High-Performance Mathematics

Parallel computing intro & auxiliary tools

Progetto Speciale per la Didattica 2023/24

Fabio Durastante (L1)

April 10, 2024

Dipartimento
di Matematica
Università di Pisa

Table of Contents

1 Parallel computing: where?

Parallel computers: Flynn’s Taxonomy

Let us start from the bottom: the machines.

Parallel computers: Flynn's Taxonomy

Let us start from the bottom: the machines.

- What is a parallel computer?

Parallel computers: Flynn’s Taxonomy

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

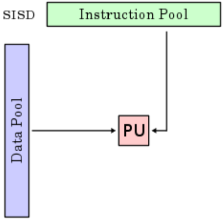
Let us start from the bottom: the machines.

- 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

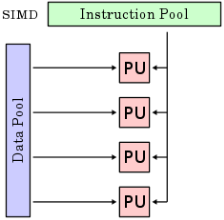
Parallel computers: Flynn’s Taxonomy

Let us start from the bottom: the machines.

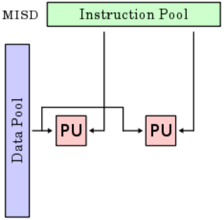
- What is a parallel computer?
- Let us *abstract* from the machine by describing Flynn’s taxonomy



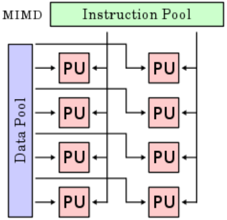
Single instruction stream, single data stream
SISD



Single instruction stream, multiple data streams
SIMD



Multiple instruction streams, single data stream
MISD

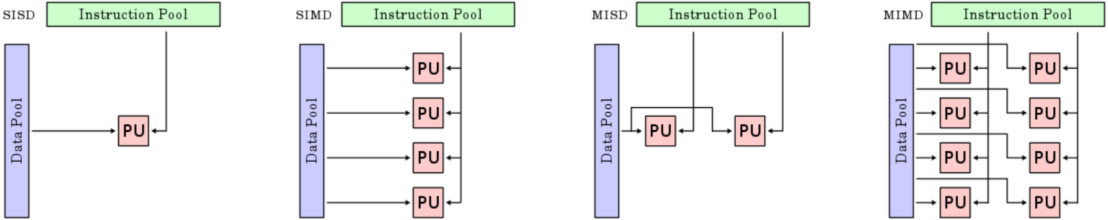


Multiple instruction streams, multiple data streams
MIMD

Parallel computers: Flynn's Taxonomy

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: the SIMD model

A parallel programming model where a **single instruction** is **executed simultaneously** on **multiple data points**.

- SIMD architectures exploit data-level parallelism.
- Single instruction stream controls multiple processing elements.

Parallel Computers: the SIMD model

A parallel programming model where a **single instruction** is **executed simultaneously** on **multiple data points**.

- SIMD architectures exploit data-level parallelism.
- They operate on multiple data elements in parallel using a single instruction stream.
- Single instruction stream controls multiple processing elements.
- Each processing element operates on a different data element.

Parallel Computers: the SIMD model

A parallel programming model where a **single instruction** is **executed simultaneously** on **multiple data points**.

- SIMD architectures exploit data-level parallelism.
- They operate on multiple data elements in parallel using a single instruction stream.
- SIMD instructions perform the same operation on multiple data elements at the same time.
- Single instruction stream controls multiple processing elements.
- Each processing element operates on a different data element.

Parallel Computers: the SIMD model

A parallel programming model where a **single instruction** is **executed simultaneously** on **multiple data points**.

- SIMD architectures exploit data-level parallelism.
- They operate on multiple data elements in parallel using a single instruction stream.
- SIMD instructions perform the same operation on multiple data elements at the same time.
- Single instruction stream controls multiple processing elements.
- Each processing element operates on a different data element.
- SIMD operations are highly efficient for certain types of computations.

Parallel Computers: the SIMD model

A parallel programming model where a **single instruction** is **executed simultaneously** on **multiple data points**.

- SIMD architectures exploit data-level parallelism.
- They operate on multiple data elements in parallel using a single instruction stream.
- SIMD instructions perform the same operation on multiple data elements at the same time.
- Single instruction stream controls multiple processing elements.
- Each processing element operates on a different data element.
- SIMD operations are highly efficient for certain types of computations.

Key Idea

Parallelism is achieved by how many different data a single operation can act on.

CPUs with many cores

Modern CPUs are **increasingly equipped with multiple cores**, enabling parallel processing of tasks.

- Historically, CPUs contained a single core, capable of executing one instruction at a time.

CPUs with many cores

Modern CPUs are **increasingly equipped with multiple cores**, enabling parallel processing of tasks.

- Historically, CPUs contained a single core, capable of executing one instruction at a time.
- **CPUs with multiple cores have become prevalent** due to the need for increased computational power and parallel processing capabilities.

CPUs with many cores

Modern CPUs are **increasingly equipped with multiple cores**, enabling parallel processing of tasks.

- Historically, CPUs contained a single core, capable of executing one instruction at a time.
- **CPUs with multiple cores have become prevalent** due to the need for increased computational power and parallel processing capabilities.

- *Samsung Galaxy S24 Ultra:* 8-core (1x3.39GHz Cortex-X4 & 3x3.1GHz Cortex-A720 & 2x2.9GHz Cortex-A720 & 2x2.2GHz Cortex-A520),

CPUs with many cores

Modern CPUs are **increasingly equipped with multiple cores**, enabling parallel processing of tasks.

- Historically, CPUs contained a single core, capable of executing one instruction at a time.
- **CPUs with multiple cores have become prevalent** due to the need for increased computational power and parallel processing capabilities.

- *Samsung Galaxy S24 Ultra:* 8-core,
- *iPhone 15 Pro:* Hexa-core, 3.78 GHz,

CPUs with many cores

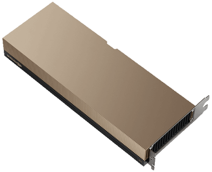
Modern CPUs are **increasingly equipped with multiple cores**, enabling parallel processing of tasks.

- Historically, CPUs contained a single core, capable of executing one instruction at a time.
- **CPUs with multiple cores have become prevalent** due to the need for increased computational power and parallel processing capabilities.
- CPUs contain *multiple independent* processing units on a single chip.
- Each core can execute its own set of instructions independently of other cores.
- Cores typically **share resources** such as **cache** and **memory access**, but they **can execute different tasks simultaneously**.

General Purpose GPUs

General Purpose GPU Computing (GPGPU) leverages the SIMD model for parallel processing using Graphics Processing Units (GPUs).

- GPUs are designed with **thousands of small, efficient cores optimized for parallel computation**.
- Originally developed for graphics rendering, GPUs are now used for general-purpose computation due to their high parallelism.
- GPGPU extends the SIMD model beyond traditional CPU architectures, enabling massively parallel processing.

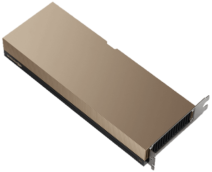


The NVIDIA H100 GPU features **640 Tensor Cores** and **128 RT Cores**, providing high-speed processing of complex data sets. It also features 80 Streaming Multiprocessors (SMs) and **18,432 CUDA cores**.

General Purpose GPUs

General Purpose GPU Computing (GPGPU) leverages the SIMD model for parallel processing using Graphics Processing Units (GPUs).

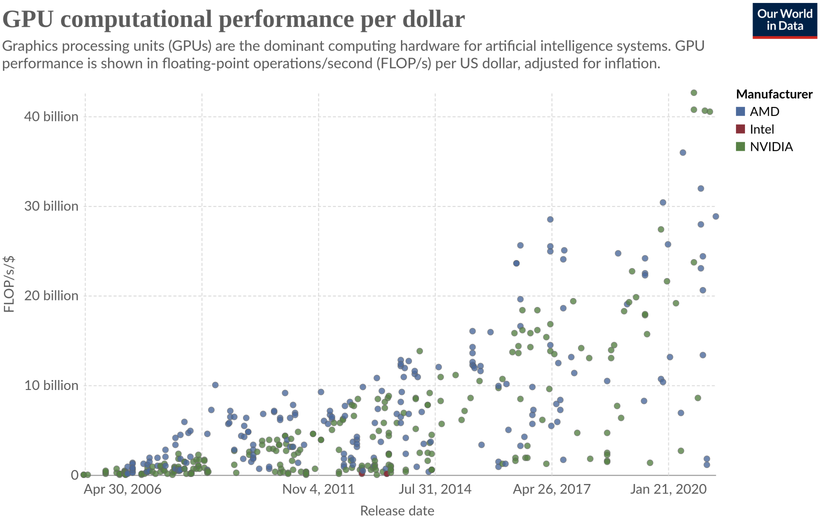
- GPUs are designed with **thousands of small, efficient cores optimized for parallel computation**.
- Originally developed for graphics rendering, GPUs are now used for general-purpose computation due to their high parallelism.
- GPGPU extends the SIMD model beyond traditional CPU architectures, enabling massively parallel processing.
- Each GPU core processes a different portion of the data, achieving parallelism.



The NVIDIA H100 GPU features **640 Tensor Cores** and **128 RT Cores**, providing high-speed processing of complex data sets. It also features 80 Streaming Multiprocessors (SMs) and **18,432 CUDA cores**.

GPU computational performance per dollar

1 Parallel computing: where?



Application-Specific Integrated Circuit

Introduction

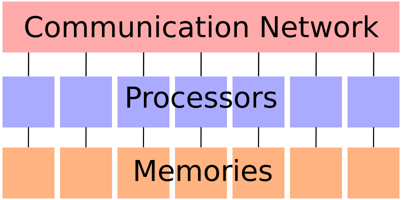
A Tensor Processing Unit (TPU) is a custom-built ASIC (Application-Specific Integrated Circuit) developed by Google specifically for accelerating machine learning workloads.

- TPUs are designed to handle the computational demands of **training** and **executing machine learning models** efficiently.
- TPUs excel at processing **tensor operations**, hence the name "Tensor Processing Unit".
- TPUs are designed to be energy-efficient, i.e., more computations per watt compared to traditional CPUs or GPUs.

Parallel Computers: the MIMD model

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.

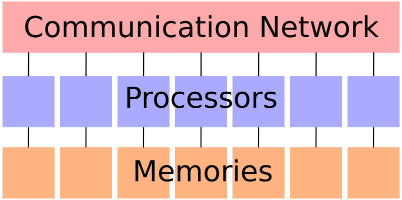
Since we want to stay in a SIMD/MIMD model, we focus on a *local memory machine model*, i.e., a set of M processors each with its own local memory that are attached to a common communication network.



Parallel Computers: the MIMD model

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.

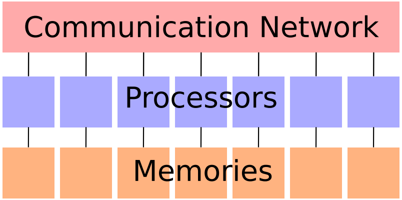


- We can be more precise about the connection between processors, one can consider a network (a collection of switches connected by communication channels) and delve in a detailed way into its pattern of interconnection, i.e., into what is called the network topology.

Parallel Computers: the MIMD model

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

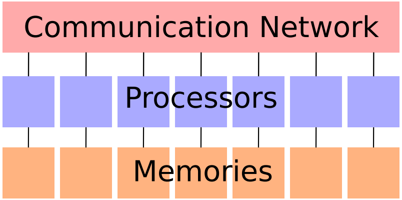
Latency the time it takes for a message to traverse the network;

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

Parallel Computers: the MIMD model

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.



Key Idea

Parallelism is achieved by receiving data which I don't have, and sending data which I have.

"One, two, three, four
Can I have a little more?
Five, six, seven, eight, nine, ten"

Modern machine are now made of:

- many **distributed systems** interconnected by a **fast network**,
- every system has **one** or **more multi-core processors**,
- different type of **accelerators** (GPUs, ASICs, FPGAs. . .) on every computing node.

All Together Now

“One, two, three, four
Can I have a little more?
Five, six, seven, eight, nine, ten”

Modern machine are now made of:

- many **distributed systems** interconnected by a **fast network**,
- every system has **one** or **more multi-core processors**,
- different type of **accelerators** (GPUs, ASICs, FPGAs. . .) on every computing node.

We need to look for **algorithms** and **programming models** to **fully exploit** all these **resources**!

Table of Contents

2 Parallel computing: where?

Parallel computing: where? – `https://www.top500.org/`

"…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."
www.netlib.org/benchmark/hpl

"...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."
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

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

"...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."
www.netlib.org/benchmark/hpl

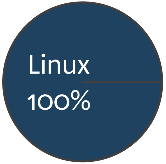
The LINPACK Benchmark.
Solution of a dense $n \times n$ system of linear equations $A\mathbf{x} = \mathbf{b}$, so that

- 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

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	Frontier	8,699,904	1,194.00	1,679.82	22,703
2	Aurora	4,742,808	585.34	1,059.33	24,687
3	Eagle	1,123,200	561.20	846.84	–
4	Supercomputer Fugaku	7,630,848	442.01	537.21	29,899
5	LUMI	2,752,704	379.70	531.51	7,107
6	Leonardo	1,824,768	238.70	255.75	7,404

OS Family System Share



Cores per Socket

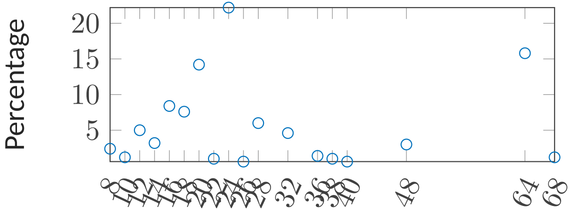


Table of Contents

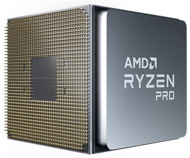
3 The tools at our disposal

The machines we have in the department

The machines of **Aula DM3** and **Aula DM4**.

- AMD Ryzen 5 PRO 5650G @ 4.4 GHz with Radeon Graphics, 1 socket with 6 cores per socket and 2 threads per core,
- 32 GB RAM,
- 1 NVIDIA T1000 8GB GDDR6.

- AMD Ryzen 5 PRO 5650G @ 4.4 GHz with Radeon Graphics, 1 socket with 6 cores per socket and 2 threads per core,
- 8 GB RAM.



The machines we have in the department

Toeplitz Cluster made of **5 + 4 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.

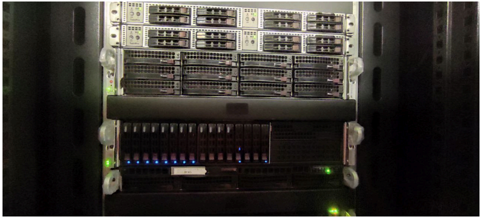
Toeplitz Cluster made of **5 + 4 nodes**:

- 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.
- 4 Nodes AMD® with
  - 2 × EPYC® 7763 64-Core Processor with 2 Threads per core, 64 Cores per socket, 2 Sockets,
  - 2 TB RAM,
  - 4 NVIDIA GPU/A40 48 GB GDDR6.

The machines we have in the department

Toeplitz Cluster made of **5 + 4 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

Toeplitz Cluster made of **5 + 4 nodes**:

- 4 Nodes AMD® with
  - 2 × EPYC® 7763 64-Core Processor with 2 Threads per core, 64 Cores per socket, 2 Sockets,
  - 2 TB RAM,
  - 4 NVIDIA GPU/A40 48 GB GDDR6.

The machines we have in the department

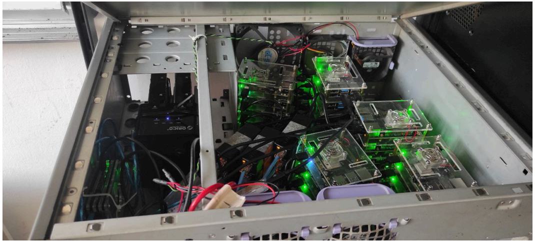
Toeplitz Cluster made of **5 + 4 nodes**:

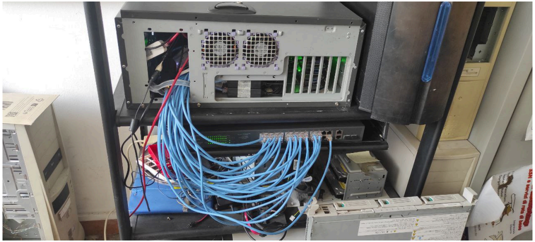
- 4 Nodes Intel® Xeon® CPU E5-2650 v4 @ 2.20GHz with 2 threads per core, 12 cores per socket and 2 socket with 256 GB;
- 1 Node Intel® Xeon® CPU E5-2643 v4 @ 3.40GHz with 2 threads per core, 6 cores per socket and 2 socket with 128 GB.

- 4 Nodes AMD® with
  - 2 × EPYC® 7763 64-Core Processor with 2 Threads per core, 64 Cores per socket, 2 Sockets,
  - 2 TB RAM,
  - 4 NVIDIA GPU/A40 48 GB GDDR6.

The machine we built here last year and that we will improve this year!

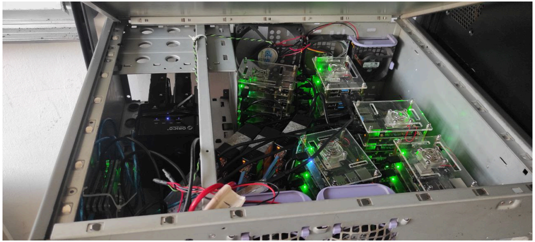
The machines we have in the department

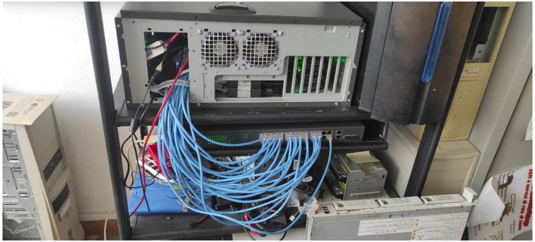




- 1 Access Node
- **20 Nodes** with
  - Hexa-core Arm® big.LITTLE™ dual Arm Cortex® A72, quad Cortex-A53 CPU
  - Arm Mali™ T860MP4 GPU
  - RAM 4 Gb LPDDR4 a 64 bit
- **6 nodes** are equipped with a Google Edge TPU coprocessor 4 TOPS (int8); 2 TOPS per watt.

The machines we have in the department





- 1 Access Node
- **20 Nodes** with
  - Hexa-core Arm® big.LITTLE™ dual Arm Cortex® A72, quad Cortex-A53 CPU
  - Arm Mali™ T860MP4 GPU
  - RAM 4 Gb LPDDR4 a 64 bit
- **6 nodes** are equipped with a Google Edge TPU coprocessor 4 TOPS (int8); 2 TOPS per watt.

We **plan** to **add** other **15 Nodes**.

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 ymbsittendra
ofer hronrade hyran scolde,
gomban gyldan. Þæt wæs god cyning.

"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

4 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.

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 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

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

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

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 $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$

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

- 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 $1, \ldots, i$, and one summing up the entries between $i+1, \ldots, n$ we take T_i time for the first part and T_{n-i} time for the second, therefore the overall time is $\max(T_i, T_{n-i})$ for doing always $O(n)$ operations.

Parallel Algorithms: *speedup*

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 ith portion occupying the P_i fraction of the overall completion time,

Parallel Algorithms: *speedup*

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 ith portion occupying the P_i fraction of the overall completion time,
- order the portions in such a way that the Nth portion subsumes all the parts of the overall processes with fixed costs.

Parallel Algorithms: *speedup*

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 ith portion occupying the P_i fraction of the overall completion time,
- order the portions in such a way that the Nth portion subsumes all the parts of the overall processes with fixed costs.
- The *speedup* of the ith portion can then be defined as

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

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

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 $\mathbf{P} = (P_1, \ldots, P_N)$, $\mathbf{S} = (S_1, \ldots, S_{N-1})$ 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*

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.

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 si $S_{\text{max}} = 1/P_N$.

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

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*

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.

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

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 $S_i \to +\infty \ \forall i$, 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

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

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**"

Table of Contents

Connecting to a Remote Machine using SSH

Secure Shell

SSH (Secure Shell) is a cryptographic network protocol that allows secure communication over an unsecured network. It is commonly used for remote login to execute commands on a remote machine securely.

- SSH provides a secure encrypted connection between a client and a server.
- It ensures that data transmitted between the client and server is encrypted, preventing eavesdropping and unauthorized access.
- SSH is widely used in managing remote servers, transferring files securely, and executing remote commands.

SSH Connection Process

Connection Process

To establish an SSH connection to a remote machine, follow these steps:

1. **Open Terminal:** Launch your terminal application.
2. **SSH Command:** Use the SSH command with the following syntax:
   `ssh username@hostname`
3. **Authentication:** Enter your password when prompted. Some setups may require SSH keys for authentication.
4. **Connected:** Once authenticated, you are connected to the remote machine's shell.

Example: Connecting to a Remote Machine

SSH Command Syntax

```
ssh username@hostname
```

Example Command

```
ssh <login-ateneo>@login.cs.dm.unipi.it
```

Note

Replace `username` with your username on the remote machine and `hostname` with the hostname or IP address of the remote machine.

SSH Key Authentication

SSH Keys

SSH keys provide a more secure method of authentication compared to passwords.

- A key pair is generated: public and private keys.
- The **public key** is stored on the **remote server**, while the **private key** remains on your **local machine**.
- Authentication is based on possession of the private key.
- SSH keys are often used for automated processes and server-to-server communication.

OpenSSH in Windows

The latest builds of Windows 10 and Windows 11 include a built-in SSH server and client that are based on OpenSSH.

- OpenSSH is a connectivity tool for remote sign-in that uses the SSH protocol.
- It encrypts all traffic between client and server to eliminate eavesdropping, connection hijacking, and other attacks.

Location of OpenSSH Client

By default, the OpenSSH client is located in the directory:
`C:\Windows\System32\OpenSSH`.

Checking Installation

You can also check that it is installed in **Windows Settings > Apps > Optional features**, then search for ”OpenSSH” in your installed features.

Connecting to a Server via SSH in Terminal (Mac)

Step 1: Open Terminal

In Finder, open the Applications folder and double click on the Utilities folder.

Step 2: Enter the SSH Command

The basic syntax of connecting to SSH is as follows:

```
ssh user@IP-Address
```

Replace `user` and `IP-Address` with the username and IP address/name of the remote server. Hit return to execute the command.

You can 🔑 **generate your key** either on your *personal machine* or on the machine `login.cs.dm.unipi.it` so as to have it preserved.

Step 1: Open Terminal

Open your terminal application.

Step 2: Run the Command

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Note

Replace `your_email@example.com` with your UNIPI email address.

Step 3: Accept Default Location

When prompted to ”Enter a file in which to save the key”, press Enter to accept the default file location. If you want to create a custom-named SSH key, type the desired file location and replace `id_ALGORITHM` with your custom key name.

```
Enter a file in which to save the key (/home/YOU/.ssh/id_ALGORITHM):
↪  [Press enter]
```

Step 4: Type Passphrase

At the prompt, type a secure passphrase.

```
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type passphrase again]
```

Adding SSH Private Key to ssh-agent

Step 5: Start ssh-agent (only on Linux)

Start the ssh-agent in the background.

```
eval `ssh-agent`
```

Step 6: Add SSH Private Key

Add your SSH private key to the ssh-agent.

```
ssh-add ~/.ssh/id_ed25519
```

Note

If you **created your key with a different name**, or if you are adding an existing key with a different name, replace `id_ed25519` in the command with the name (and place) of your private key file.

Virtual Private Network (VPN)

VPN

A Virtual Private Network (VPN) extends a private network across a public network, enabling users to securely transmit data as if their devices were directly connected to the private network.

- VPNs provide privacy, security, and anonymity by encrypting data and masking IP addresses.
- They are commonly used for remote access to corporate networks, bypassing geographical restrictions, and enhancing online privacy.

Virtual Private Network (VPN)

VPN

A Virtual Private Network (VPN) extends a private network across a public network, enabling users to securely transmit data as if their devices were directly connected to the private network.

- VPNs provide privacy, security, and anonymity by encrypting data and masking IP addresses.
- They are commonly used for remote access to corporate networks, bypassing geographical restrictions, and enhancing online privacy.

The University of Pisa uses it to allow **access** to its sensitive resources **from machines external to the university network**.

How VPN Works

Connection Process

To establish a VPN connection, follow these steps:

1. **VPN Client:** Install and configure a VPN client software on your device.
2. **Authentication:** Enter your credentials (username and password) or use other authentication methods.
3. **VPN Server:** Connect to a VPN server hosted by a VPN service provider.
4. **Tunneling:** Establish a secure encrypted connection between your device and the VPN server.
5. **Data Transmission:** Transmit data through the encrypted tunnel, ensuring privacy and security.

Types of VPN

1. Remote Access VPN

- Allows individual users to securely connect to a private network remotely.
- Commonly used by employees to access corporate networks from outside the office.

2. Site-to-Site VPN

- Connects multiple networks together, such as branch offices to a central corporate network.
- Provides secure communication between different geographical locations.

Advantages of VPN

1. Security

- Encrypts data transmitted over public networks, preventing unauthorized access.
- Protects against cyber threats and surveillance.

2. Privacy

- Masks IP addresses, preserving anonymity and preventing tracking.
- Secures online activities from ISPs and government surveillance.

3. Access Control

- Grants access to restricted resources based on user credentials and policies.
- Enables bypassing of geo-blocked content.

Protection Against Cyber Attacks

To counter the expansion of cyber attacks on University resources, increasingly stringent filters have been introduced to protect digital resources hosted within the University network.

VPN Service

The University VPN service offers different profiles for accessing different digital resources:

- Access to UNIPI resources (Accesso risorse UNIPI)
- Internet through UNIPI (Internet attraverso UNIPI)
- Bibliographic resources (to be *discontinued*)
- External Staff

Protection Against Cyber Attacks

To counter the expansion of cyber attacks on University resources, increasingly stringent filters have been introduced to protect digital resources hosted within the University network.

Which profile to choose

To obtain an IP address internal to UNIPI you must choose “Internet through UNIPI” (Internet attraverso UNIPI).

VPN Service

The University VPN service offers different profiles for accessing different digital resources:

- Access to UNIPI resources (Accesso risorse UNIPI)
- Internet through UNIPI (Internet attraverso UNIPI)
- Bibliographic resources (to be *discontinued*)
- External Staff

Installing the VPN Program

5 Auxiliary tools

For PC/Mac

The technology used is **Connect Tunnel** by SonicWALL. You need to download the program from here (choose the version for your platform).

For Smartphone/Tablet

Android: Download the SonicWALL Mobile Connect app from the PlayStore.

iOS: Download the SonicWALL Mobile Connect app from the Apple Store.

To get versions from your phone you can use QR codes.

Server Information

The VPN connection server is called ***access.unipi.it***.

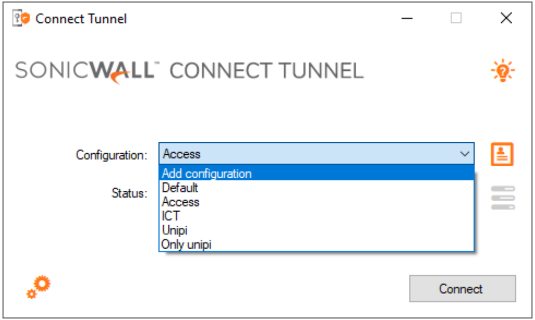


Figure: Adding a New Configuration

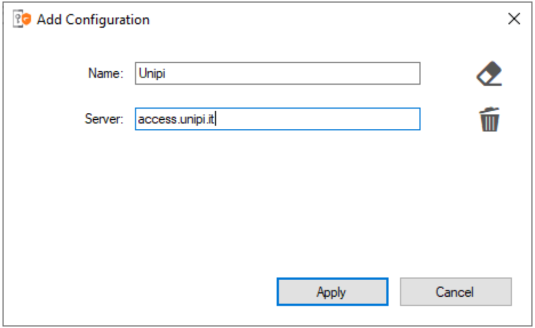


Figure: Naming and Addressing the Server

Configuring a VPN Connection (Contd.)

Connection Procedure

- Save the configuration.
- Select it and press *Connect*.
- The system will present the service terms and, upon first access, ask to select the connection profile.
- To change the profile selection, modify the configuration and use the ”forget login group” function represented in the Windows application by the eraser icon.

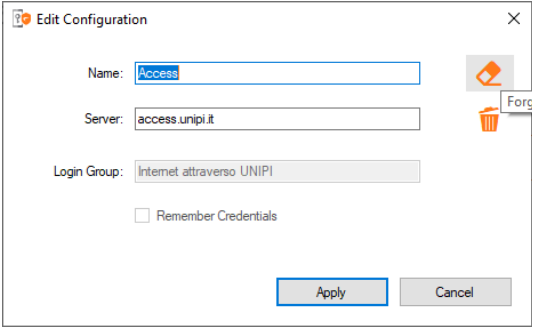


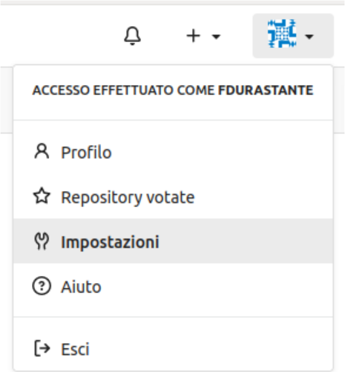
Figure: Forget Login Group Function

Software Version Control: GIT

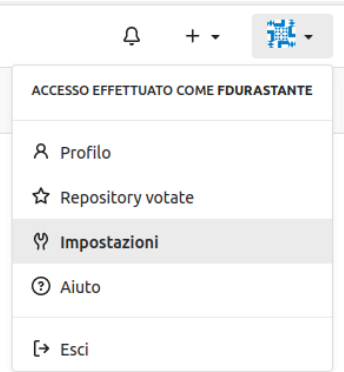
In software engineering, version control is a class of systems responsible for managing changes to **computer programs**, **documents**, large web sites, or other *collections of information*. Version control is a component of software configuration management.

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

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



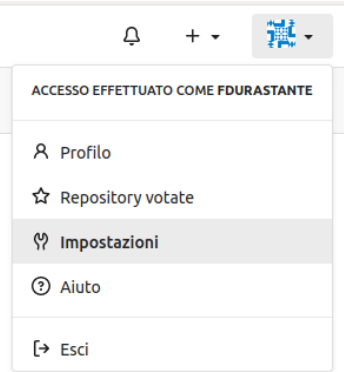
From the settings menu you have access to the configurations of the Git service.



- SSH key entry:

Getting an up-and-running GIT account

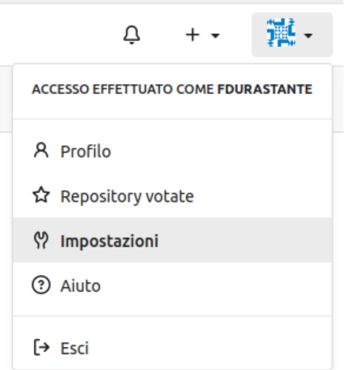
From the settings menu you have access to the configurations of the Git service.



- SSH key entry

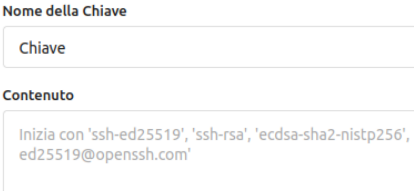
Getting an up-and-running GIT account

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

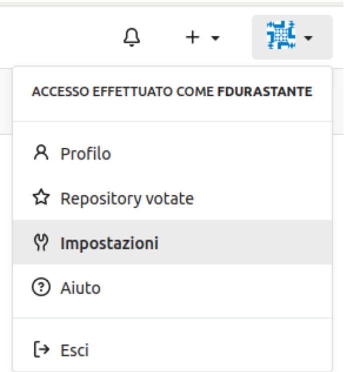


- SSH key entry
  - 

  - Which inserts similarly:

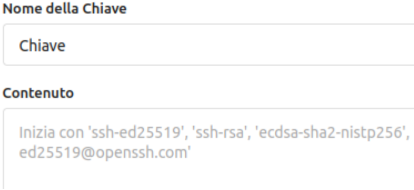


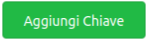
From the settings menu you have access to the configurations of the Git service.

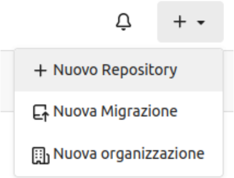


- SSH key entry
  - 

  - Which inserts similarly:



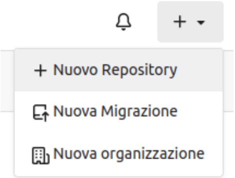
  - Concluding with: 



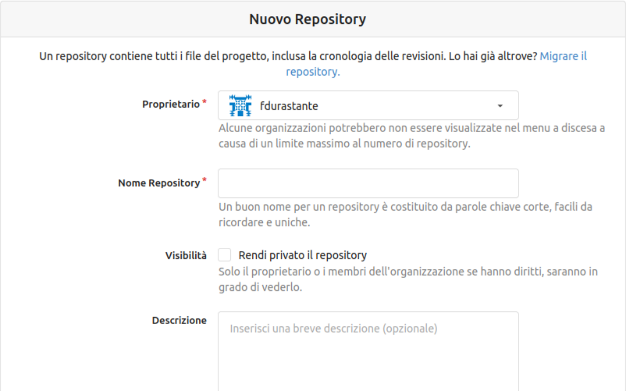
- You can create a new repository easily.

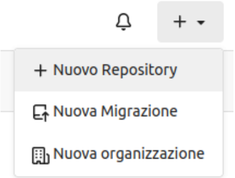
A repository

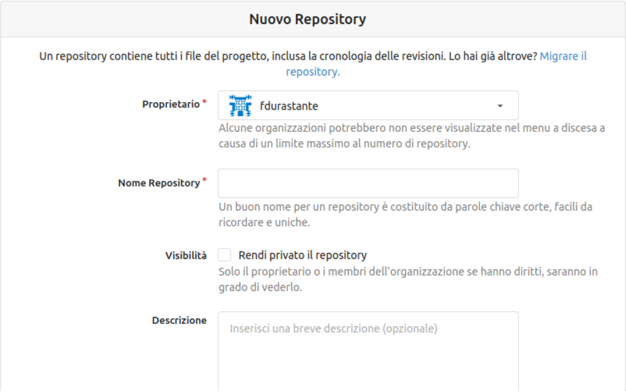
5 Auxiliary tools

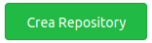


- You can create a new repository easily.



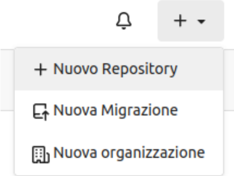


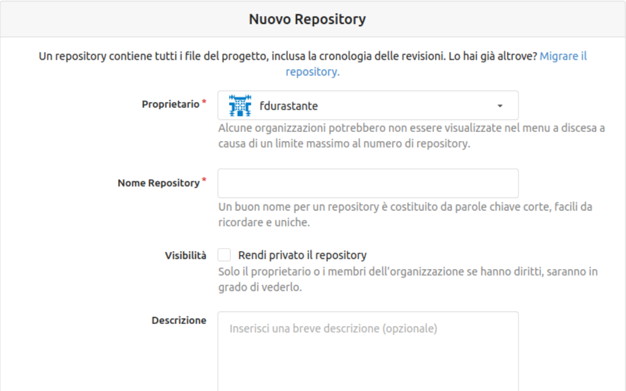


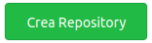
- You can create a new repository easily.
- And then: 

A repository

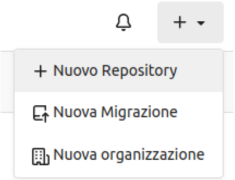
5 Auxiliary tools

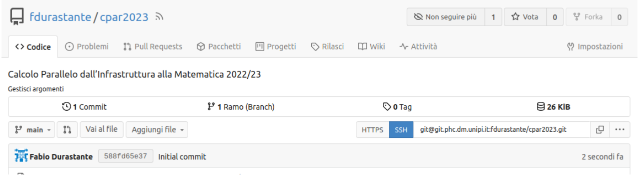


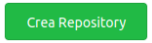


- You can create a new repository easily.
- And then: 

A repository





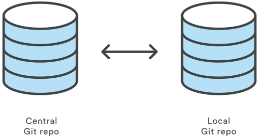
- You can create a new repository easily.
- And then: 


```
git clone git@git.phc.dm.unipi.it:HighPerformanceMathematics/HPM-Lezioni2024.git
cd HPM-Lezioni2024
```

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

GIT Workflow

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.