High Performance Computing

A.k.a SuperComputing

Introduction to Parallel Algorithmics

SuperComputing

- Why ?
- How to achieve this ?
  - Hardware level
  - Software level

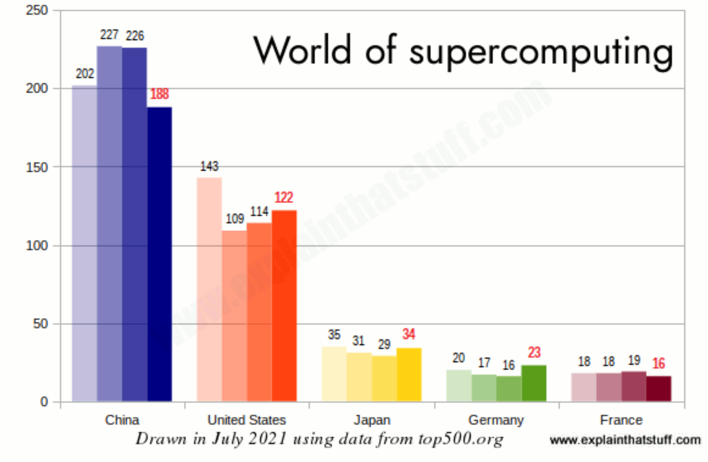
Why parallel computing ?

- Solve a problem faster
  - Rely on machines that are more powerful
    - Increase chip performances (Moore law)
    - Or Number of processing elements >> 1: multicore to manycore processors (eg 1000+ cores, includes GPU cards)
- Solve problems whose size is too high to be handled by one single machine
  - 90% of data volume ever 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), through e.g. the MPI standard
      - And/or Through a physical or virtual shared memory
        - NUMA: Non Uniform Memory Access
        - typical of GPU numerous memory / cache levels

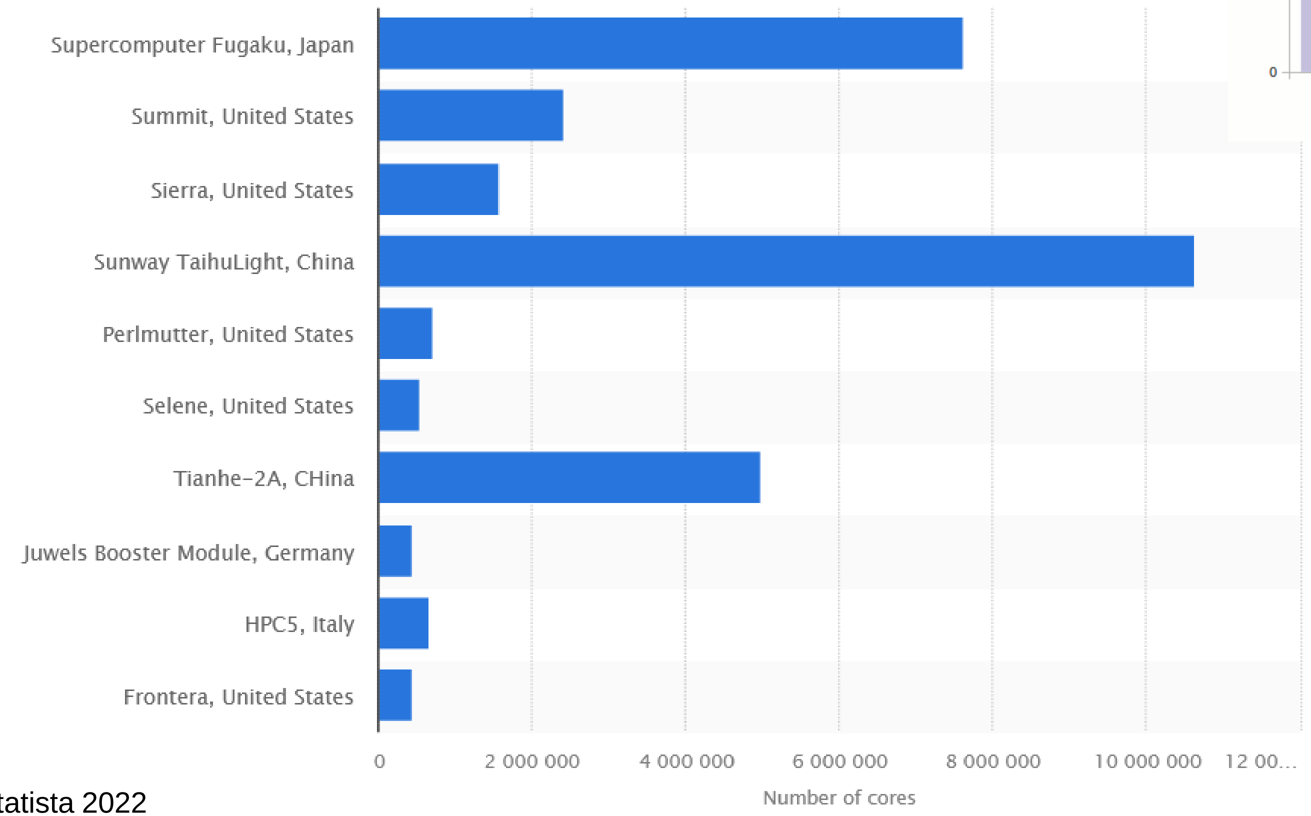
Massively parallel computing / HPC / Supercomputers

- 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



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



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

Top’500

- International ranking of supercomputers
  - November 2024 | TOP500 latest list
  - Given 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.

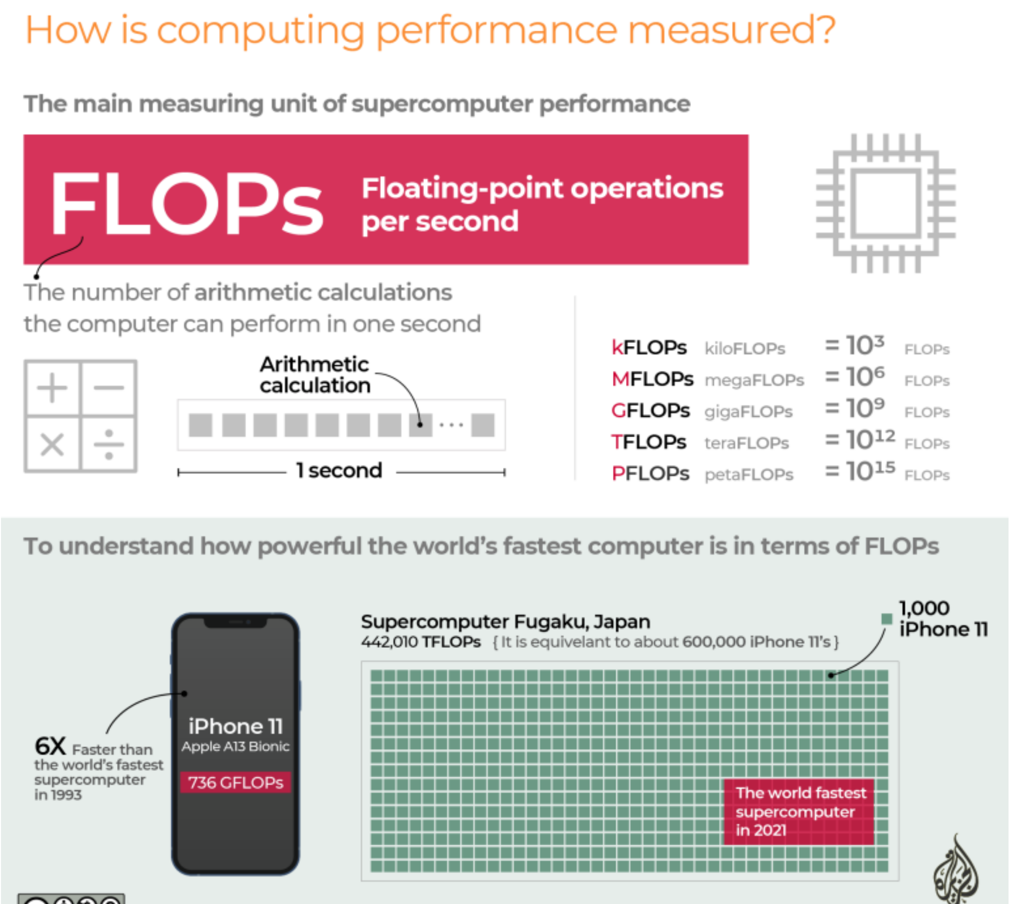
HPL Benchmark

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



Top'500 nov 2023

1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Microsoft Azure United States
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland

- Another new system named Eagle, installed in the Microsoft Azure Cloud in the USA, has take the No. 3 spot. This is the highest rank a cloud system has ever achieved on the TOP500. In fact, it was only 2 years ago that a previous Azure system was the first cloud system ever to enter the TOP10 at spot No. 10. This Microsoft NDv5 system has an HPL score of 561.2 PFlop/s and is based on Intel Xeon Platinum 8480C processors and NVIDIA H100 accelerators.
- Fugaku has moved to its current ranking of No. 4 after achieving No. 2 in the June 2023 list and holding the No. 1 spot from June 2020 until November 2021. This system is based in Kobe, Japan, and has an HPL score of 442.01 PFlop/s. It remains the highest ranked system outside the USA.
- The LUMI system based at Euro HPC/CSC in Kajaani, Finland, achieved the No. 5 spot with an HPL score of 379.70 PFlop/s. This system is the largest in Europe and has seen multiple upgrades that keep it near the top of the list, thi time improving from an HPL score of 309.10 PFlop/s. on the last list.

2024

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**El Capitan** - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	**HPC6** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461

https://www.servethehome.com/microsoft-azure-eagle-is-a-paradigm-shifting-cloud-supercomputer-nvidia-intel/

2024

The 64th edition of the TOP500 reveals that **El Capitan** has achieved the top spot and is officially the third system to reach exascale computing after Frontier and Aurora. Both systems have since moved down to No. 2 and No. 3 spots, respectively. Additionally, new systems have found their way onto the Top 10.

The new **El Capitan** system at the **Lawrence Livermore National Laboratory** in California, U.S.A., has debuted as the most powerful system on the list with an HPL score of 1.742 EFlop/s. It has 11,039,616 combined CPU and GPU cores and is based on AMD 4th generation EPYC processors with 24 cores at 1.8GHz and AMD Instinct MI300A accelerators. El Capitan relies on a Cray Slingshot 11 network for data transfer and achieves an energy efficiency of 58.89 Gigaflops/watt. This power efficiency rating helped El Capitan achieve No. 18 on the GREEN500 list as well.

The **Frontier** system at **Oak Ridge National Laboratory** in Tennessee, U.S.A, has moved down to the No. 2 spot. It has increased its HPL score from 1.206 Eflop/s on the last list to 1.353 Eflop/s on this list. Frontier has also increased its total core count, from 8,699,904 cores on the last list to 9,066,176 cores on this list. It relies on Cray's Slingshot 11 network for data transfer.

The **Aurora** system at **Argonne Leadership Computing Facility** in Illinois, U.S.A, has claimed the No. 3 spot on this TOP500 list. The machine kept its HPL benchmark score from the last list, achieving 1.012 Exaflop/s. Aurora is built by Intel based or the HPE Cray EX – Intel Exascale Compute blade which uses Intel Xeon CPU Max Series Processors and Intel Data Center GPU Max Series accelerators that communicate through Cray's Slingshot-11 network interconnect.

The **Eagle** system installed on the **Microsoft Azure Cloud** in the U.S.A. claimed the No. 4 spot and remains the highest-ranked cloud-based system on the TOP500. It has an HPL score of 561.2 PFlop/s

The only other new system in the TOP 5 is the **HPC6** system at No. 5. This machine is installed at **Eni S.p.A** center in Ferrera Erbognone, Italy and has the same architecture as the No. 2 system Frontier. The HPC6 system at Eni achieved an HPL benchmark of 477.90 PFlop/s and is now the fastest system in Europe.

TOP500 Release

THE LIST

PRESS RELEASE

LIST HIGHLIGHTS

Statistics

PERFORMANCE DEVELOPMENT

SUBLIST GENERATOR

LIST STATISTICS

TREE MAPS

HISTORICAL CHARTS

Downloads

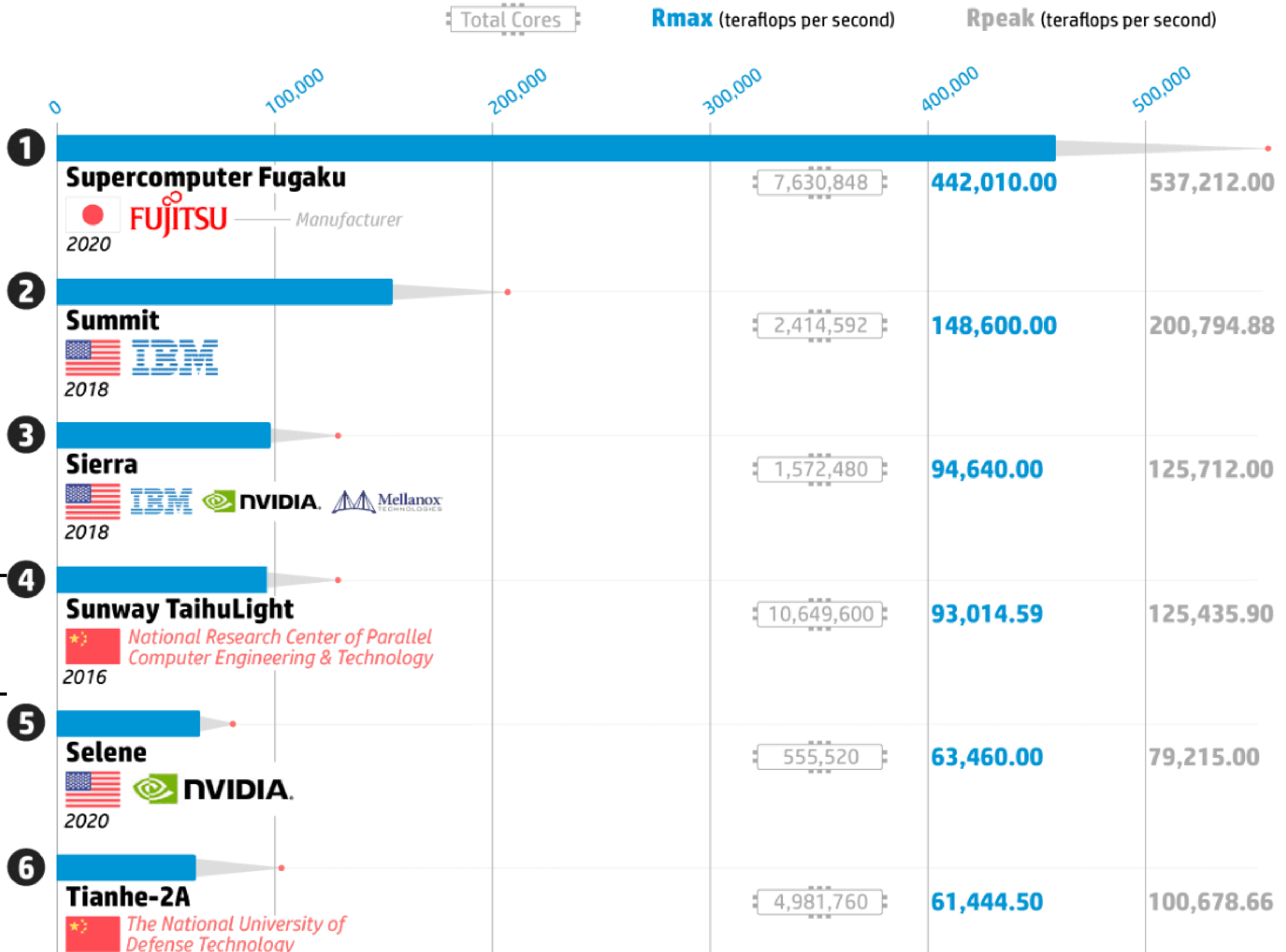
TOP500 LIST (XML)

TOP500 LIST (EXCEL)

2021

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

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).



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

Top'500 sublist for green supercomputers, 2023

https://www.top500.org/lists/green500/2023/11/

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	Power (kW)	Energy Efficiency (GFlops/watts)
1	293	**Henri** - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo Flatiron Institute United States	8,288	2.88	44	65.396
2	44	**Frontier TDS** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	120,832	19.20	309	62.684
3	17	**Adastra** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Grand Equipement National de Calcul Intensif - Centre Informatique National de l'Enseignement Suprieur (GENCI-CINES) France	319,072	46.10	921	58.021

Local HPC resources accessible through Job Schedulers

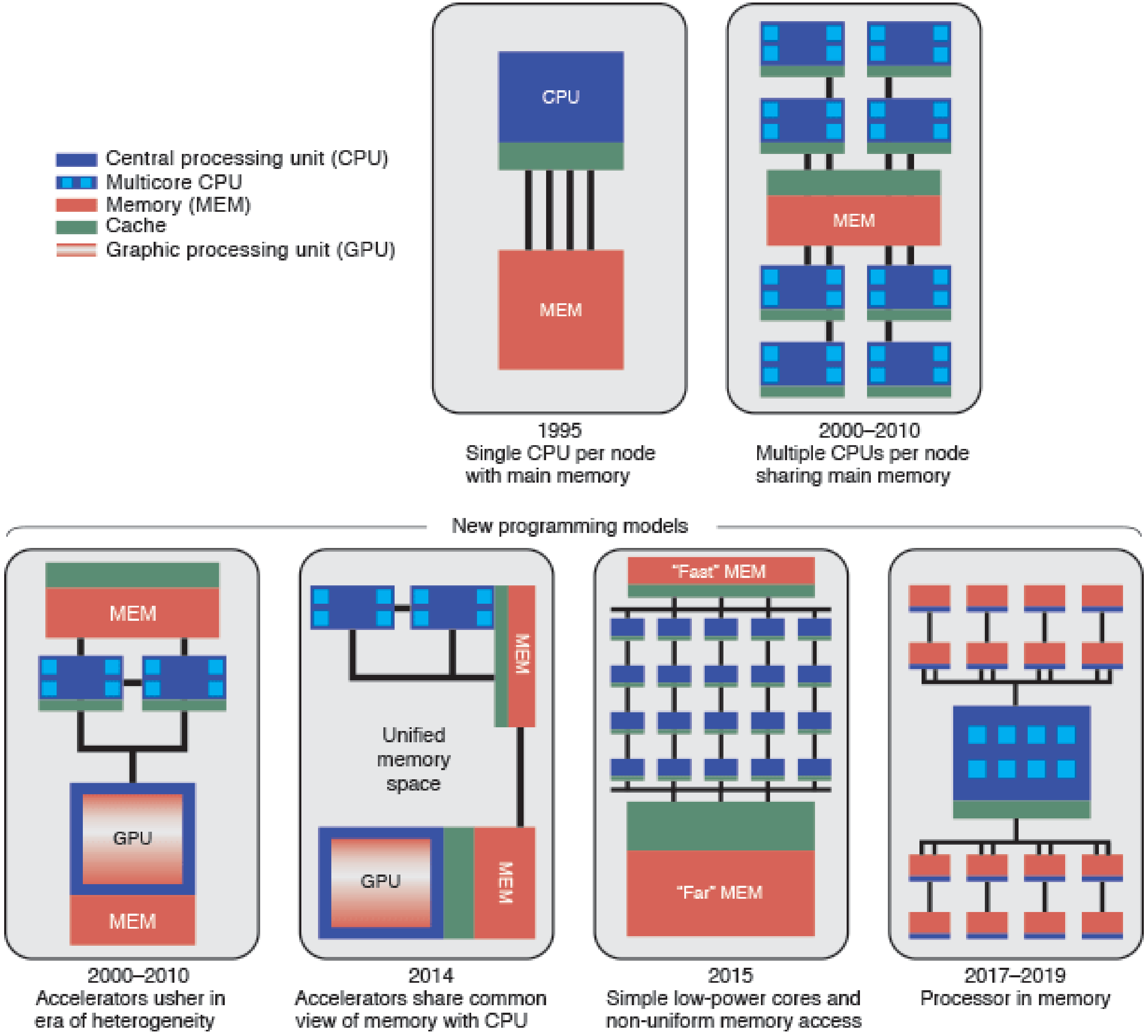
- [Abel and Boole (single machine VMs), direct SSH access]
- Polytech in house local cluster

- Maison de la simulation (MSI) of UniCA cluster
- The French (and EU wide now) Grid’5000 interconnection of high perf clusters
- GENCI https://www.genci.fr/**Qui sommes-nous ?**
  - GENCI (Grand Equipement National de Calcul Intensif) est une très grande infrastructure de recherche de classe IR*. Il s’agit d’un opérateur public créé en 2007 afin de démocratiser l'usage de la simulation numérique par le calcul haute performance associé aujourd’hui à l'usage de l'intelligence artificielle et bientôt aux dispositifs prototypes de calcul quantique.

- EU funded EuroHPC supercomputers
  - Eg Karolina

KAROLINA SUPERCOMPUTER

Evolution of memory archis



How to program a supercomputer

First exhibit the source of parallelism

Design an appropriate algorithm along an appropriate algorithmic model

Implement it using the right programming languages and runtime

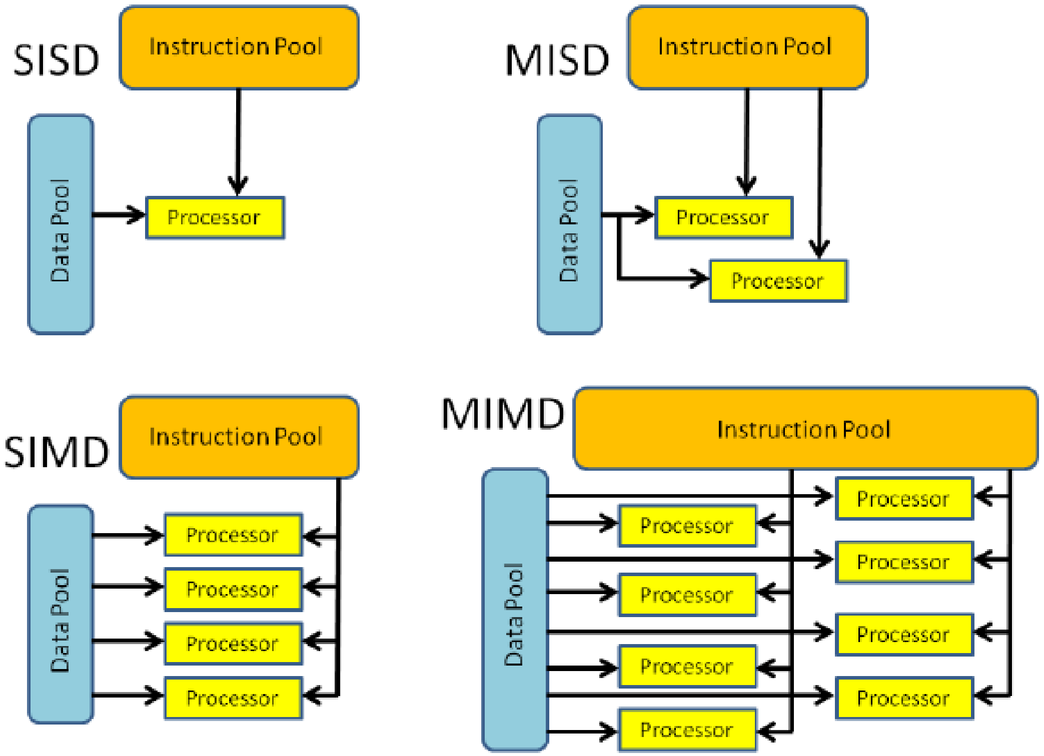
Beyond multi thread (« manual ») programming

- Multi thread « manual » programming
- Fork Join like « high-level » approaches
- Are not sufficient, as only apply to a single multi-core/many core CPU with shared flat memory
- To increase computing power, several CPUs are interconnected
  - By a virtual shared memory space, featuring different performance access
  - By a high-speed interconnection network: requires explicit message exchanges between CPUs not hosted on the same server
- Also, modern hardware like GPU cards
  - do have several levels of memory featuring different performance access
  - necessitates that all cores run <u>synchronously</u>: one (set of) parallel instruction at a time

Highly parallel computing model

- Change the viewpoint compared to a multi core program
  - where the « goal » is to feed each core with a thread of computation, spreading tasks
  - Parallelism comes from the numerous tasks (more or less independant and concurrent)
  - **Control or task level parallelism**
- When one seeks to solve big size problems, what is the source of parallelism ?
  - It comes from the –big – volume of data!
  - Data to which the same sort of treatment must be applied: **data parallelism**

The Flynn Classification (1972)



- A GPU is an SIMD co processor, more generally, SIMT (threads)
- There existed SIMD parallel machines

Quelques exemples de machines SIMD

- Des machines des années 80/90
  - Illiac IV, MPP, DAC, Connection Machine CM-1/2, MasPar MP-1/2
- Un grand retour aujourd'hui
  - Processeurs Intel et mode SSE/SSE-2 (unités vectorielles)
  - NVidia Tesla: jusqu'à 2x2496 threads à 745Mhz et 12GB de RAM
  - Intel Xeon Phi: 61 cores @ 1.238Ghz et 16GB de RAM

The theoretical Parallel Random Access Machine (PRAM) computation model

- Like a RAM/Turing machine is appropriate to exhibit algorithms
  - to solve problems in sequential
  - and evaluate their complexity in time and in space
- A PRAM is the playground to exhibit **data parallel** algorithms to solve problems in parallel
- Allows to fully abstract away from the hardware
- A PRAM is an ideal SIMD computer
  - No limit on memory size
    - Any big size data (collection of elements) fits in memory
    - Any data access is taking the same unit of time (1 unit of time, to Read or Write to an address in memory)
  - No limit on the number of processors (be it a CPU, a core, ...)
    - As many processors as data elements to be processed can be enrolled
  - No need to bother about synchronization among the computations done at each adress
    - There is one single instruction run at a time, in parallel on as many data in memory as needed

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
  - Data races may arise !!
- … 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 parallel time and number of processors 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
  - <u>Pour chaque </u>(proc number) i <u>en parallèle</u>
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 <u>For each</u> **i** <u>in parallel</u> => 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

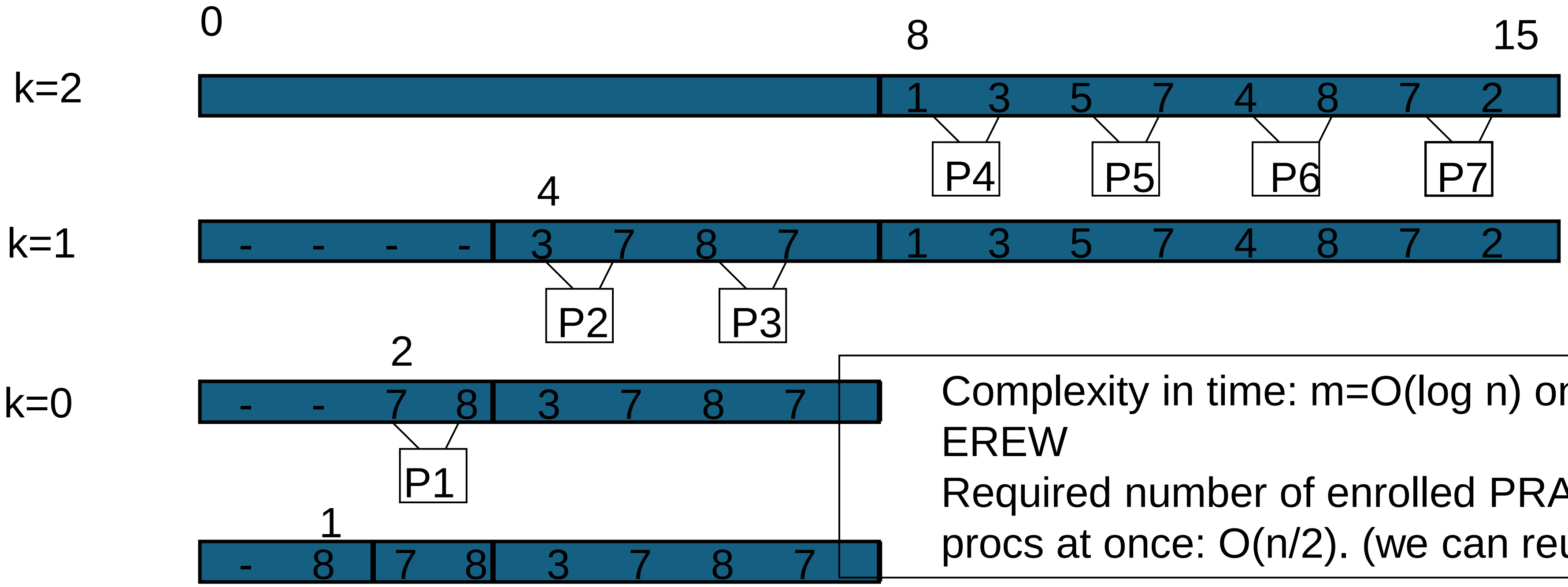
Example: compute the maximum (v2)

if $n=2^m$, if array A has size $2n$, 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)

Work of PRAM algorithms

- SURFACE : number of processors used by a PRAM
    - algorithm $A_{//}(N)$ working on a size N problem:
        - $H(A_{//}(N))$ = the **maximum** amount of **procs required** in a given parallel PRAM instruction, during the algo
- PARALLEL TIME $T_p(A_{//}(N))$ of the algo using P procs:
    - $T_p(A_{//}(N))$ = the number of computation steps when using P procs
- WORK : product of SURFACE by PARALLEL TIME
    - $W = \mathbf{H}(A_{//}(N)) * T_{H(A_{//}(N))}(A_{//}(N))$

      $= \quad P * T_P$

Speedup, Efficiency

- Consider the (best)Seq time to solve the problem $A_{seq}(N)$ whose time is $T_{seq}(N)$ (using 1 proc!)

- SPEEDUP (acceleration factor) of $A_{//}(N)$ using P procs
  - $S_p(N) = T_{seq}(N) \; / \; T_p(A_{//}(N))$
  - Theoretical goal is to have $S_p(N) = p$

By using p procs, the speed-up is p

- EFFICIENCY of $A_{//}(N)$ using P procs
  - e = Sequential work / Parallel work
  - $e = T_{seq}(N) * 1/ W$

    $= T_{seq}(N) \; / \; p^* \, T_p(A_{//}(N))$
  - Theoretical goal is to have e=1

Seq runtime has been converted into adding procs!

WORK EFFICIENCY

- A parallel algorithm is said to be WORK EFFICIENT:
  - Its work is of the same amount than the best sequential algorithm
  - i.e. e == 1

Hint: reduce the number of used processors to reduce the work

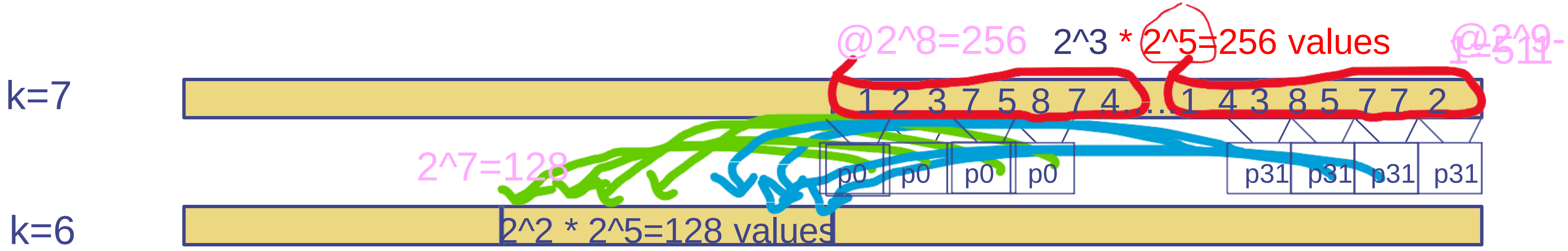
Let us see two approaches for this on the Maximum computation

Example: EREW maximum computation –v2 => v3

m=k=8, k’=6

N=2^k=2^8 = 256. p=2^(k-1)=2^7=128. p’=2^(k’-1)=2^5=32

each of the O(log(2^8))=O(8) instructions of the $Maxv2_{//}$(2^8) will be simulated by up to 2^5 procs, executing up to 2^(k-k’)=2^2=4 max binary operations



Pour (k=m-1; k>=0; k--) **/*still costs m parallel steps */**

/*enroll up to 2^(k-1) procs per step ? NO, just enroll up to q=2^(k’-1) */

<u>Pour chaque</u> proc q <u>en parallele</u>

Pour (l=0;l<2^(k-k’); l++) /*costs 2^(k-k’) seq time*/

A[zz] = max(A[yy],A[yy+1]);

<u>Complexity analysis</u>: Needed number of processors= 2^(k’-1).

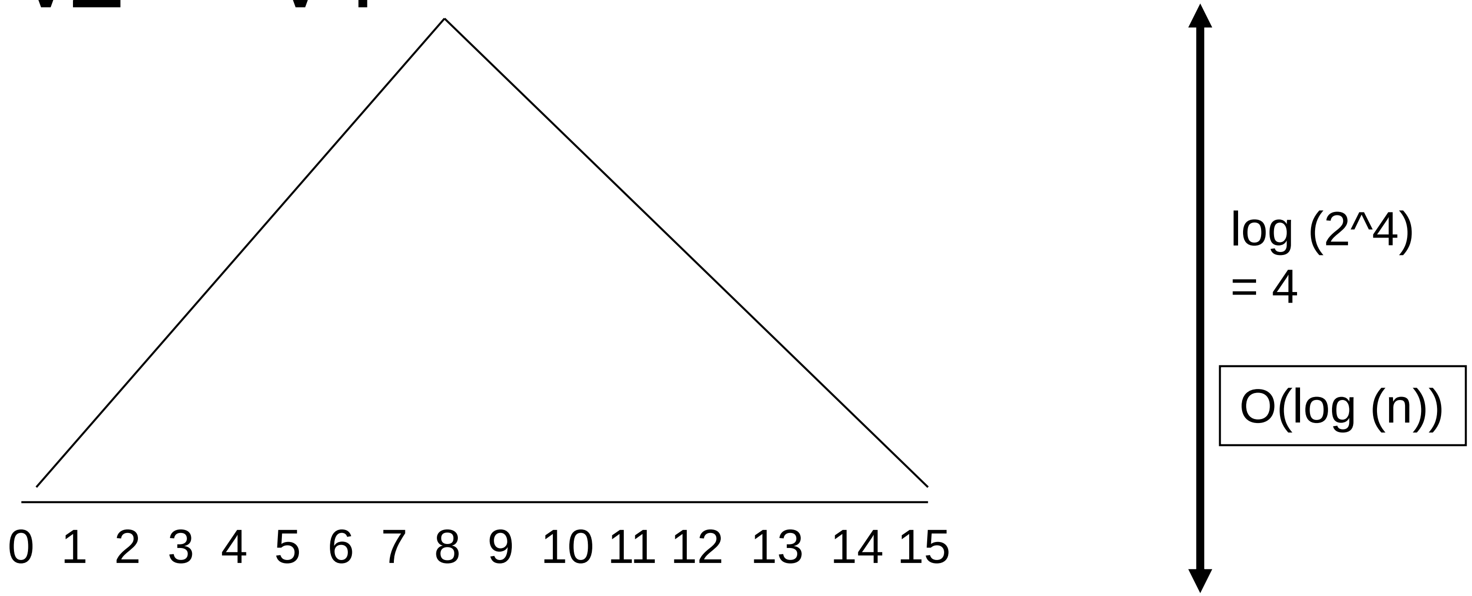
We still have k=log(n) sequential steps, and some of these do not take O(1) operations but more.

The worst step costs 2^(k-k’) seq time. Then, it decreases up to 1; but if we count **roughly**, parallel time is O(k*2^(k-k’))=O(k*2^(k-k’))*2^(k’-1). If k’=log k, parallel time=O(log n*n/logn)=O(n); work would be O(n*n/logn), clearly non optimal !

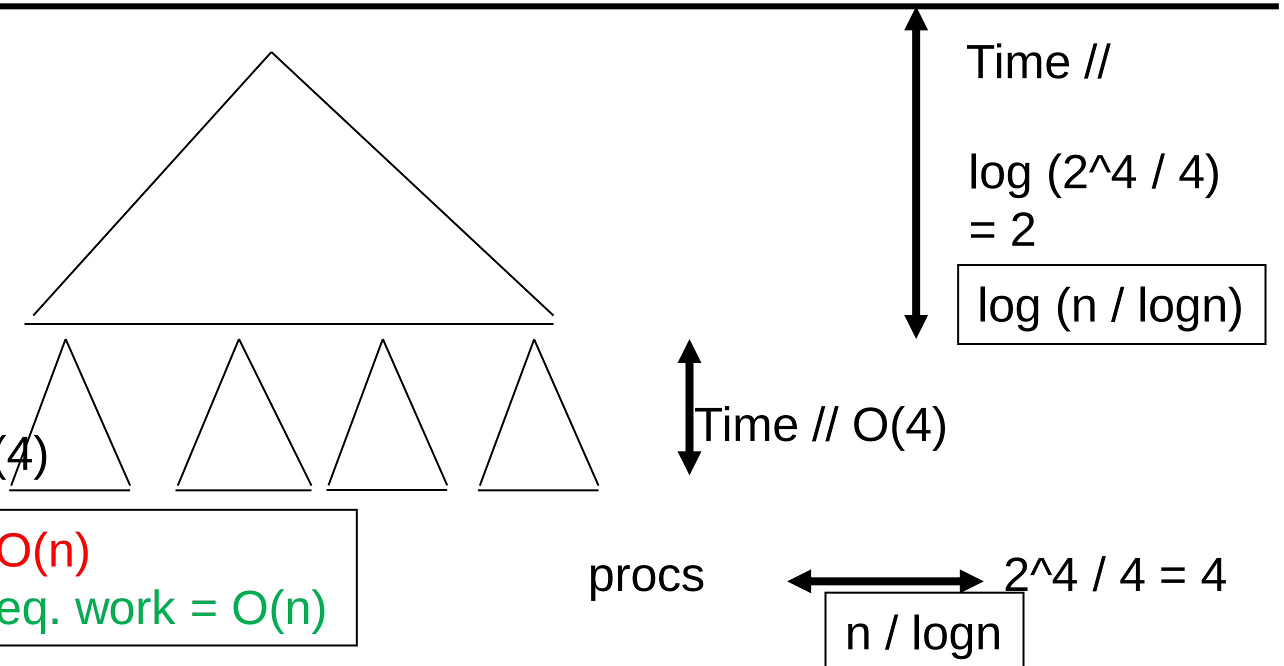
If instead we sum (in reverse order) the sequential steps : 1+2+4+...+2^(k-k’) = O(2x2^(k-k’)) time in total . If k’=log k, parallel time= O(n/ log n)

Work is bounded by 2^(k-k’+1) *2^(k’-1)=2^k=O(n) whatever k’ is.

Max computation v2 => v4



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



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

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

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 => Embarrassingly //

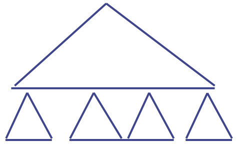
Exo: write PRAM algo

Write the v4 algorithm (cf exo3 of TD1)

- Cf the LMS to check what is the algorithm

WORK EFFICIENCY

- A parallel algorithm is said to be WORK EFFICIENT:
  - Its work is of the same amount than the best sequential algorithm
  - i.e. e == 1
- Recap of our Examples:
  - *max-v3* versus O(n) seq max.
    - Using the simulation theorem (not studied yet) we can simply deduce //time as follows:
      - O(t* (p / p')) ; when p=n, choose p'=n/logn ; t = log n
    - max-v3 time = log n * (n/n/logn) = $\log^2$ n (limiting factor: same t)
    - max-v3 work= n/logn * $\log^2$ n = n*log n => not work efficient
      - Because of the same t=log n on same initial size=n ….
  - *max-v2 with subtrees*= v4, versus O(n) seq max.
    - [log n + O(log (n/logn))] * (n/logn) = O(n), work efficient
    - Here max-v4 with subtrees applies the Brent Principle (not studied yet)



Typical massively parallel / data parallel languages for shared memory

- CUDA or OpenCL (see next lessons on GPU programming)
- OpenMP SIMD related pragma extensions (to exploit vectorization)
- Python Pytorch, NumPy library extensions
- Fortran parallel (HPF), Matlab
- Hidden data parallelism in some languages & API:
  - Java parallel streams (limited amount of data, not distributed but on the same JVM)
  - C++ STL
  - Data parallel Haskell