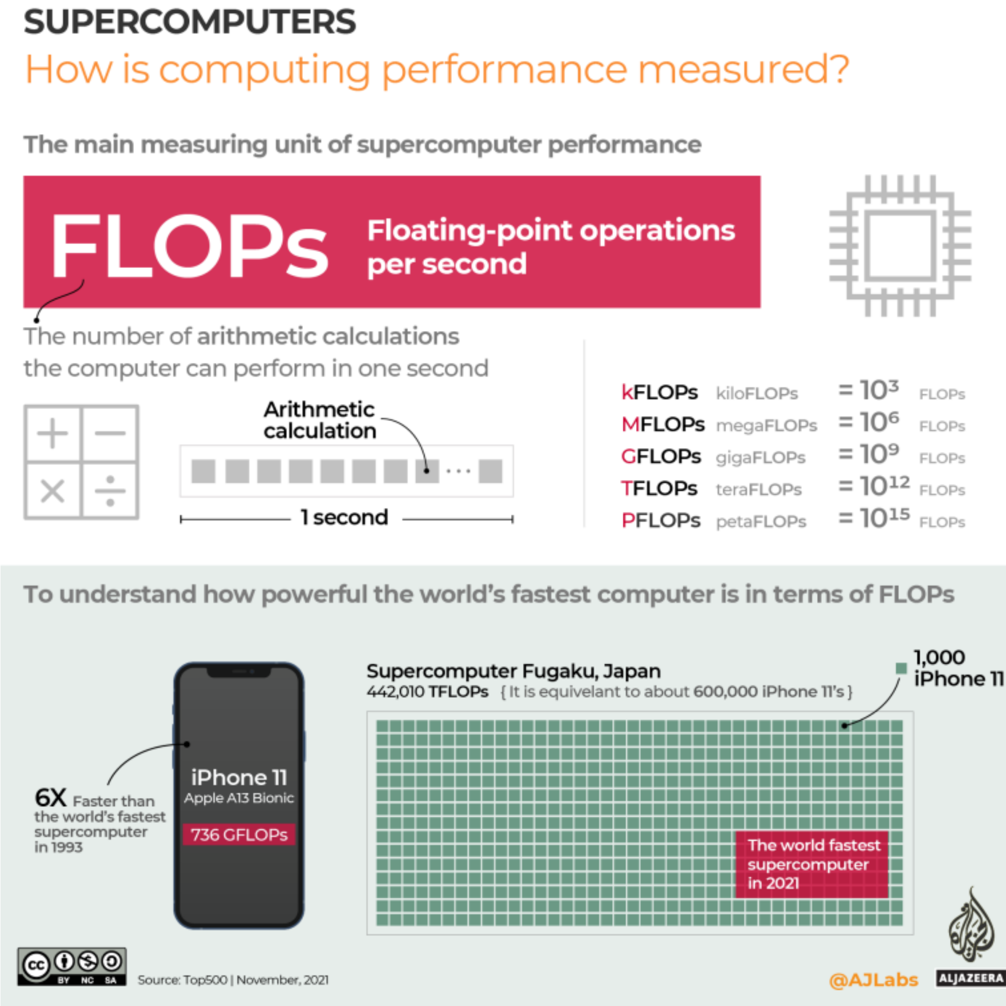
  on specific benchmarks (eg Linpack : solving the system Ax=b)
  - From https://www.exascaleproject.org/what-is-exascale/
    At a quintillion (10^{18}) calculations each second, exascale supercomputers will more realistically simulate the processes involved in precision medicine, regional climate, additive manufacturing, the conversion of plants to biofuels, the relationship between energy and water use, the unseen physics in materials discovery and design, the fundamental forces of the universe, and much more.

HPL Benchmark

HPL is a High-Performance Linpack benchmark implementation. The code solves a uniformely random system of linear equations and reports time and floating-point execution rate using a standard formula for operation count.

HPL is written in a portable ANSI C and requires an MPI implementation as well as either BLAS or VSIPL library. Such choice of software dependencies gives HPL both portability and performance.

HPL is often one of the first programs run on large computer installations to produce a result that can be submitted to TOP500.



SUPERCOMPUTERS

How is computing performance measured?

The main measuring unit of supercomputer performance

FLOPs Floating-point operations per second

The number of arithmetic calculations the computer can perform in one second

kFLOPs	kiloFLOPs	= 10^3	FLOPs
MFLOPs	megaFLOPs	= 10^6	FLOPs
GFLOPs	gigaFLOPs	= 10^9	FLOPs
TFLOPs	teraFLOPs	= 10^{12}	FLOPs
PFLOPs	petaFLOPs	= 10^{15}	FLOPs

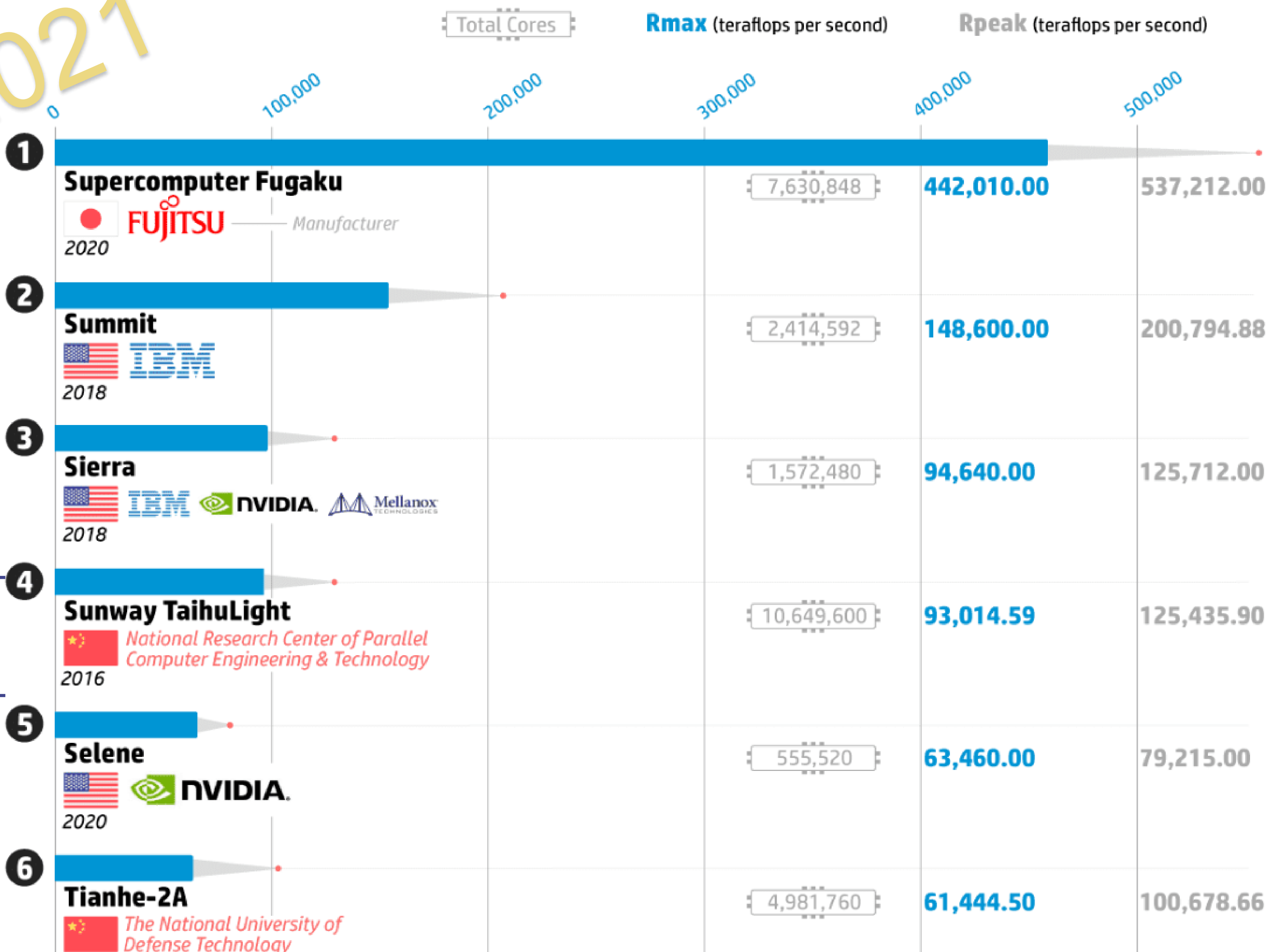
To understand how powerful the world's fastest computer is in terms of FLOPs

Supercomputer Fugaku, Japan
442,010 TFLOPs { It is equivelant to about 600,000 iPhone 11's }

Source: Top500 | November, 2021

In supercomputing, **Rmax** and **Rpeak** are scores used to rank supercomputers on their performance. A computer's **Rmax** score ranks its maximum achieved performance, and the **Rpeak** score ranks its theoretical peak performance. They are typically measured in **flops** (floating-point operations per second).

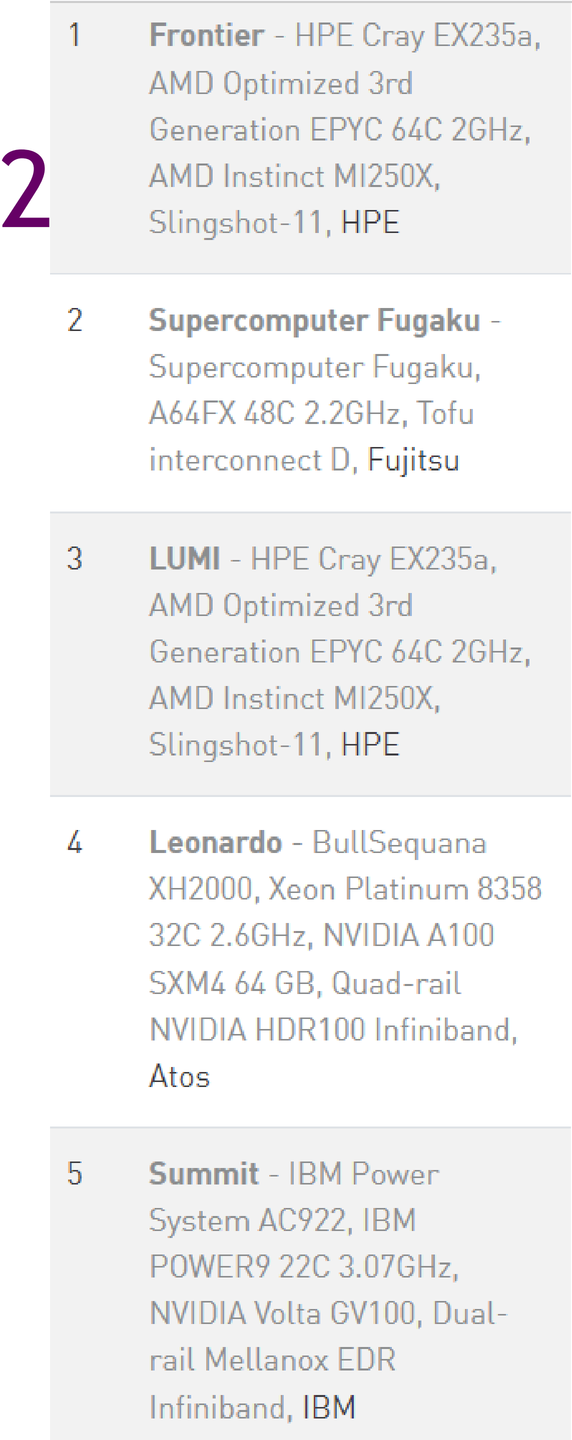
2021



https://www.hp.com/us-en/shop/tech-takes/fastest-supercomputers-ever-built

Top’500 nov 2022

https://eurohpc-ju.europa.eu/lumi-leonardo-among-worlds-five-most-powerful-supercomputers-2022-11-14_en



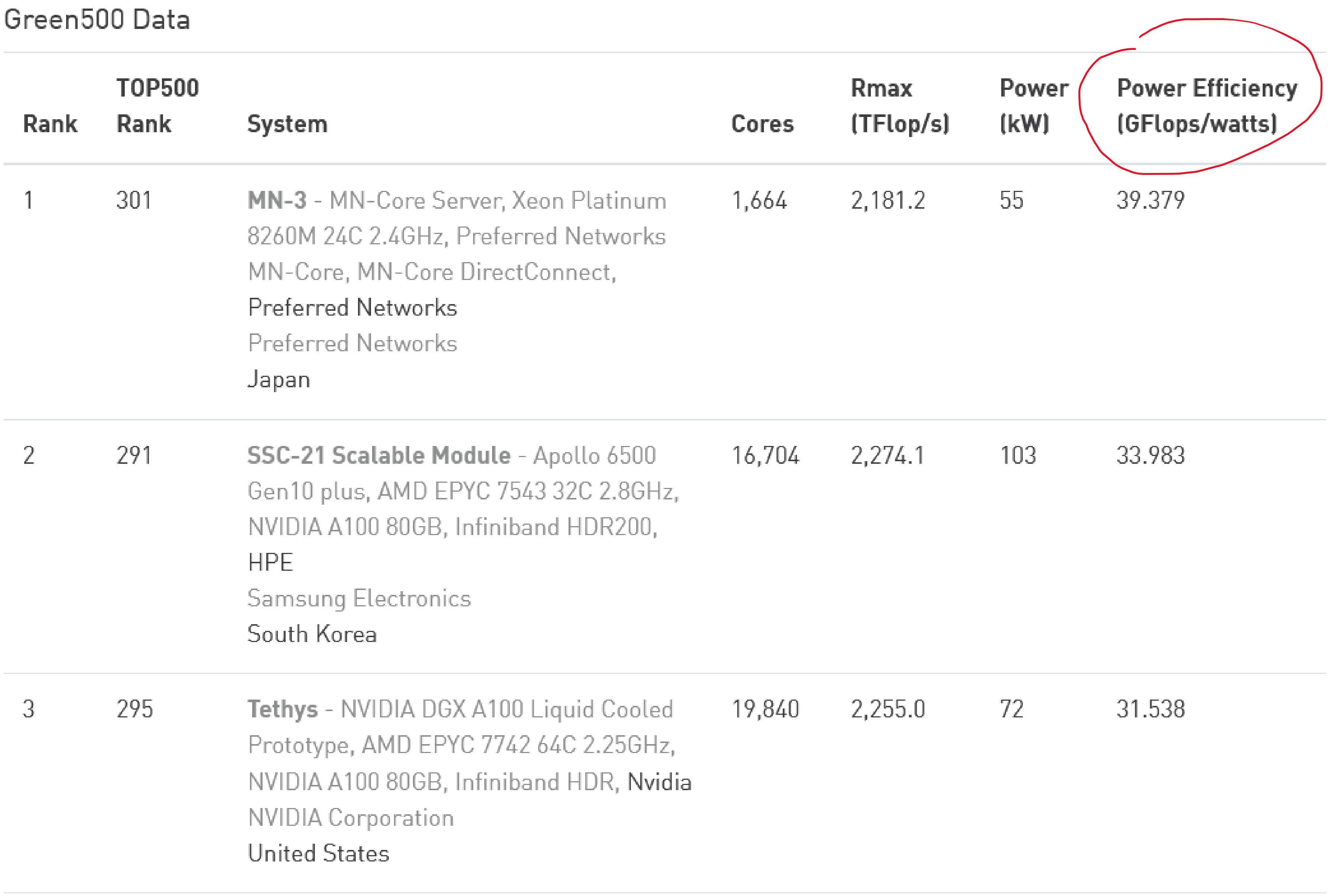
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE
2	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu
3	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE
4	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos
5	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM

- The Frontier system at the Oak Ridge National Laboratory, Tennessee, USA remains the No. 1 system on the TOP500 and is still the only system reported with an HPL performance exceeding one Exaflop/s. Frontier brought the pole position back to the USA on the June listing with an HPL score of 1.102 Exaflop/s.

- The LUMI system at EuroHPC/CSC in Finland entered the list last June at No. 3. It is again listed as No. 3 but only thanks to an upgrade of the system, which doubled its size. With its increased HPL score of 309 Pflop/s it remains the largest system in Europe.

- The only new machine to grace the top of the list was the No. 4 Leonardo system at EuroHPC/CINECA in Bologna, Italy. The machine achieved an HPL score of .174 EFlop/s with 1,463,616 cores.

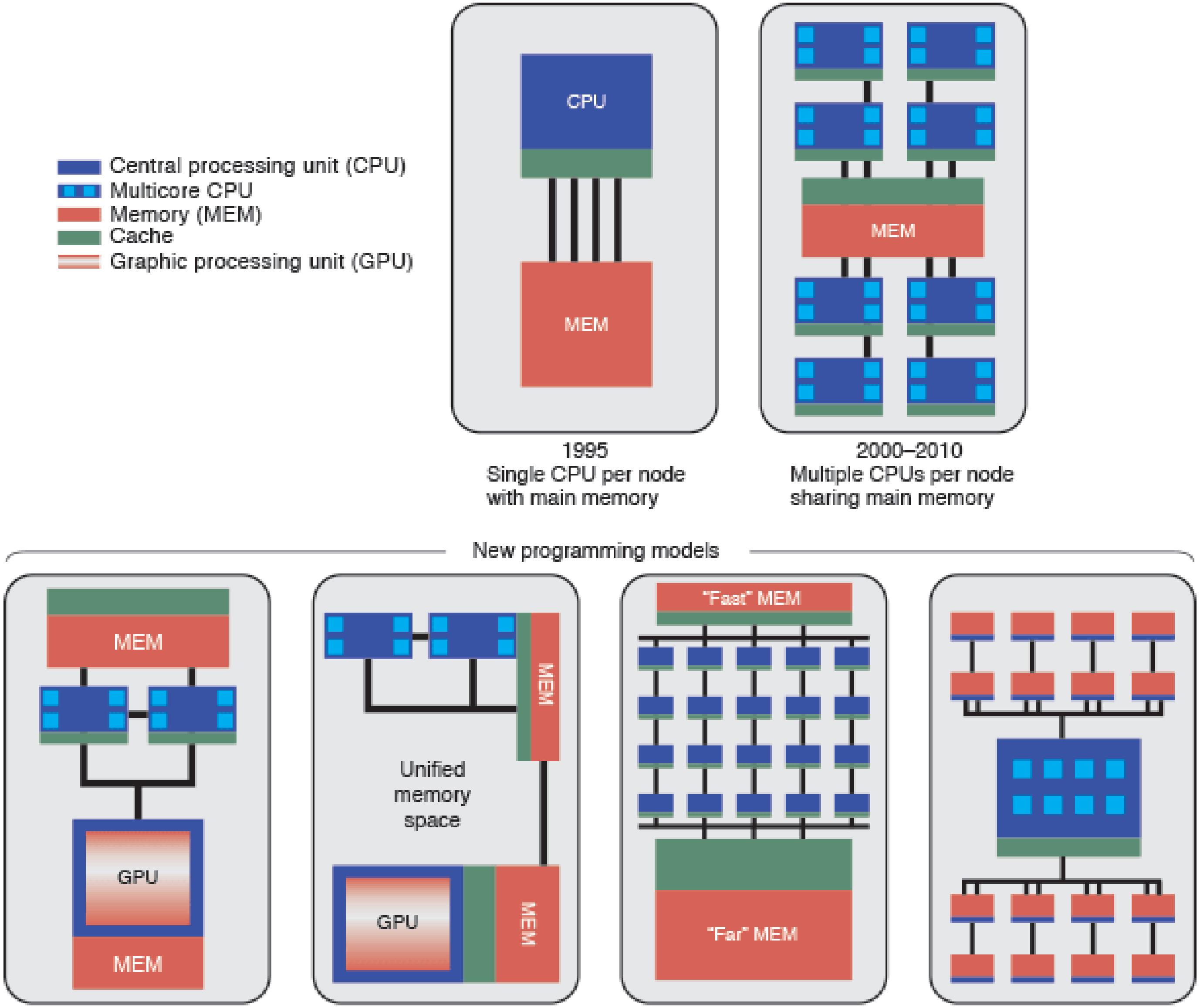
Top’500 sublist for green supercomputers, 2021



Green500 Data

Rank	TOP500 Rank	System	Cores	Rmax (TFlop/s)	Power (kW)	Power Efficiency (GFlops/watts)
1	301	**MN-3** - MN-Core Server, Xeon Platinum 8260M 24C 2.4GHz, Preferred Networks MN-Core, MN-Core DirectConnect, Preferred Networks Preferred Networks Japan	1,664	2,181.2	55	39.379
2	291	**SSC-21 Scalable Module** - Apollo 6500 Gen10 plus, AMD EPYC 7543 32C 2.8GHz, NVIDIA A100 80GB, Infiniband HDR200, HPE Samsung Electronics South Korea	16,704	2,274.1	103	33.983
3	295	**Tethys** - NVIDIA DGX A100 Liquid Cooled Prototype, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100 80GB, Infiniband HDR, Nvidia NVIDIA Corporation United States	19,840	2,255.0	72	31.538

Evolution of memory archis



- 🟦 Central processing unit (CPU)
- ▦ Multicore CPU
- 🟥 Memory (MEM)
- 🟩 Cache
- ▨ Graphic processing unit (GPU)

1995
Single CPU per node
with main memory

2000–2010
Multiple CPUs per node
sharing main memory

2000–2010
Accelerators usher in
era of heterogeneity

2014
Accelerators share common
view of memory with CPU

2015
Simple low-power cores and
non-uniform memory access

2017–2019
Processor in memory

Plan

1. Motivation

Which includes A very brief glimpse of parallel machines

2. Introduction to the theoretical PRAM model and typical parallel algorithms

Theoretical model of parallelism

- On the Parallel Random Access Machine: PRAM
- Why studying problems in this PRAM model ?
  - Establish the complexity of a problem to be solved in parallel
  - Provides a complexity of a problem dependant of the number of processors to be involved to solve it
  - A lower bound: even with an unlimited number of processors, a problem cannot be solved in less than time T, in parallel
  - Optimality of the way to solve a problem with P processors compared to the time it would take to solve it optimally and sequentially (on 1 processor)

PRAM

- A PRAM (Parallel Random Access Machine) is an abstract machine (as a RAM is)
  - An unbounded number of parallel processors
  - A shared (flat) global memory with direct (Random) access
  - A sequence of instructions to run
  - A **single** instruction pointer that all procs. follow

  - => the execution is synchronous: one single (and same) instruction at a time, on each proc. (SIMD style)
  - => no need of synchronisation barriers to cross before going to the next PRAM instruction!!

PRAM variants regarding access mode to the global shared memory

- As all processors may access at the same time the memory, at same memory cell/address
- … need to establish some rules
  - EREW (exclusive read, exclusive write) :
    - It is forbidden to read or write to a same @ in parallel
  - CREW (concurrent read exclusive write)
    - Reading the same @ by several procs allowed
  - CRCW (concurrent read, concurrent write)
    - Arbitrary mode: the write op. of only one arbitrarily chosen process to that same @ succeeds
    - Consistant mode: all write operations succeed only if the value to write at a same @ is the same
    - Associative mode: an associative operation is run on all values to be written at a same @, before the result is written at that @

Simulation between PRAM variants

- Simulate (or emulate) one variant of a PRAM onto another variant
  - The cost in //time and proc. needed is well known
  - => the complexity of one algorithm for a given PRAM variant becomes easily transposable onto another PRAM variant
  - Ex: a computation that takes Parallel time= O(1) on a CRCW PRAM takes O(log p) on a p-CREW PRAM (using p processors)

Exo

Elements of the algorithmic language

- A simple Pseudo-language
  - seq and // loops, data structures (eg arrays, lists, …) with random access operations to any element (index j), conditional tests/branches
- All variables are shared –by default
- //Loop instruction example

  -
    ```
    Pour chaque (proc number) i en parallèle
    faire
                x[i] = y[i]
    FinPour
    ```
    - Here, read operations of the y[i] are executed in //, then, in // the write operations into the x[i] are run.
  - Notice that For each **i** in parallel => proc number **i** is active

Example: compute the maximum (v1)

- We look for max(T[i]) where T has n entries
  - Each T(i) stores a number
- We use a PRAM having n^2 procs.
  - Each PRAM proc will acccess T[i] and T[j]
- We use an auxiliary array of n booleans: m(i)

```
pourchaque 1 < i < n en parallele
  m[i]=TRUE
pourchaque 1< i,j<n en parallele
  if ( T[i] < T[j])
      m[i] = FALSE
pourchaque 1 < i < n en parallele
  if ( m[i] = TRUE )
      max = T[i]
```

Exo: intrinsic seq complexity

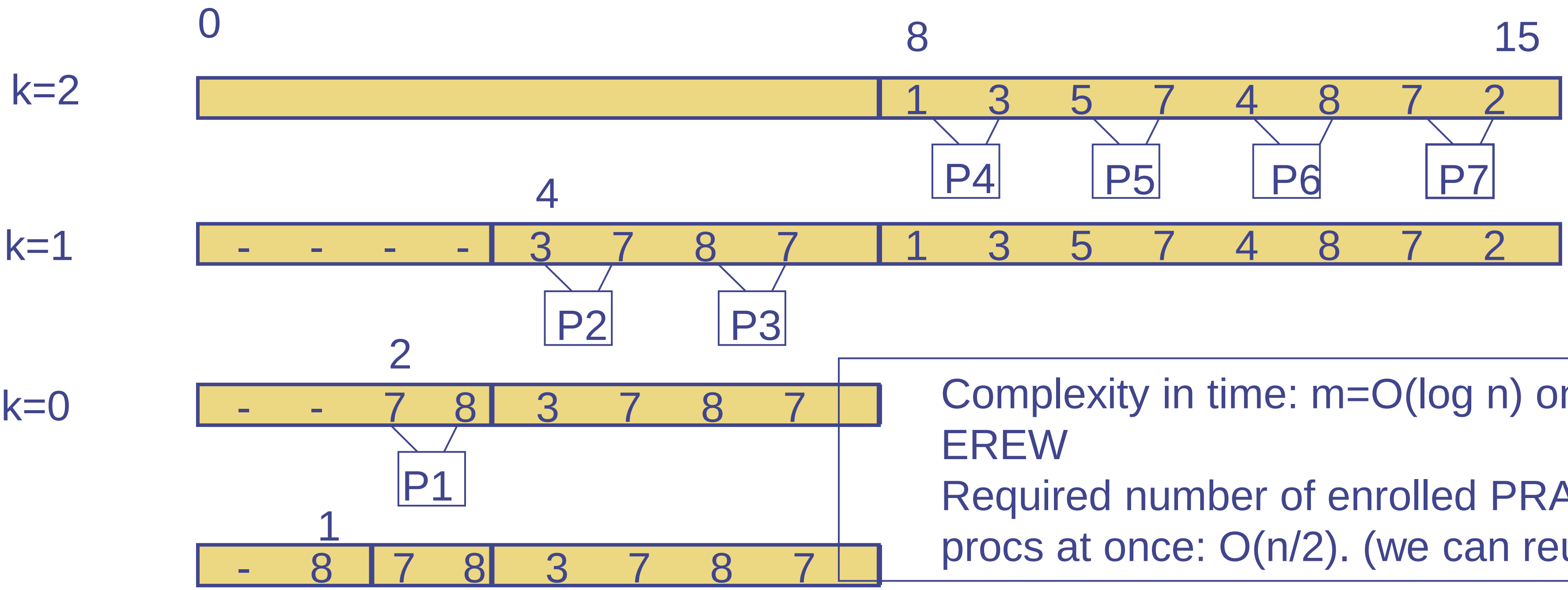
- Complexity :
  - //time= O(1) on an arbitrary CRCW (is enough!)
  - O(log n)// time on a EREW or a CREW

Exo: simulation complexity

Example: compute the maximum (v2)

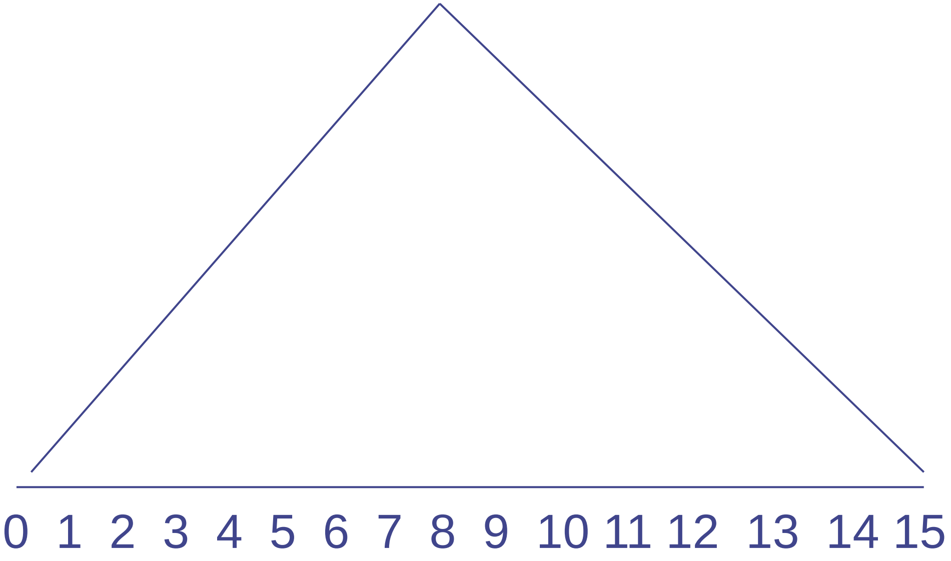
if $n=2^m$, if array A has size 2*n*, and if one aims to compute the maximum of the *n* values of A stored at position A[n], A[n+1],...,A[2n-1], the algorithm outputs the result in A[1] as follows:

```
Pour (k=m-1; k>=0; k--)
        Pour chaque j from 2^k to 2^(k+1)-1 en parallele
                A[j] = max(A[2j],A[2j+1]);
```



Complexity in time: m=O(log n) on a EREW
Required number of enrolled PRAM procs at once: O(n/2). (we can reuse)

Time //



log (2^4)
= 4

O(log (n))

procs

2^4

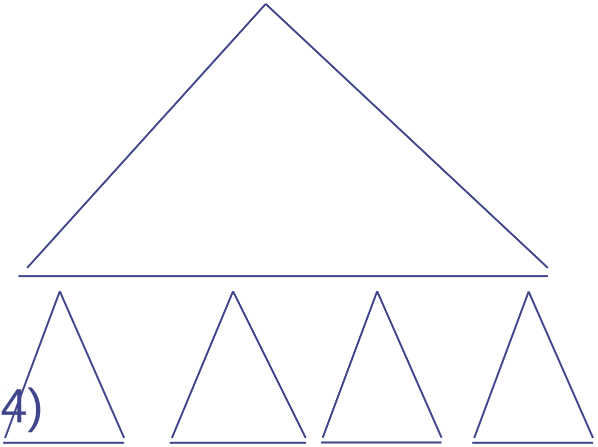
O(n)

WORK = //Time * Procs= O(log n) * O(n)

Time //

log (2^4 / 4)
= 2

log (n / logn)



Time for treatment of
subtree in sequential =O(4)

Time // O(4)

WORK = O(log (n)) * O(n/log n) = O(n)
So, it is optimal compared to the seq. work = O(n)

procs

2^4 / 4 = 4

n / logn

Generalisation

- Traversal of a complete binary tree of logarithmic depth
  - From bottom (leaves) to the top (root), and sub problems results are merged in pair
    - Still, it is not a pure ‘divide and conquer’ approach
    - As the sub problems standing at the leaves level exist naturally
    - As new sub problems pop up when the algorithm goes one level up
- How to get an optimal PRAM WORK complexity ?
  - Reduce the needed total number of processors (resources)…
  - …while not increasing the time complexity
  - By reducing the problem size to be solved by the PRAM algo
    - Instead of n procs => n/log n procs in order to solve problem of size n/logn
    - Instead of tree traversal= log n //time => log (n/logn) => O(logn) //time
  - Add to it the //time needed by all processes to solve each « big » inital sub-problem using the seq. approach:
    - O(logn) //time using the n/log n procs. , each proc handle logn input data
    - No interaction needed during sub problems solving => <u>Embarrassingly //</u>

Exo: write PRAM algo