MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

IN COLLABORATION WITH MIT MANAGEMENT SLOAN SCHOOL

UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

MASTER MEIM 2022-2023

DIGITAL TECH

High Performance Computing

Lesson 4

Prof. Livia Marcellino

Prof. of High Performance Computing, Università degli Studi di Napoli Parthenope

Parallel Software Design

The knowledge of how the hardware is made and the study of tools available allows us for choosing the most suitable HPC environment and the strategy of more efficient parallelization for the numerical resolution of our large-scale problem.

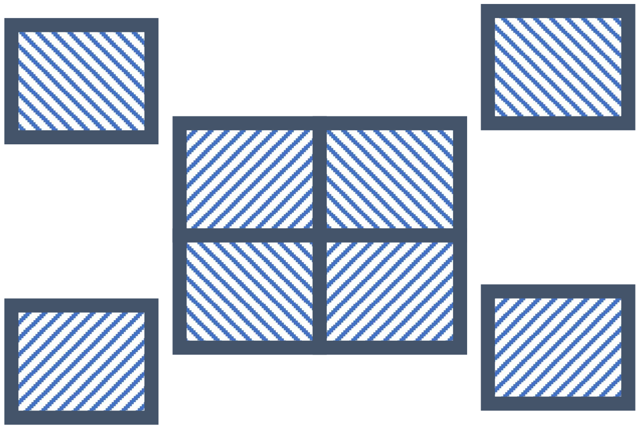
PARALLEL COMPUTING

Decompose a problem

in more subproblems

and solve them at the same time

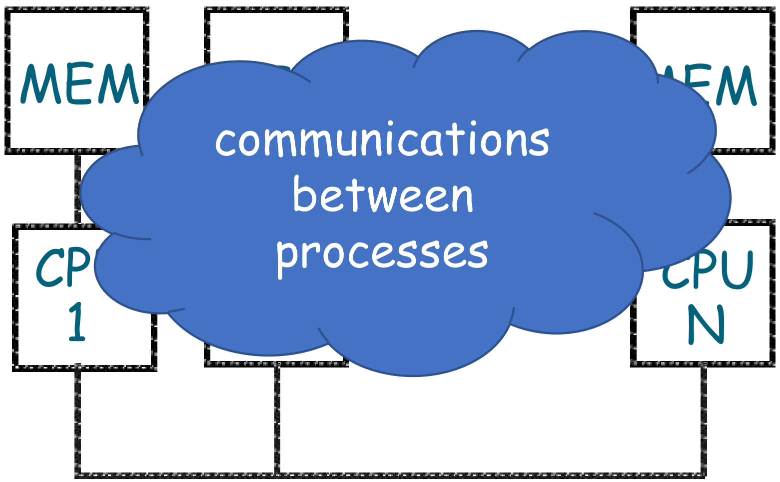
with more processing units!



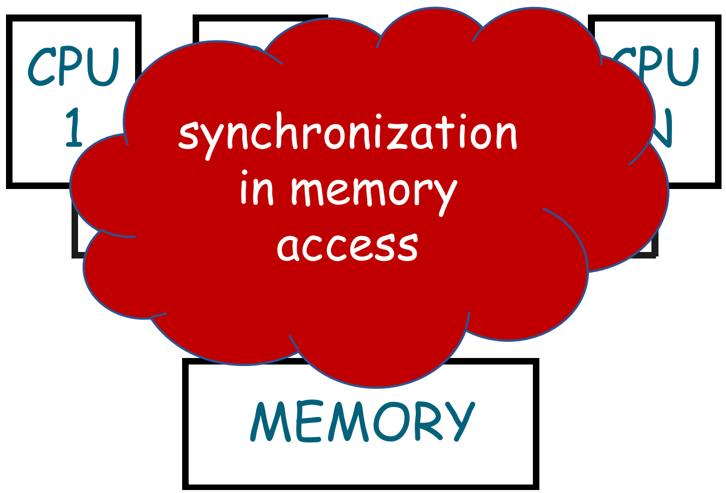
Need to create machines that can distribute the work among them
hardware development

The most important modern parallel architectures

Computer MIMD
DM
(distributed-memory)



Computer MIMD
SM
(shared-memory)



synchronization
and communication
between different
devices

MASTER IN ENTREPRENEURSHIP
INNOVATION MANAGEMENT
IN COLLABORATION WITH MIT SLOAN

UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

Parallel and distributed software performance: performance measures

Evaluate the efficiency of
a parallel algorithm
in a parallel computing environment

What does "EFFICIENCY" mean?

Efficiency of a sequential algorithm

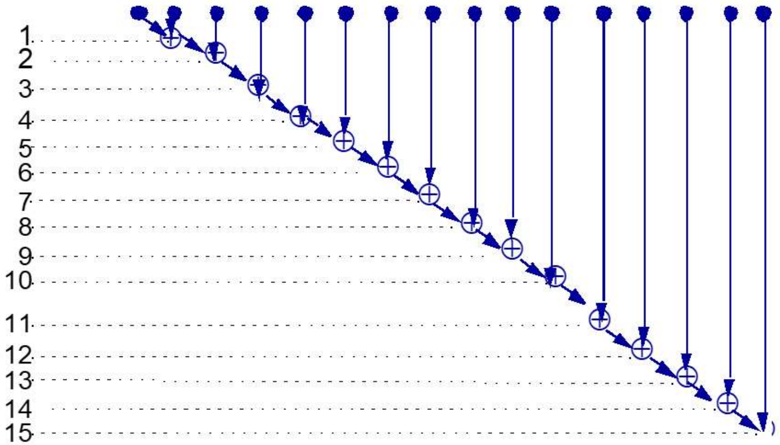
- COMPUTATIONAL COMPLEXITY T(N)

  Operations number make by the algorithm

- SPACE COMPLEXITY S(N)

  Variables number used by the algorithm

Example: sum of *N=16* numbers



addiction number = 15

temporal steps = 15

Time complexity

T(N)=N-1 addictions

MASTER IN ENTREPRENEURSHIP INNOVATION MANAGEMENT IN COLLABORATION WITH MIT SLOAN

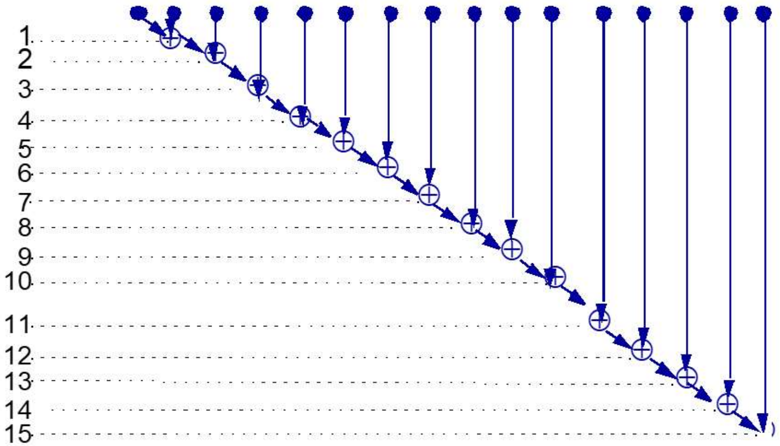
UNIVERSITÀ DEGLI STUDI DI NAPOLI PARTHENOPE

Example: sum of *N=16* numbers

addiction number = 15

temporal steps = 15

SEQUENTIAL ALGORITHM **1 CPU**



Time complexity

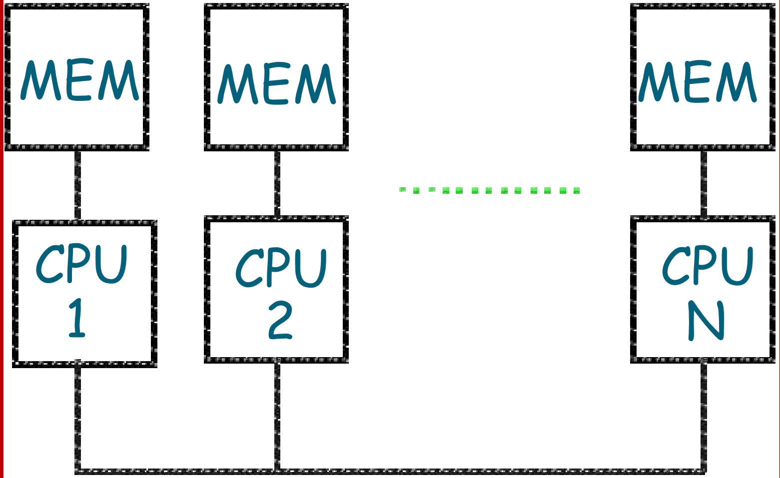
$T_1(16)=15$

Execution time of serial software

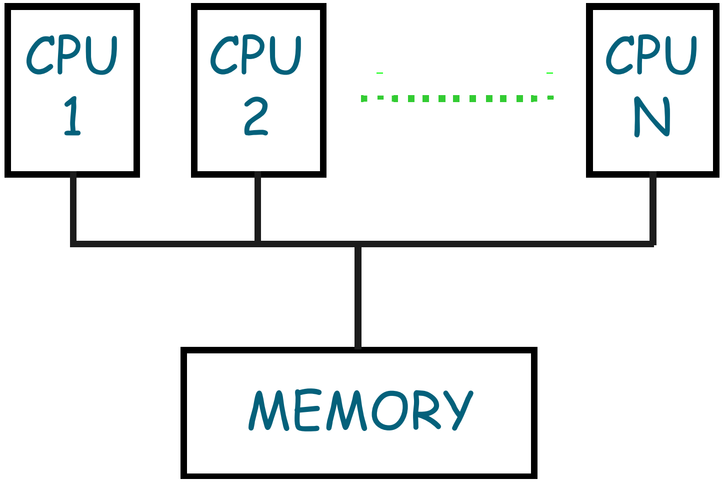
$$\tau = k \cdot T_1(n) \cdot \mu$$

…from now on we will consider the multicore environment

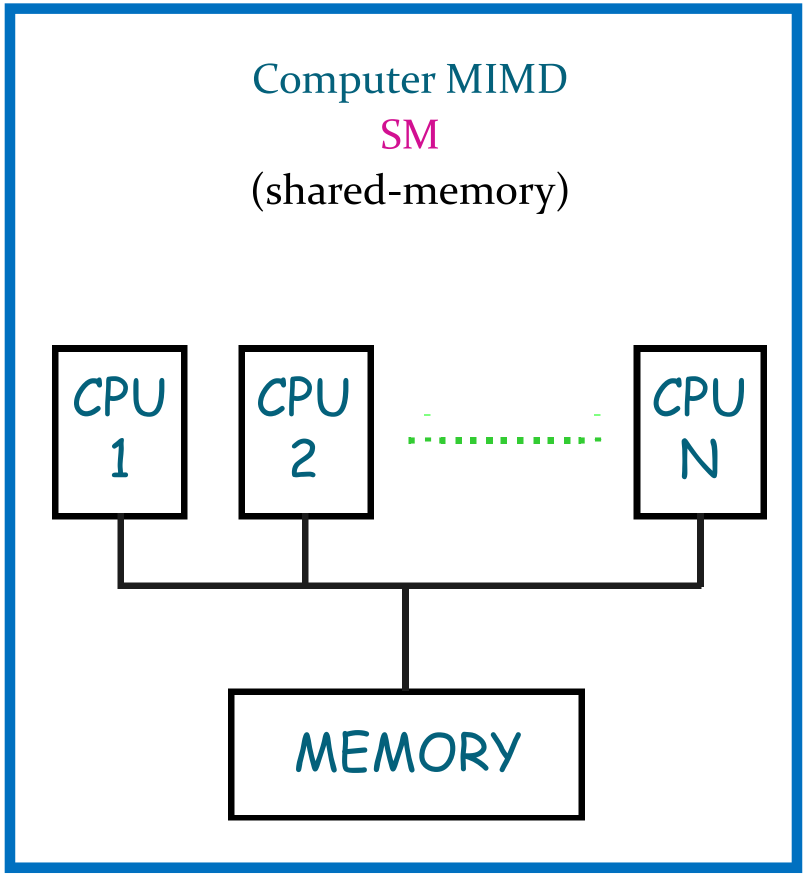
Computer MIMD
DM
(distributed-memory)



Computer MIMD
SM
(shared-memory)



…from now on we will consider the multicore environment



Efficiency of a parallel algorithm

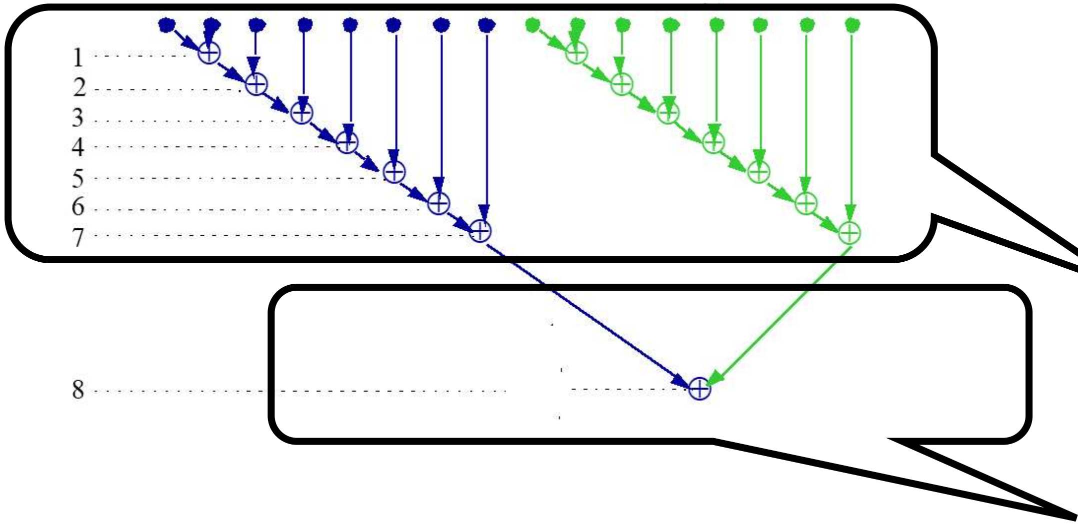
Example: sum of *N=16* numbers

Additions number = 15

BUT

Temporal steps = 8

PARALLEL ALGORITHM **2 CPUs**



Temporal steps 1-7: 14 addictions

(7 for each CPU)

- Temporal step 8: 1 addiction

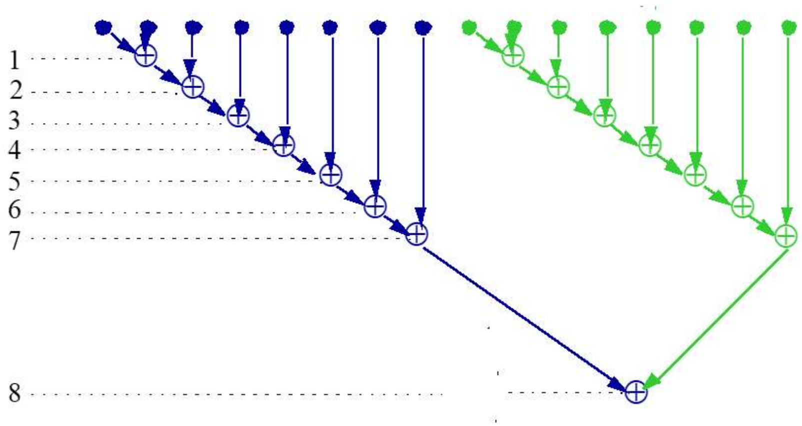
Example: sum of *N=16* numbers

Additions number = 15

BUT

Temporal steps = 8

PARALLEL ALGORITHM **2 CPUs**



$T_2(16)=8$

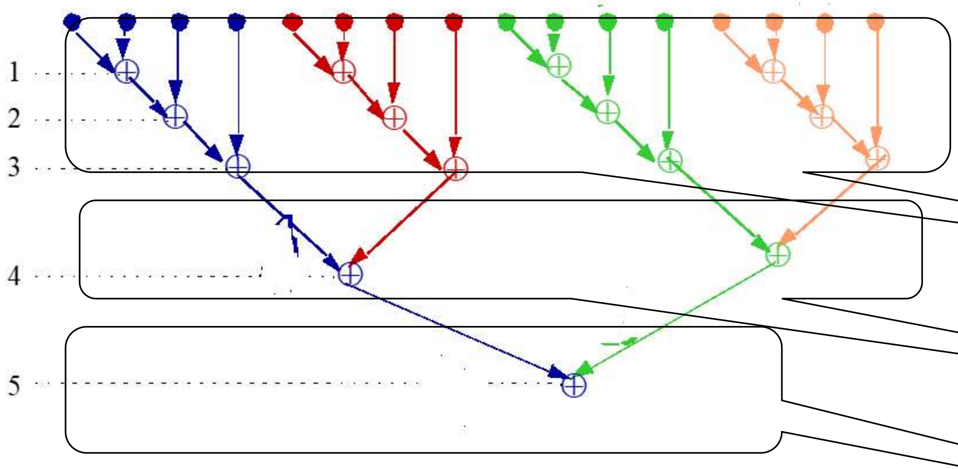
Example: sum of *N=16* numbers

Addictions number = 15

BUT

Temporal steps = 5

PARALLEL ALGORITHM 4 CPUs



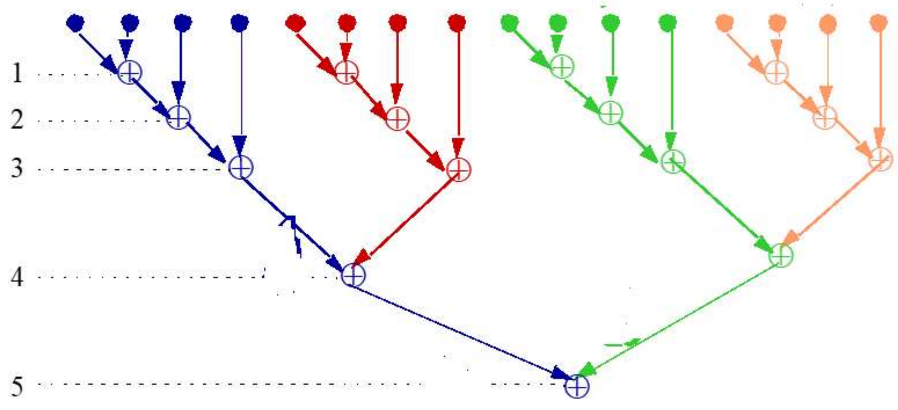
Temporal steps 1-3: 12 addictions

Temporal steps 4: 2 addictions

Temporal steps 5: 1 addiction

Example: sum of *N=16* numbers

PARALLEL ALGORITHM **4 CPUs**



Addictions number = 15

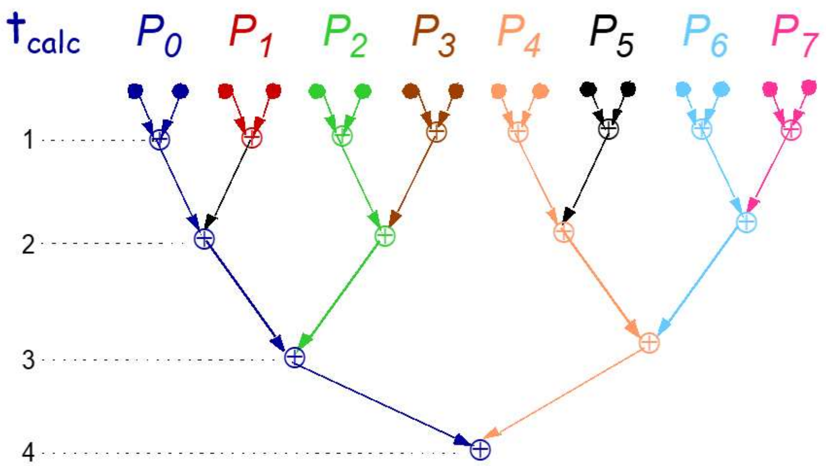
BUT

Temporal steps = 5

$T_4(16)=5$

Example: sum of *N=16* numbers

PARALLEL ALGORITHM **8 CPUs**



Addictions number = 15

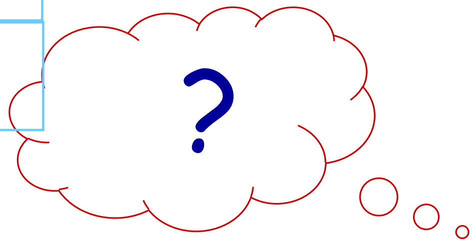
BUT

Temporal steps = 4

$T_8(16)=4$

now I put everything in a table…

p	$T_p(16)$
1	15
2	8
4	5
8	4
p	?



In general, how much is T_p?

In general: $T_p(N)$ computation

PARALLEL ALGORITHM: sum of *N* numbers

p CPUs

p=1	$T_1=15$	
p=2	$T_2=8 = (7+1)$	
p=4	$T_4=5 = (3+2)$	N = 16
p=8	$T_8=4 = (1+3)$	
……………….		

$$T_p(N) = (N/p - 1 + \log_2 p)$$

Questions…

p	T_p
1	15
2	8
4	5
8	4

What is the fastest algorithm?

?

How much faster is it than the sequential algorithm?

Example: sum of $N=16$ numbers

p	T_p	T_1/T_p
1	15	1.00
2	8	1.88
4	5	3.00
8	4	3.75

The algorithm that uses 8 CPUs is the fastest

It is **3.75 times** faster than that with 1 CPU

Speed-up

The ratio of T_1 to T_p is defined

$$S_p = \frac{T_1}{T_p}$$

The speed up measures the execution time reduction with respect to the serial algorithm

$$S_p < p$$

IDEAL SPEEDUP

$$S_p^{ideale} = p$$

Remark

$$S_p^{ideale} = \frac{T_1}{T_p} = p$$

$$O_h = (pT_p - T_1)$$

OVERHEAD

The OVERHEAD measures

how much the speed up differs from the ideal one

Example: sum of *N=16* numbers

p	Speed-up	Speed-up ideal
2	1.88	2
4	3.00	4
8	3.75	8

The speed-up on 8 GPUs is the highest

BUT

The speed-up using 2 GPUs is

"the closest" to the ideal speed-up

Example: sum of *N=16* numbers

… if you compare the speed-up to the number of CPUs …

p	S_p	S_p/p
2	1.88	0.94
4	3.00	0.75
8	3.75	0.47

The best ratio

using p=2

Efficiency

The ratio of S_p to p

$$E_p = \frac{S_p}{p}$$

measures how much the algorithm exploits the parallelism

IDEAL EFFICIENCY

$$E_p^{ideale} = \frac{S_p^{ideale}}{p} = 1$$

Remark:

Given definitions are only for the

MIMD-SM environments

For

MIMD-DM and GPU environments,

the execution time does NOT depend only on the operations number

I have to consider also times for data communications

Let’s take a break

Why do we have to measure the performance of a parallel algorithm?

the need of a real time solution!

Big Data Problems

- Search on the Internet
- Automatic Planning
- Advertising and Marketing
- Banking and financial services
- Media and Entertainment
- Meteorology
- Health Care
- Cyber Security
- Training

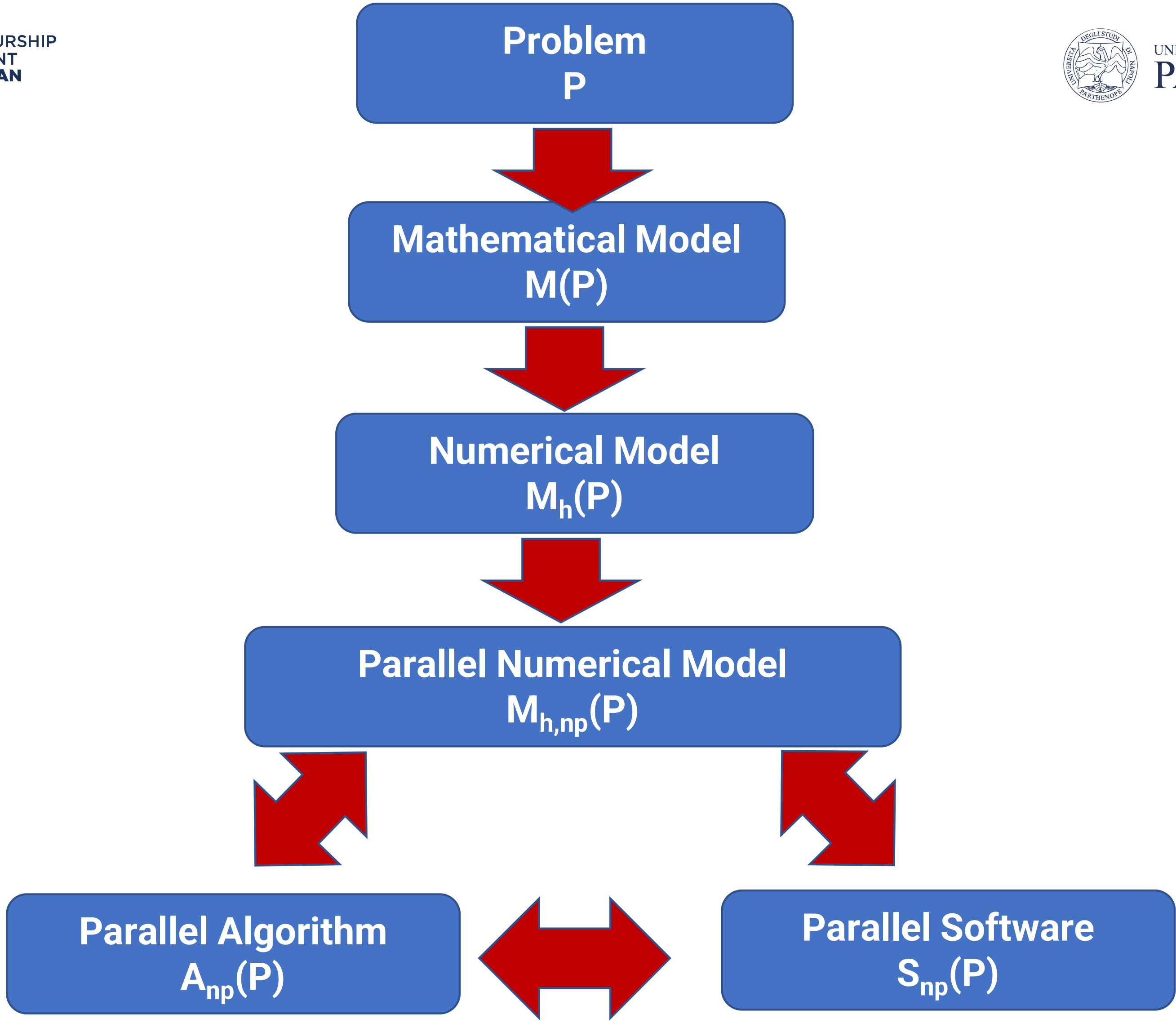
Problems characterized by the need to obtain real-time solution (or just in time!)

MASTER IN ENTREPRENEURSHIP
INNOVATION MANAGEMENT
IN COLLABORATION WITH MIT SLOAN

UNIVERSITÀ DEGLI STUDI DI NAPOLI
PARTHENOPE

High Performance Computing for financial applications

Problem Solving chain



High Performance Computing for financial applications

Among applications that can benefit from HPC architectures there is the **Financial Data Mining**.

Financial Data Mining →

- Dynamic multi-stage management of portfolio
- Risk management
- Stock valuation

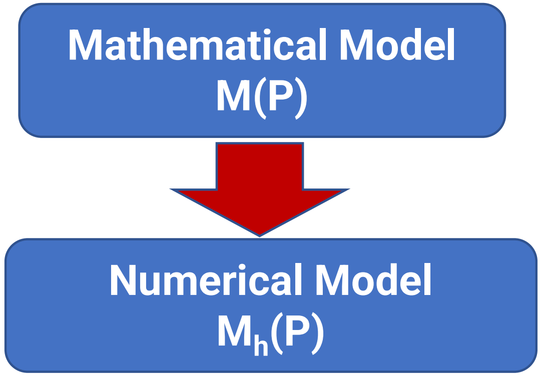
High Performance Computing for financial data mining

Dynamic multi-stage management of portfolio

Portfolio optimization is widely used by ***banks and companies*** to offer financial services, it is used to solve the problem of **how to diversify investments** in different asset classes.

Due to the many uncertain factors, the final financial model is a ***stochastic problem***, where the parameters are random.

To solve it efficiently, a possible solution is **to decompose the problem into sub-problems,** which are solved in **parallel**.
In this way managers can predict the solution, using their models on parallel algorithms based on the previous day's trading results and rebalance their portfolio **in real time**.

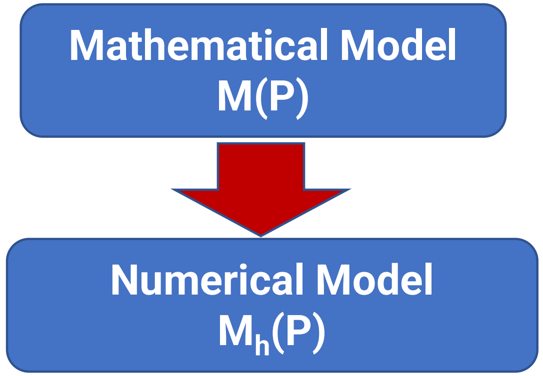


High Performance Computing for financial data mining

Risk management

Stock investments always imply a compromise called **risk-reward**. The goal of investors is to minimize this risk and increase gain.

The **Value-at-Risk** is used to forecast the loss of money and usually it is estimated using the **Monte-Carlo** method (simulatation of possible "scenarios" that can happen in the real world based on past security prices and probability theory).



Increasing the number of simulations the results obtained can become very accurate.

This problem it is easily parallelizable since each simulation can be performed independently, by using a **functional decomposition**.

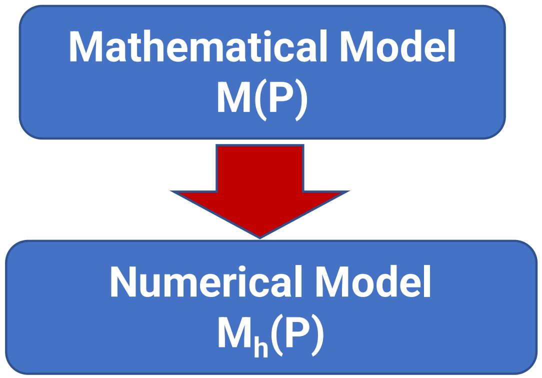
High Performance Computing for financial data mining

Stock valuation

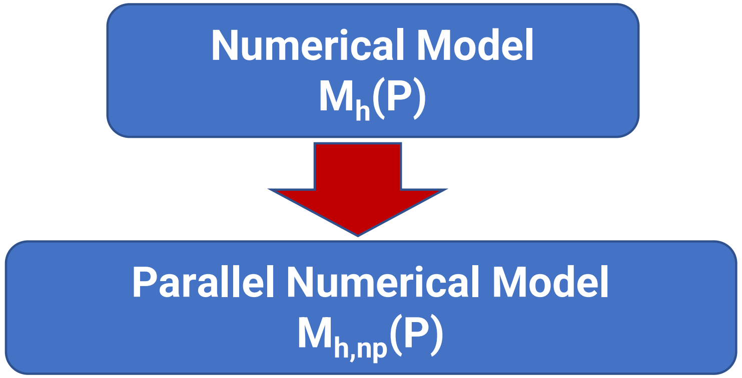
In order to evaluate the stock price mathematical models, based on **partial-differential equation** are used.

The computational kernel of the numerical solution involves **linear and non linear systems**.

Parallel solution of linear and non linear systems is a challage in MIMD environment



Parallel solution of linear systems of equations



System of equations

A linear system of equations (m equations and n unknowns)

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{cases}$$

If m=n I can try to compute

the unique solution,
if it exist

In matrix form: $Ax = b$

A → coefficient matrix

x → vettore of unknowns

b → vettor of known terms (Right-Hand Side Vector)

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Among numerical methods to solve linear System of equations…

$$x = A \quad b$$

LU factorization

A → coefficient matrix

x → vettore of unknowns

b → vettor of known terms (Right-Hand Side Vector)

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Among numerical methods to solve linear System of equations

by LU factorization

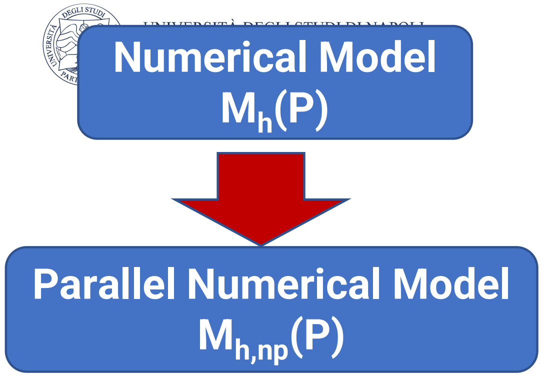
Compute A=LU

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

In many cases a square matrix A can be "factored" into a product of a lower triangular matrix and an upper triangular matrix, in that order. That is, $A = LU$ where L is lower triangular and U is upper triangular. In that case, for a system $A\mathbf{x} = \mathbf{b}$ that we are trying to solve for $\mathbf{x}$ we have

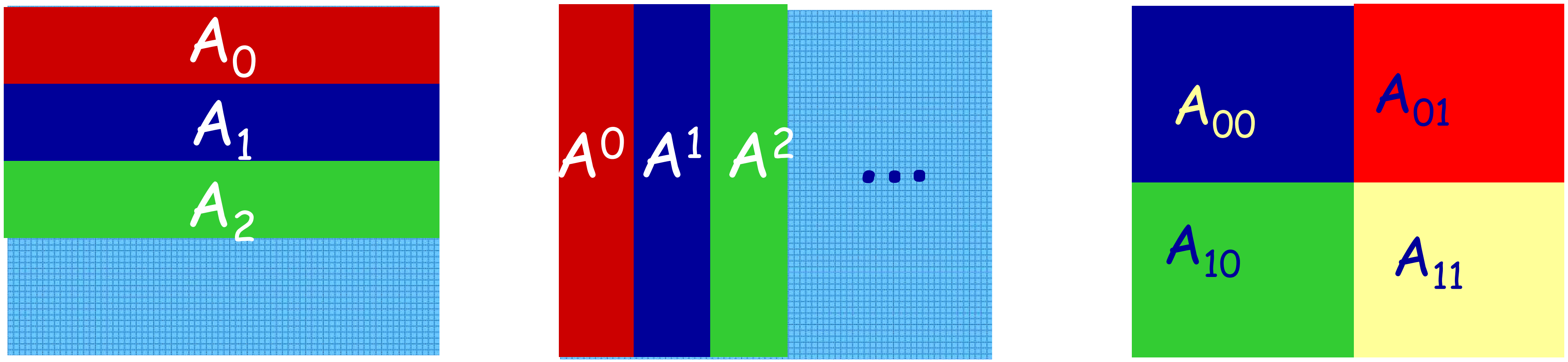
$$A\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad (LU)\mathbf{x} = \mathbf{b} \quad \Rightarrow \quad L(U\mathbf{x}) = \mathbf{b}$$

Note that $U\mathbf{x}$ is simply a vector; let's call it $\mathbf{y}$. We then have two systems, $L\mathbf{y} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{y}$. To solve the system $A\mathbf{x} = \mathbf{b}$ we first solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$. Once we know $\mathbf{y}$ we can then solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$, which was our original goal.



what can I do in parallel?

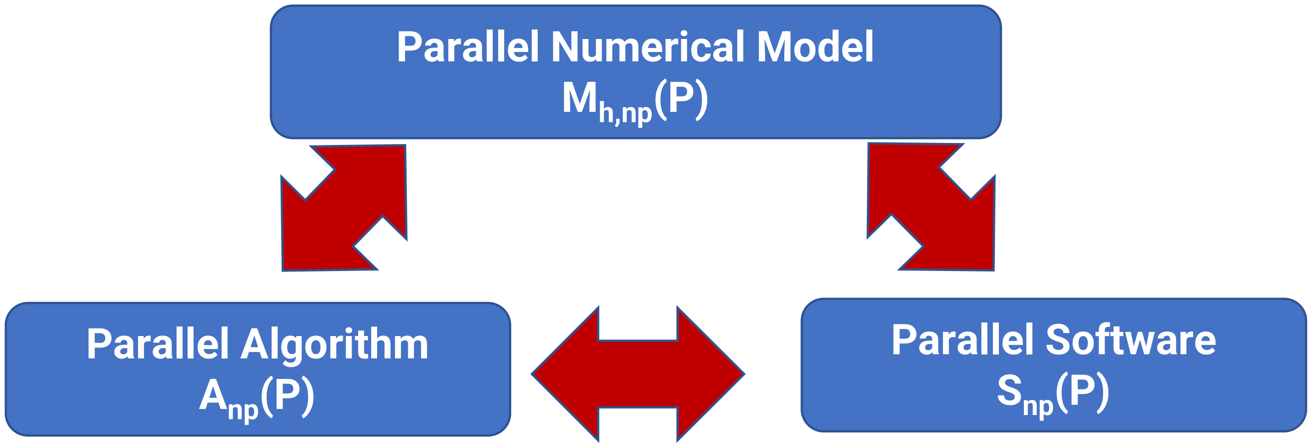
The work on matrix A must be decomposed among the processing units



Problem Solving chain

…

coming up to the software



Python for parallel computing on multicore environments

Python for computing

Python is an object-oriented "high-level" programming language, suitable for multiple uses such as data analysis, artificial intelligence, websites, scripting, ***numerical computation***, etc.
Python was developed in 90' to improve the Perl language.

Main features:

- Easy understanding
- Dynamic Typing
- Dynamic Bindings
- Interpreted language
- Garbage Collector
- Portability

By using Python solve a linear system with matrix A of size 4000x4000 on Intel®Core i7-1065G7, in a serial way.
Execution time: 5.91 seconds

Python for accelerate computing

NumPy

NumPy (**Numerical Python**) it is an open source extension for scientific computing in Python. It provides support for large matrices and multidimensional arrays with precompiled fast functions that operate efficiently on these data structures.

Cython

Cython is a Python compiled language aimed at getting results comparable to the performance of the C programming language. It offers the combined power of Python and C in order to exploit the characteristics of both languages.

Numba

Numba is a just-in-time (JIT) compiler for Python, compatible with NumPy. It allows us to optimize many functions using the LLVM infrastructure which produces optimized machine code.

Python for parallel computing

PyOMP

PyOmp is a new, experimental library developed in December 2021 by researchers at Intel.
It uses Numba's naïve parallelism and the OpenMP directives in a Python environment to exploit the full power of modern multicore parallel architectures.

PyPardiso

PyPardiso is a library that acts as interface for the linear system solver PARDISO, used in Python, i.e. an open source, interpreted and object-oriented programming language (flexibility and portability).

The acronym PARDISO stands for "PARallel DIrect SOLver“.
This package is a thread-safe, high-performance, robust, memory efficient and easy to use software for solving large sparse symmetric and unsymmetric linear systems of equations on shared-memory, distributed-memory multiprocessors and NVIDIA's GPUs.


That's all Folks!