



Calcolo Parallelo dall'Infrastruttura alla Matematica

Calcolo parallelo: perché, quali infrastrutture, quali problemi?

Laurea Triennale e Magistrale in Matematica

Fabio Durastante

March 30, 2023





Table of Contents

1 Parallel computing: why?

- ▶ Parallel computing: why?
Linear Systems, *mon amour*
- ▶ Parallel computing: where?
Flynn's Taxonomy
Bēowulf
- ▶ Parallel computing: how?
An example of contemporary application
- ▶ First order of business: GIT
- ▶ Exercises



Scientific computing

1 Parallel computing: why?

“**Computational science** (also **scientific computing** or **scientific computation** (SC)) is a rapidly growing multidisciplinary field that uses advanced computing capabilities to *understand and solve complex problems*. It is an area of science which spans many disciplines, but at its core it involves the development of *models and simulations to understand natural systems*.”

Wikipedia



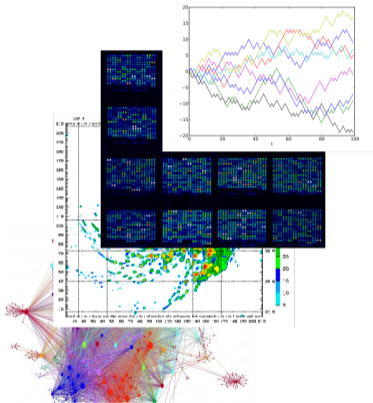
Leonardo, CINECA



What are the applications?

1 Parallel computing: why?

- Computational finance,
- Computational biology,
- Simulation of complex systems,
- Network analysis
- Multi-physics simulations,
- Weather and climate models,
- ...

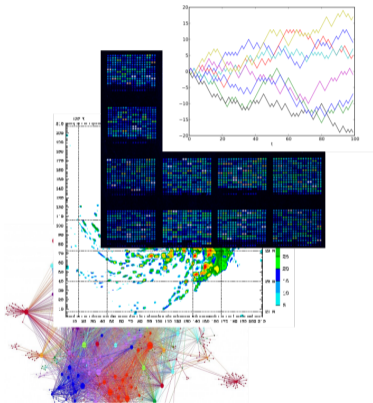




What are the applications?

1 Parallel computing: why?

- Computational finance,
- Computational biology,
- Simulation of complex systems,
- Network analysis
- Multi-physics simulations,
- Weather and climate models,
- ...



Why the need for **parallelism**?



Moore's law

1 Parallel computing: why?



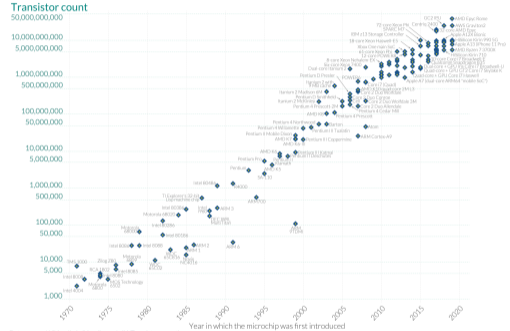
“The complexity for minimum component costs has increased at a rate of **roughly a factor of two per year**. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years.”

G. Moore, 1975

Moore's Law: The number of transistors on microchips has doubled 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





Moore's law

1 Parallel computing: why?



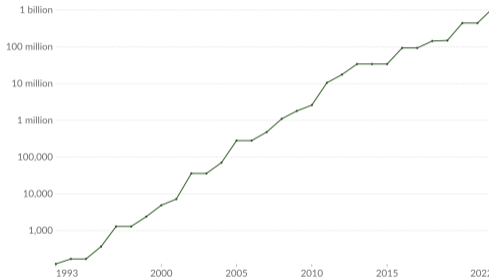
“The complexity for minimum component costs has increased at a rate of **roughly a factor of two per year**. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years.”

G. Moore, 1975

Computational capacity of the fastest supercomputers

The number of floating-point operations¹ carried out per second by the fastest supercomputer in any given year. This is expressed in **gigaFLOPS**, equivalent to 10^9 floating-point operations per second.

Our World
in Data



Source: TOP500 Supercomputer Database (2023)

OurWorldInData.org/technological-change • CC BY

¹ **Floating-point operation:** A floating-point operation (FLOP) is a type of computer operation. One FLOP is equivalent to one addition, subtraction, multiplication, or division of two decimal numbers.

Computers *should* reach the physical limits of Moore's Law at some point in the 2020s...exponential functions saturates physical capabilities!



Parallel computing: why?

1 Parallel computing: why?

- We are hitting the wall of single processor transistor count/computing capabilities,



Parallel computing: why?

1 Parallel computing: why?

- We are hitting the wall of single processor transistor count/computing capabilities,
- Some applications needs more memory than the one that could be available on a single machine,



Parallel computing: why?

1 Parallel computing: why?

- We are hitting the wall of single processor transistor count/computing capabilities,
- Some applications needs more memory than the one that could be available on a single machine,
- Optimization of sequential algorithms can bring us only to a certain extent



Parallel computing: why?

1 Parallel computing: why?

- We are hitting the wall of single processor transistor count/computing capabilities,
- Some applications needs more memory than the one that could be available on a single machine,
- Optimization of sequential algorithms can bring us only to a certain extent

“διαίρει καὶ βασίλευε”
(diáirei kái basíleue)



Parallel computing: why?

1 Parallel computing: why?

- We are hitting the wall of single processor transistor count/computing capabilities,
- Some applications needs more memory than the one that could be available on a single machine,
- Optimization of sequential algorithms can bring us only to a certain extent

“διαίρει καὶ βασίλευε”

(diáirei kái basíleue)

Dividi et Impera



Parallel computing: why?

1 Parallel computing: why?

- We are hitting the wall of single processor transistor count/computing capabilities,
- Some applications needs more memory than the one that could be available on a single machine,
- Optimization of sequential algorithms can bring us only to a certain extent

“διαίρει καὶ βασίλευε”

(diáirei kái basíleue)

Dividi et Impera

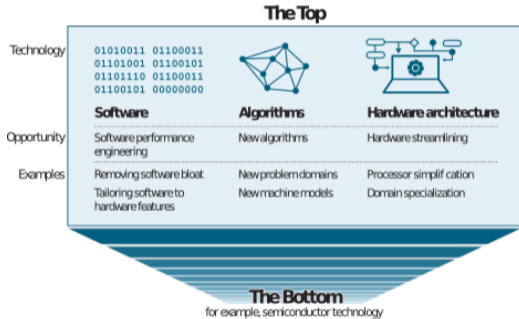
Therefore, we need

- **Algorithms** that can work in parallel,
- A **communications protocol** for parallel computation integrated with our programming languages,
- **Parallel machines** that can actually run this code.



The philosophy behind the effort

1 Parallel computing: why?



“As miniaturization wanes, the silicon-fabrication improvements at the Bottom will no longer provide the predictable, broad-based gains in computer performance that society has enjoyed for more than 50 years. Software performance engineering, development of algorithms, and hardware streamlining at the Top can continue to make computer applications faster in the post-Moore era.”

C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, “There’s plenty of room at the Top: What will drive computer performance after Moore’s law?”, *Science* (2020)



Linear Systems

1 Parallel computing: why?

Solve : $A\mathbf{x} = \mathbf{b}$,

where

- $A \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$,

is often the most time consuming computational kernel in many areas of computational science and engineering problems.



Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.

$n \sim 10$



“In a ground wire problem involving a **large** number of ground conductors, 13 simultaneous equations were solved...” – Dwight (1930)”

“The second machine, now in operation, was designed for the direct solution of **nine or fewer** simultaneous equations.” – Wilbur, J. B. (1936)



Linear Systems

1 Parallel computing: why?

Solve : $A\mathbf{x} = \mathbf{b}$,

where

- $A \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“Finally, though the labour of relaxation in three dimensions is prohibitively great, the future use of the **new electronic calculating machines** in this connexion is a distinct possibility” - Fox, L. (1947)



Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“The Ferranti PEGASUS computer, with a main store of 4096 words, can solve a maximum of **86 simultaneous equations** by its standard subroutine and takes about **45 minutes** to complete this calculation.” – Wilson, L. B. (1959)



Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“...the bound imposed by this is $m + n \leq 474$. In addition, this number of equations would fill one standard (1.800ft) reel of magnetic tape, and the **fifty-odd hours** taken in the calculation might be thought excessive.” -

Barron, Swinnerton-Dyer (1960)



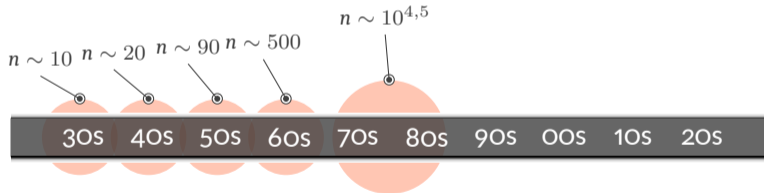
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“...handling problems involving sets of simultaneous equations of **two-thousandth order**, and SAMIS available through “Cosmic” at the University of Georgia, which can treat **up to 10,000 simultaneous equations.**” –

Melosh, Schmele (1969)



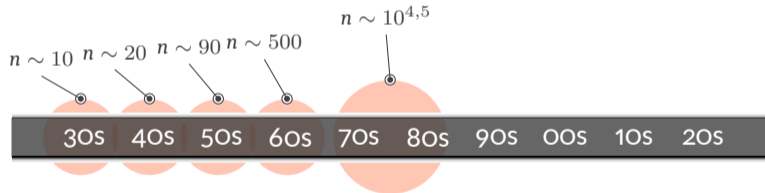
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“The mini-computer cost algorithm is applied to the same complex shell problem used previously, **with 9120 degrees of freedom** [...]. The running times, however, are **40 and 70 hr**, respectively! It would appear that improvement of mini-computer speeds is required...” – Kamel, McCabe (1978)



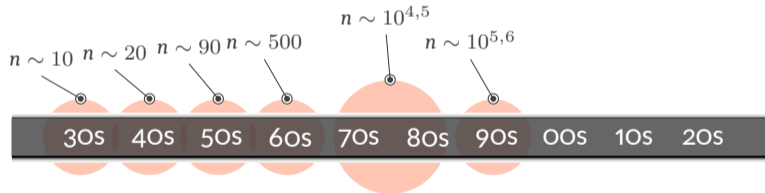
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“For instance, Pomerell in 1994 reports on successful application of **preconditioned Krylov methods** for very ill-conditioned unstructured finite element systems **of order up to 210,000** that arise in semiconductor device modeling.” – Saad Y., van der Vorst, H.A. (2000)



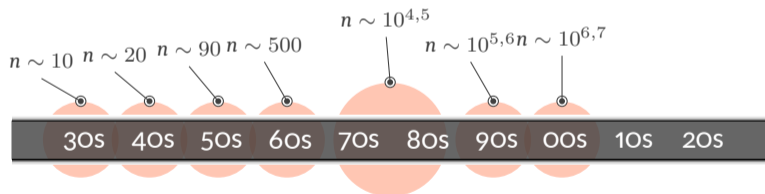
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $A \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“As a second example, we show results (Table VIII) for a problem arising in ocean modeling (barotropic equation) **with $n = 370,000$ unknowns** and approximately 3.3 million nonzero entries...” – Benzi, M. (2002)



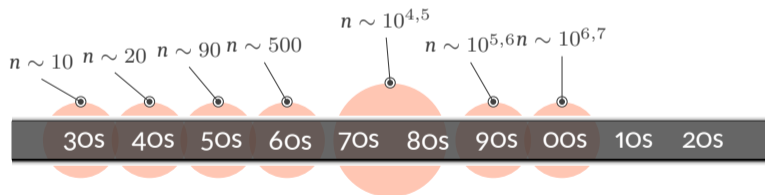
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $A \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(A) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“Problem: **Large**, mesh size: $180 \times 60 \times 30$, **# unknowns (in simulation): 1,010,160**, Solution time 45.7 h” – Wang, de Sturler, Paulino (2006)



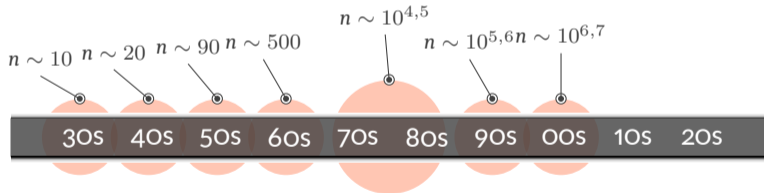
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



“The parallel GMRES was tested on the Tesla T10P GPU using a set of matrix data from the oil field simulation data of Conoco Phillips. The order of the system ranges **from ~ 2000 to ~ 1.1 million.**” – M. Wang, H. Klie, M. Parashar, H. Sudan (2009)



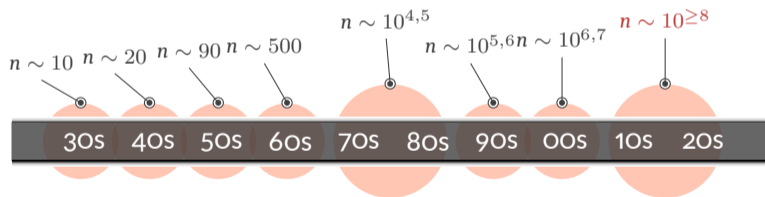
Linear Systems

1 Parallel computing: why?

$$\text{Solve : } \mathbf{Ax} = \mathbf{b},$$

where

- $\mathbf{A} \in \mathbb{R}^{n \times n}$ is a **very large** and **sparse matrix** $\text{nnz}(\mathbf{A}) = O(n)$,
- $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$.



The **exascale** challenge, using computer that do 10^{15} Flops, targeting next-gen systems doing 10^{18} Flops to solve problems with **tens of billions** of unknowns.



Table of Contents

2 Parallel computing: where?

- ▶ Parallel computing: why?
Linear Systems, *mon amour*
- ▶ Parallel computing: where?
Flynn's Taxonomy
Bēowulf
- ▶ Parallel computing: how?
An example of contemporary application
- ▶ First order of business: GIT
- ▶ Exercises



Parallel computers: Flynn's Taxonomy

2 Parallel computing: where?

Let us start from the bottom: the **machines**.



Parallel computers: Flynn's Taxonomy

2 Parallel computing: where?

Let us start from the bottom: the **machines**.

- What is a parallel computer?



Parallel computers: Flynn's Taxonomy

2 Parallel computing: where?

Let us start from the bottom: the **machines**.

- What is a parallel computer? well, it can be a certain number of different “things”
 - Multi-core computing
 - Symmetric multiprocessing
 - Distributed computing
 - Cluster computing
 - Massively parallel computing
 - Grid computing
 - General-purpose computing on graphics processing units (GPGPU)
 - Vector processors



Parallel computers: Flynn's Taxonomy

2 Parallel computing: where?

Let us start from the bottom: the **machines**.

- What is a parallel computer? well, it can be a certain number of different “things”
 - **Multi-core computing**
 - Symmetric multiprocessing
 - Distributed computing
 - **Cluster computing**
 - **Massively parallel computing**
 - Grid computing
 - **General-purpose computing on graphics processing units (GPGPU)**
 - Vector processors

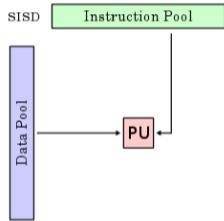


Parallel computers: Flynn's Taxonomy

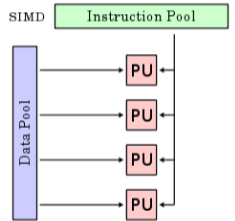
2 Parallel computing: where?

Let us start from the bottom: the **machines**.

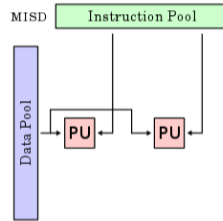
- What is a parallel computer?
- Let us *abstract* from the machine by describing Flynn's taxonomy



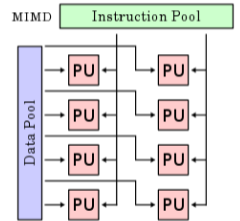
Single instruction
stream, single data
stream
SISD



Single instruction
stream, multiple data
streams
SIMD



Multiple instruction
streams, single data
stream
MISD



Multiple instruction
streams, multiple data
streams
MIMD

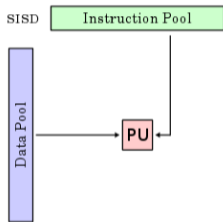


Parallel computers: Flynn's Taxonomy

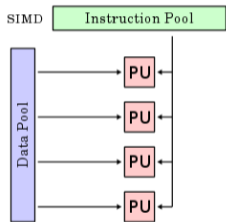
2 Parallel computing: where?

Let us start from the bottom: the **machines**.

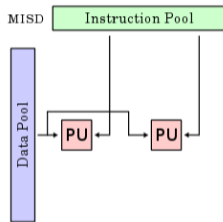
- What is a parallel computer?
- Let us *abstract* from the machine by describing Flynn's taxonomy



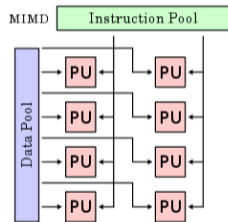
Single instruction
stream, single data
stream
SISD



Single instruction
stream, multiple data
streams
SIMD



Multiple instruction
streams, single data
stream
MISD



Multiple instruction
streams, multiple data
streams
MIMD

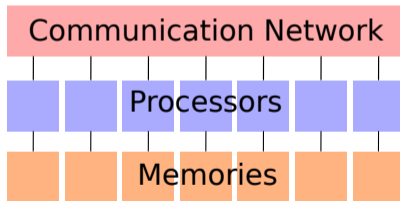


Parallel Computers: our computer model

2 Parallel computing: where?

For our task of introducing parallel computations we need to fix a **specific multiprocessor model**, i.e., a specific generalization of the sequential RAM model in which there is more than one processor.

Since we want to stay in a SIMD/MIMD model, we focus on a *local memory machine model*, i.e., a set of M processors each with its own local memory that are attached to a common communication network.



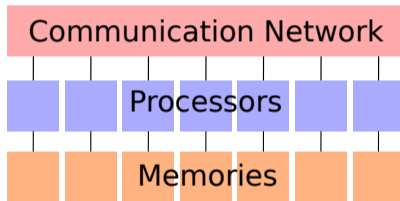


Parallel Computers: our computer model

2 Parallel computing: where?

For our task of introducing parallel computations we need to fix a **specific multiprocessor model**, i.e., a specific generalization of the sequential RAM model in which there is more than one processor.

Since we want to stay in a SIMD/MIMD model, we focus on a *local memory machine model*, i.e., a set of M processors each with its own local memory that are attached to a common communication network.



- We can be more precise about the connection between processors, one can consider a network (a collection of switches connected by communication channels) and delve in a detailed way into its pattern of interconnection, i.e., into what is called the network topology.

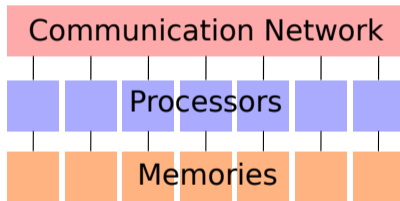


Parallel Computers: our computer model

2 Parallel computing: where?

For our task of introducing parallel computations we need to fix a **specific multiprocessor model**, i.e., a specific generalization of the sequential RAM model in which there is more than one processor.

Since we want to stay in a SIMD/MIMD model, we focus on a *local memory machine model*, i.e., a set of M processors each with its own local memory that are attached to a common communication network.



- An alternative is to summarize the network properties in terms of two parameters: **latency** and **bandwidth**

Latency the time it takes for a message to traverse the network;

Bandwidth the rate at which a processor can inject data into the network.



Parallel computing: where? - <https://www.top500.org/>

2 Parallel computing: where?

“...we have decided in 1993 to assemble and maintain a list of the 500 most powerful computer systems. Our list has been compiled twice a year since June 1993 with the help of high-performance computer experts, computational scientists, manufacturers, and the Internet community in general...

In the present list (which we call the TOP500), we list computers ranked by their performance on the **LINPACK Benchmark.**”

http:

[//www.netlib.org/benchmark/hpl/](http://www.netlib.org/benchmark/hpl/)



Parallel computing: where? - <https://www.top500.org/>

2 Parallel computing: where?

“...we have decided in 1993 to assemble and maintain a list of the 500 most powerful computer systems. Our list has been compiled twice a year since June 1993 with the help of high-performance computer experts, computational scientists, manufacturers, and the Internet community in general...

In the present list (which we call the TOP500), we list computers ranked by their performance on the **LINPACK Benchmark**.”

http:

[//www.netlib.org/benchmark/hpl/](http://www.netlib.org/benchmark/hpl/)

The **LINPACK** Benchmark.

Solution of a dense $n \times n$ system of linear equations $\mathbf{Ax} = \mathbf{b}$, so that

- $\frac{\|\mathbf{Ax} - \mathbf{b}\|}{\|\mathbf{A}\| \|\mathbf{x}\| n \epsilon} \leq O(1)$, for ϵ machine precision,
- It uses a specialized right-looking LU factorization with look-ahead



Parallel computing: where? - <https://www.top500.org/>

2 Parallel computing: where?

“...we have decided in 1993 to assemble and maintain a list of the 500 most powerful computer systems. Our list has been compiled twice a year since June 1993 with the help of high-performance computer experts, computational scientists, manufacturers, and the Internet community in general...

In the present list (which we call the TOP500), we list computers ranked by their performance on the **LINPACK Benchmark**.”

[http:](http://www.netlib.org/benchmark/hpl/)

[//www.netlib.org/benchmark/hpl/](http://www.netlib.org/benchmark/hpl/)

The **LINPACK** Benchmark.






Solution of a dense $n \times n$ system of linear equations $A\mathbf{x} = \mathbf{b}$, so that

- Measuring
 - R_{\max} the performance in GFLOPS for the largest problem run on a machine,
 - N_{\max} the size of the largest problem run on a machine,
 - $N_{1/2}$ the size where half the R_{\max} execution rate is achieved,
 - R_{peak} the theoretical peak performance GFLOPS for the machine.

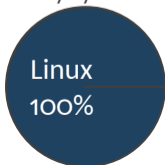


The TOP500 List

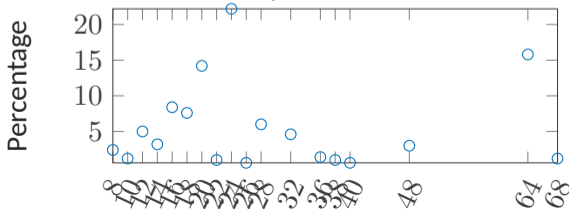
2 Parallel computing: where?

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)	
1	Frontier	8,730,112	1,102.00	1,685.65	21,100	
2	Supercomputer Fugaku	7,630,848	442.01	537.21	29,899	
3	LUMI	2,220,288	309.10	428.70	6,016	
4	Leonardo	1,463,616	174.70	255.75	5,610	
5	Summit	2,414,592	148.60	200.79	10,096	

OS Family System Share



Cores per Socket





The machines we have in the department

2 Parallel computing: where?

The **Toeplitz Cluster** made of **5 nodes**:

- 4 Nodes Intel® Xeon® CPU E5-2650 v4 @ 2.20GHz with 2 threads per core, 12 cores per socket and 2 socket with 256 GB;
- 1 Node Intel® Xeon® CPU E5-2643 v4 @ 3.40GHz with 2 threads per core, 6 cores per socket and 2 socket with 128 GB.



The machines we have in the department

2 Parallel computing: where?

The **Toeplitz Cluster** made of **5 nodes**:

- 4 Nodes Intel® Xeon® CPU E5-2650 v4 @ 2.20GHz with 2 threads per core, 12 cores per socket and 2 socket with 256 GB;
- 1 Node Intel® Xeon® CPU E5-2643 v4 @ 3.40GHz with 2 threads per core, 6 cores per socket and 2 socket with 128 GB.

A **new machine** we are buying with the “*Dipartimento di Eccellenza*” project:





The machines we have in the department

2 Parallel computing: where?

The **Toeplitz Cluster** made of **5 nodes**:

- 4 Nodes Intel® Xeon® CPU E5-2650 v4 @ 2.20GHz with 2 threads per core, 12 cores per socket and 2 socket with 256 GB;
- 1 Node Intel® Xeon® CPU E5-2643 v4 @ 3.40GHz with 2 threads per core, 6 cores per socket and 2 socket with 128 GB.

A **new machine** we are buying with the “*Dipartimento di Eccellenza*” project:



The machine we will build here!



Bēowulf

2 Parallel computing: where?

HWÆT: WE GAR-DENA IN GEARDAGUM
þeodcyninga þrym gefrunon.
Hu ða æþelingas ellen fremedon!
Oft Scyld Scefing sceaþena þreatum
monegum mægþum meodosetla ofteah,
egsode eorl, syððan ærest wearð
feasceaft funden. He þæs frofre gebad,
weox under wolcnum, weorðmyndum þah,
oð þæt him æghwylc þara ymb sittendra
ofer hronrade hyran scolde,
gomban gyldan. þæt wæs god cyning.





Bēowulf

2 Parallel computing: where?

“Bēowulf is a **multi-computer architecture** which can be used for parallel computations. It is a system which usually consists of **one server node**, and **one or more client nodes connected via Ethernet** or some other network. It is a system built using **commodity hardware components**, like any PC capable of running a Unix-like operating system, with standard Ethernet adapters, and switches.”

Radajewski, Radajewski; Eadline, Douglas
(22 November 1998).
“Beowulf HOWTO”. ibiblio.org. v1.1.1.





Table of Contents

3 Parallel computing: how?

- ▶ Parallel computing: why?
Linear Systems, *mon amour*
- ▶ Parallel computing: where?
Flynn's Taxonomy
Bēowulf
- ▶ **Parallel computing: how?**
An example of contemporary application
- ▶ First order of business: GIT
- ▶ Exercises



Parallel Algorithms

3 Parallel computing: how?

In a fairly general way we can say that a **parallel algorithm** is an algorithm which can do *multiple operations* in a given time.



Parallel Algorithms

3 Parallel computing: how?

In a fairly general way we can say that a **parallel algorithm** is an algorithm which can do *multiple operations* in a given time.

Example: the sum of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

$$\begin{aligned}\mathbf{x} &= [x_1 \ x_2 \ \cdots \ x_i \ x_{i+1} \ \cdots \ x_n] \\ + \\ \mathbf{y} &= [y_1 \ y_2 \ \cdots \ y_i \ y_{i+1} \ \cdots \ y_n] \\ = \\ \mathbf{x} + \mathbf{y} &= [x_1 + y_1 \ x_2 + y_2 \ \cdots \ x_i + y_i \ \cdots \ x_n + y_n]\end{aligned}$$

- If we do the operation sequentially we do $O(n)$ operations in T_n



Parallel Algorithms

3 Parallel computing: how?

In a fairly general way we can say that a **parallel algorithm** is an algorithm which can do *multiple operations* in a given time.

Example: the sum of two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

$$\begin{aligned}\mathbf{x} &= [x_1 \ x_2 \ \cdots \ x_i \mid x_{i+1} \ \cdots \ x_n] \\ + \\ \mathbf{y} &= [y_1 \ y_2 \ \cdots \ y_i \mid y_{i+1} \ \cdots \ y_n] \\ = \\ \mathbf{x} + \mathbf{y} &= [x_1 + y_1 \ x_2 + y_2 \ \cdots \ x_i + y_i \mid \cdots \ x_n + y_n]\end{aligned}$$

- If we do the operation sequentially we do $O(n)$ operations in T_n
- If we split the operation among 2 processors, one summing up the entries between $1, \dots, i$, and one summing up the entries between $i + 1, \dots, n$ we take T_i time for the first part and T_{n-i} time for the second, therefore the overall time is

17/37 $\max(T_i, T_{n-i})$ for doing always $O(n)$ operations.



Parallel Algorithms: *speedup*

3 Parallel computing: how?

Let us think again abstractly and quantify the **overall speed gain** for a given gain in a subset of a process.

- We break a process into N distinct portions with the i th portion occupying the P_i fraction of the overall completion time,



Parallel Algorithms: *speedup*

3 Parallel computing: how?

Let us think again abstractly and quantify the **overall speed gain** for a given gain in a subset of a process.

- We break a process into N distinct portions with the i th portion occupying the P_i fraction of the overall completion time,
- order the portions in such a way that the N th portion subsumes all the parts of the overall processes with fixed costs.



Parallel Algorithms: *speedup*

3 Parallel computing: how?

Let us think again abstractly and quantify the **overall speed gain** for a given gain in a subset of a process.

- We break a process into N distinct portions with the i th portion occupying the P_i fraction of the overall completion time,
- order the portions in such a way that the N th portion subsumes all the parts of the overall processes with fixed costs.
- The *speedup* of the i th portion can then be defined as

$$S_i \triangleq \frac{t_{\text{original}}}{t_{\text{optimized}}}, \quad i = 1, \dots, N - 1$$

where the numerator and denominator are the original and optimized completion time.



Parallel Algorithms: *speedup*

3 Parallel computing: how?

Let us think again abstractly and quantify the **overall speed gain** for a given gain in a subset of a process.

Amdahl's Law

Then the **overall speedup** for $\mathbf{P} = (P_1, \dots, P_N)$, $\mathbf{S} = (S_1, \dots, S_{N-1})$ is:

$$S(\mathbf{P}, \mathbf{S}) = \left(P_N + \sum_{i=1}^{N-1} \frac{P_i}{S_i} \right)^{-1} .$$



Parallel Algorithms: *Amdahl's Law*

3 Parallel computing: how?

Let us make some observations on Amdahl's Law

- We are not assuming about whether the original completion time involves some optimization,
- We are not making any assumption on what our optimization process is,
- We are not even saying that the process in question involves a computer!

Amdahl's Law is a fairly general way of looking at how processes can be speed up by dividing them into sub-tasks with lower execution time.



Parallel Algorithms: Amdahl's Law

3 Parallel computing: how?

Let us make some observations on Amdahl's Law

- We are not assuming about whether the original completion time involves some optimization,
- We are not making any assumption on what our optimization process is,
- We are not even saying that the process in question involves a computer!

Amdahl's Law is a fairly general way of looking at how processes can be speed up by dividing them into sub-tasks with lower execution time.

Moreover, it fixes the **theoretical maximum speedup** in various scenarios.

- If we allow all components S_i to grow unbounded then the upper bound on all scenario is $S_{\max} = 1/P_N$.

Let us decline it in the context of the potential utility of *parallel hardware*.



Parallel Algorithms: Amdahl's Law for parallel hardware

3 Parallel computing: how?

Consider now having a parallel machine that permits us dividing the execution of code across M hardware units, then the problem independent maximum speedup that such hardware can provide is M .

Parallel Efficiency

We define the parallel efficiency E as

$$E \triangleq \frac{S_{\text{overall}}}{M},$$

where $E = 100\%$ correspond to the maximal use of the available hardware. When $S_{\text{max}} < M$, it is then impossible to take full advantage of all available execution units.



Parallel Algorithms: Amdahl's Law for parallel hardware

3 Parallel computing: how?

Consider now having a parallel machine that permits us dividing the execution of code across M hardware units, then the problem independent maximum speedup that such hardware can provide is M .

Parallel Efficiency

We define the parallel efficiency E as

$$E \triangleq \frac{S_{\text{overall}}}{M},$$

where $E = 100\%$ correspond to the maximal use of the available hardware. When $S_{\text{max}} < M$, it is then impossible to take full advantage of all available execution units.

Goal: we require very large S_{max} and correspondingly tiny P_N .



Parallel Algorithms: Amdahl's Law for parallel hardware

3 Parallel computing: how?

Consider now having a parallel machine that permits us dividing the execution of code across M hardware units, then the problem independent maximum speedup that such hardware can provide is M .

Parallel Efficiency

We define the parallel efficiency E as

$$E \triangleq \frac{S_{\text{overall}}}{M},$$

where $E = 100\%$ correspond to the maximal use of the available hardware. When $S_{\text{max}} < M$, it is then impossible to take full advantage of all available execution units.

Goal: we require very large S_{max} and correspondingly tiny P_N .

Every dusty corner of a code must scale, any portion that doesn't becomes the rate-limiting step!



Parallel Algorithms: *Amdahl's Law* limitations

3 Parallel computing: how?

What we are neglecting and what we are tacitly assuming

- We are neglecting *overhead costs*, i.e., the cost associated with parallel execution such as
 - initializing (spawning) and joining of different computation threads,
 - communication between processes, data movement and memory allocation.
- We considered also the ideal case in which $S_i \rightarrow +\infty \forall i$, observe that with finite speedup on portions 1 through $N - 1$, the S_{overall} might continue to improve with increasing number of execution units.
- We are assuming that the size of the problem remains fixed while the number of execution units increases, this is called the case of **strong scalability**. In some contexts, we need to turn instead to **weak scalability** in which the problem size grows proportionally to the number of execution units.



Gustafson's law

3 Parallel computing: how?

In the **weak scalability** case the right framework is to use **Gustafson's law**

Gustafson's law

$$S = s + p \times N = s + (1 - s) \times N = N + (1 - N) \times s$$

where

- S is the theoretical speedup of the program with parallelism (**scaled speedup**),
- N is the number of computing units,
- s and p are the fractions of time spent executing the serial parts and the parallel parts of the program on the parallel system, i.e., $s + p = 1$.



Gustafson's law

3 Parallel computing: how?

In the **weak scalability** case the right framework is to use **Gustafson's law**

Gustafson's law

$$S = s + p \times N = s + (1 - s) \times N = N + (1 - N) \times s$$

where

- S is the theoretical speedup of the program with parallelism (**scaled speedup**),
- N is the number of computing units,
- s and p are the fractions of time spent executing the serial parts and the parallel parts of the program on the parallel system, i.e., $s + p = 1$.

“Solving a **larger problem** in the **same amount of time** should be possible by using **more computing units**”



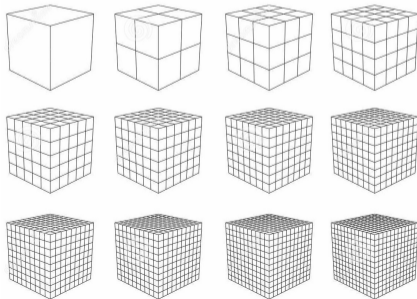
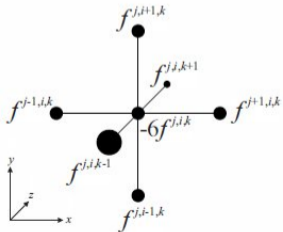
Test Case

3 Parallel computing: how?

Poisson equation

$-\Delta f = 1$ on unit cube, with Dirichlet Boundary Conditions

- 7-point finite-difference discretization
- cartesian grid with uniform refinement along the coordinates for increasing mesh size





Test Case

3 Parallel computing: how?

Solver/preconditioner settings

- AMG as preconditioner of Flexible CG, stopped when $\|\mathbf{r}^k\|_2/\|\mathbf{b}\|_2 \leq 10^{-6}$, or itmax = 500
 - KCMATCH K-cycle with 2 inner iterations, CMATCH building aggregates of max size 8, unsmoothed prolongators
 - VSCMATCH V-cycle, CMATCH building aggregates of max size 8, smoothed prolongators
 - VSDVB V-cycle for decoupled classic smoothed aggregation
- 1 sweep of forward/backward Hybrid Gauss-Seidel smoother, parallel CG preconditioned with Block-Jacobi and ILU(o) at the coarsest level
- coarsest matrix size $n_c \leq 200np$, with np number of cores

An example from: P. D'Ambra, F. Durastante, and S. Filippone, "AMG preconditioners for linear solvers towards extreme scale", *SIAM J. Sci. Comput.* (2021).



Experimental environment & Comparison

3 Parallel computing: how?

Piz Daint - Swiss National Supercomputing Center by PRACE

- Cray Model XC40/Cray XC50 architecture with 5704 hybrid compute nodes (Intel Xeon E5-2690 v3 with Nvidia Tesla P100 accelerator)
- Cray Aries routing and communications ASIC with Dragonfly network topology
- GNU compiler rel. 8, Cray MPI 7, Cray-libsci 20.09.1
- PSBLAS 3.7, AMG4PSBLAS 1.0 (See: psctoolkit.github.io)





Experimental environment & Comparison

3 Parallel computing: how?

Hypre: Scalable Linear Solvers and Multigrid Methods by LLNL

- **BoomerAMG** as preconditioner of **CG**, stopped when $\|\mathbf{r}^k\|_2 / \|\mathbf{b}\|_2 \leq 10^{-6}$, or $\text{itmax} = 500$
- **V-cycle** with 1 sweep of forward/backward Hybrid Gauss-Seidel smoother, LU factorization at the coarsest level
- **3 coarsening schemes**: hybrid RS/CLJP (**Flg**), Hybrid Maximal Independent Set (**HMIS**), HMIS with first level of aggressive coarsening (**HMIS1**); default parameters for coarsest matrix size $1 \leq n_c \leq 9$, coupled with modified (long-range) classical interpolation



Weak scaling (256K dofs \times core): Iteration number

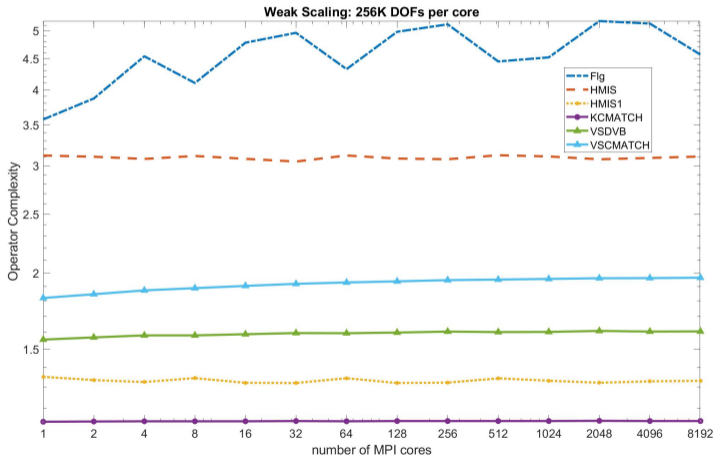
3 Parallel computing: how?

np	$n/10^6$	AMG4PSBLAS			Hypre		
		KCMATCH	VSCMATCH	VSDVB	Flg	HMIS	HMIS1
1	0.256	12	7	11	6	6	12
2	0.512	12	7	12	7	9	15
2^2	1.036	12	7	13	7	12	17
2^3	2.048	12	7	14	8	13	17
2^4	4.075	12	8	14	8	14	20
2^5	8.049	13	9	15	8	14	20
2^6	16.384	12	8	15	9	16	22
2^7	32.604	12	8	15	10	18	25
2^8	63.917	13	9	16	10	20	27
2^9	131,072	14	8	18	11	22	29
2^{10}	256,000	15	8	17	12	25	32
2^{11}	511,335	16	12	21	13	29	37
2^{12}	1024,192	15	8	26	13	35	40
2^{13}	2097,152	16	9	27	14	37	44



Weak scaling (256K dofs \times core): operator complexity

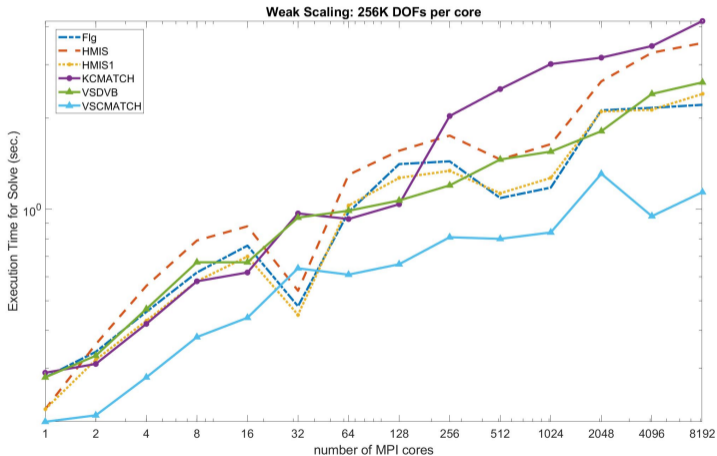
3 Parallel computing: how?





Weak scaling (256K dofs \times core): solve time

3 Parallel computing: how?





Results at extreme scale: MPI vs hybrid MPI-CUDA

3 Parallel computing: how?

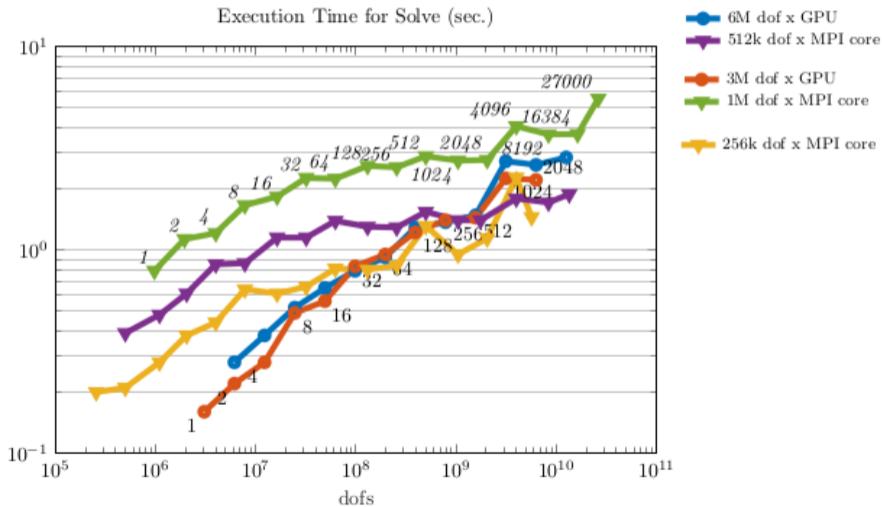




Table of Contents

4 First order of business: GIT

- ▶ Parallel computing: why?
Linear Systems, *mon amour*
- ▶ Parallel computing: where?
Flynn's Taxonomy
Bēowulf
- ▶ Parallel computing: how?
An example of contemporary application
- ▶ **First order of business: GIT**
- ▶ Exercises



Software Version Control: GIT

4 First order of business: GIT

In **software engineering**, version control is a class of systems responsible for managing changes to **computer programs, documents**, large web sites, or other *collections of information*. Version control is a component of software configuration management.




- We are going to use GIT: <https://git-scm.com/>,
- Specifically, the Gitea instance run by the PHC: <https://git.phc.dm.unipi.it/>.



Getting an up-and-running GIT account


4 First order of business: GIT

1. Go to: <https://git.phc.dm.unipi.it/>,
2. Click on: [← **Accedi** (top right of the screen),
3. Then: **Accedi con** ,
4. Use **UNIPD credentials** to login.



Getting an up-and-running GIT account

4 First order of business: GIT

1. Go to: <https://git.phc.dm.unipi.it/>,
2. Click on: [← **Accedi** (top right of the screen),
3. Then: **Accedi con** ,
4. Use **UNIFI credentials** to login.

Create an **SSH key**:

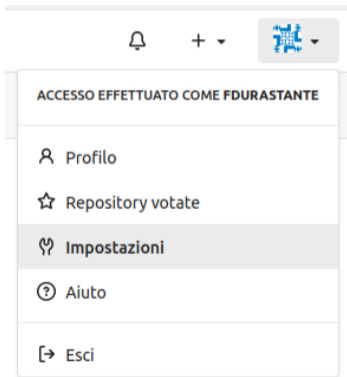
1. Open a terminal (CTRL+ALT+T),
2. Write: `ssh-keygen -t ed25519 -C 'fabio.durastante@unipi.it` (use your own E-mail address!)
3. Press ENTER to confirm **default file location** (`~/ .ssh`),
4. At the prompt, type a secure passphrase (you have to remember it!),
5. Run: `eval "$ (ssh-agent -s) "` and then `ssh-add ~/ .ssh/id_ed25519`.



Getting an up-and-running GIT account

4 First order of business: GIT

From the settings menu you have access to the configurations of the Git service.



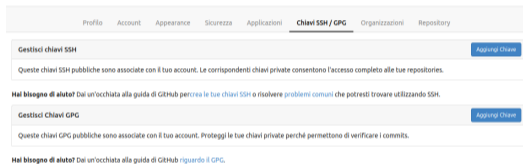
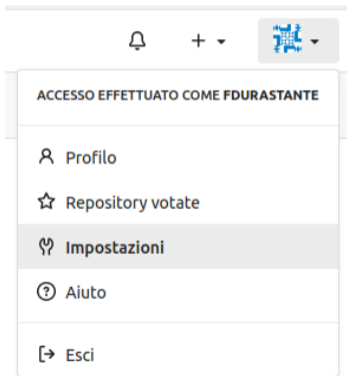


Getting an up-and-running GIT account

4 First order of business: GIT

From the settings menu you have access to the configurations of the Git service.

- SSH key entry:

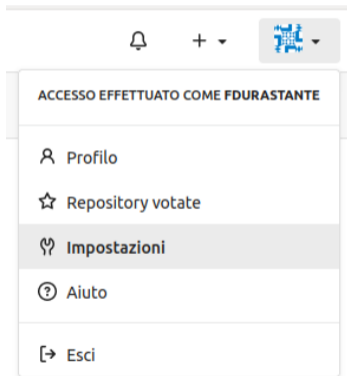




Getting an up-and-running GIT account

4 First order of business: GIT

From the settings menu you have access to the configurations of the Git service.



- SSH key entry

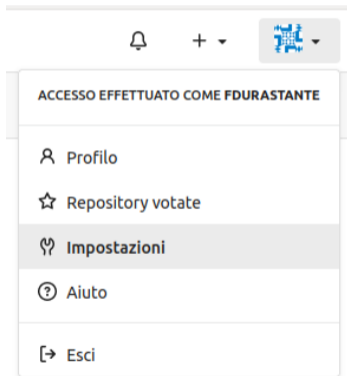




Getting an up-and-running GIT account

4 First order of business: GIT

From the settings menu you have access to the configurations of the Git service.



- SSH key entry

Chiavi SSH / GPG

—

— Which inserts similarly:

Nome della Chiave

Chiave

Contenuto

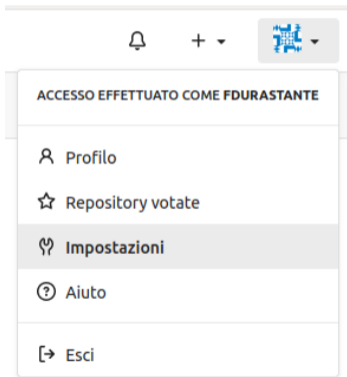
Inizia con 'ssh-ed25519', 'ssh-rsa', 'ecdsa-sha2-nistp256', ed25519@openssh.com'



Getting an up-and-running GIT account

4 First order of business: GIT

From the settings menu you have access to the configurations of the Git service.



- SSH key entry

Chiavi SSH / GPG

— Which inserts similarly:

Nome della Chiave

Contenuto

```
Inizia con 'ssh-ed25519', 'ssh-rsa', 'ecdsa-sha2-nistp256',  
ed25519@openssh.com'
```

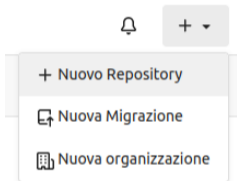
— Concluding with:

Aggiungi Chiave



A repository

4 First order of business: GIT

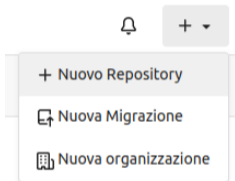


- You can **create a new repository** easily.



A repository


4 First order of business: GIT



- You can **create a new repository** easily.

Nuovo Repository

Un repository contiene tutti i file del progetto, inclusa la cronologia delle revisioni. Lo hai già altrove? [Migrare il repository.](#)

Proprietario *  fdurastante

Alcune organizzazioni potrebbero non essere visualizzate nel menu a discesa a causa di un limite massimo al numero di repository.

Nome Repository *

Un buon nome per un repository è costituito da parole chiave corte, facili da ricordare e uniche.

Visibilità **Rendi privato il repository**
Solo il proprietario o i membri dell'organizzazione se hanno diritti, saranno in grado di vederlo.

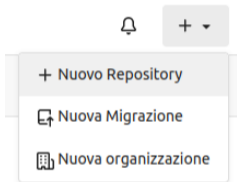
Descrizione

Inserisci una breve descrizione (opzionale)



A repository


4 First order of business: GIT



- You can **create a new repository** easily.
- And then: 

Nuovo Repository

Un repository contiene tutti i file del progetto, inclusa la cronologia delle revisioni. Lo hai già altrove? [Migrare il repository.](#)

Proprietario *  fdurastante

Alcune organizzazioni potrebbero non essere visualizzate nel menu a discesa a causa di un limite massimo al numero di repository.

Nome Repository *

Un buon nome per un repository è costituito da parole chiave corte, facili da ricordare e uniche.

Visibilità **Rendi privato il repository**
Solo il proprietario o i membri dell'organizzazione se hanno diritti, saranno in grado di vederlo.

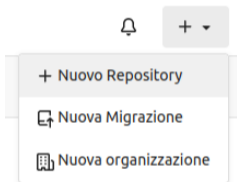
Descrizione


Inserisci una breve descrizione (opzionale)



A repository


4 First order of business: GIT



- You can **create a new repository** easily.
- And then: 

Nuovo Repository

Un repository contiene tutti i file del progetto, inclusa la cronologia delle revisioni. Lo hai già altrove? [Migrare il repository.](#)

Proprietario *  fdurastante

Alcune organizzazioni potrebbero non essere visualizzate nel menu a discesa a causa di un limite massimo al numero di repository.

Nome Repository *

Un buon nome per un repository è costituito da parole chiave corte, facili da ricordare e uniche.

Visibilità **Rendi privato il repository**
Solo il proprietario o i membri dell'organizzazione se hanno diritti, saranno in grado di vederlo.

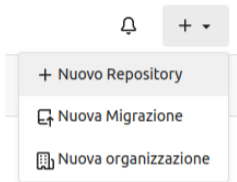
Descrizione

Inserisci una breve descrizione (opzionale)



A repository

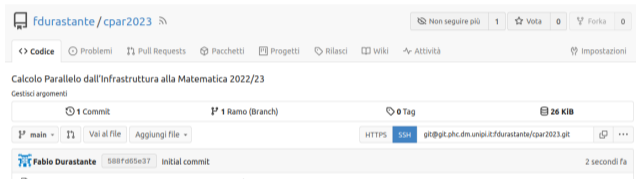
4 First order of business: GIT



- You can **create a new repository** easily.
- And then: 

```
git clone git@git.phc.dm.unipi.it:fdurastante/cpar2023.git
cd cpar2023
```

The folder will contain these slides, and – in the future – the other material we will use.

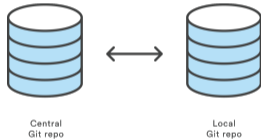




GIT Workflow

4 First order of business: GIT

We will use GIT to *exchange files* and working on *writing code*.



The **repository** is where files' current and historical data are stored, often on a server.

- checkout** To *check out* is to create a local working copy from the repository,
- pull, push** Copy revisions from one repository into another. Pull is initiated by the receiving repository, while push is initiated by the source.
- commit** To *commit* is to **write** or **merge** the **changes made in the working copy back to the repository**. A **commit** contains **metadata**, typically the **author information** and a **commit message** that describes the change.
- merge** is an operation in which two sets of changes are applied to a file or set of files.



Table of Contents

5 Exercises

- ▶ Parallel computing: why?
Linear Systems, *mon amour*
- ▶ Parallel computing: where?
Flynn's Taxonomy
Bēowulf
- ▶ Parallel computing: how?
An example of contemporary application
- ▶ First order of business: GIT
- ▶ Exercises



Exercises

5 Exercises

- Review the slides at least once to get used to the vocabulary.
- Make working in a Linux shell comfortable, e.g., `ls`, `cd`, `ssh`, `mkdir`, `mv`, `cp`, `rm`, `grep`, `diff`;
- Follow this GIT tutorial to familiarize yourself with the commands and workflow:
<https://git-scm.com/docs/gittutorial>
- Propose a name for our Beowulf machine:
<https://forms.gle/F9XtvAWmVqv6jD5f7>
- Have fun.



Calcolo Parallelo dall'Infrastruttura alla Matematica *Thank you for listening!*

Any questions?