

ADVANCED ALGORITHMS

Parallel Algorithms and Architectures



DATA SCIENCE &
ARTIFICIAL INTELLIGENCE

2025–2026



**UNIVERSITÀ
DEGLI STUDI
DI TRIESTE**

Tommaso Padoan

Teaching Material

All topics seen in class are included and explained in the *handouts*.

Textbook: Leighton F. T. (1992). Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes. Morgan Kaufmann Publishers

Further reading: Roosta S. H. (2012). Parallel Processing and Parallel Algorithms: Theory and Computation. Springer

Slides, handouts, recordings and other info will be available on the Team of Algorithmic Data Mining:

► CD2025 340SM ALGORITHMIC DATA MINING

► *access code:* qxhwjyo

Topics

- ▶ Parallel computing: algorithms, processing and architectures
- ▶ Design and performance analysis of parallel algorithms
- ▶ Interconnection networks and metrics
- ▶ Linear Array and Ring networks – sorting, convolution and DFT
- ▶ Mesh and Torus networks – graph problems and matrix multiplication
- ▶ Binary Tree network – associative operations and selection
- ▶ Hypercube network – communications and embeddings

Exam

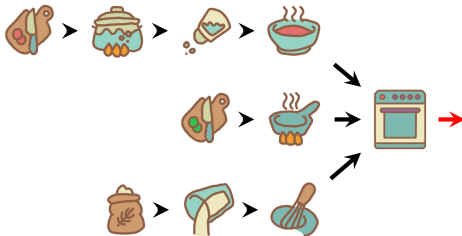
The part of the exam about this module will usually be one big exercise comprising multiple questions, including

1. Description of the graph structure and properties of a given interconnection network
2. Design of a parallel algorithm on such network for a given problem
3. Analysis of the performances of such algorithm

Parallel Computing: not just Parallel Algorithms



Parallel Computing: not just Parallel Algorithms



The Problem of Parallelization

To parallelize a program or algorithm means to carry out its operations simultaneously on multiple processing units

The **parallelizability** of an algorithm can range from easily parallelizable to completely unparallelizable

- ▶ Depending on both its design and the problem it is meant to solve

Parallelization has to take into account dependencies between operations

Dependencies

Dependence analysis identifies the constraints on the order of execution, dividing them into two classes of dependencies

- **Data dependency**: when an instruction refers to the same data accessed by another one preceding it

```
x ← 3  
y ← x + 2
```

- **Control dependency**: when the execution or skip of an instruction is determined by the evaluation of a preceding one

```
if x > y :  
    z ← 4
```

Data Parallelism vs Instruction Parallelism

Parallelism can be achieved in different ways

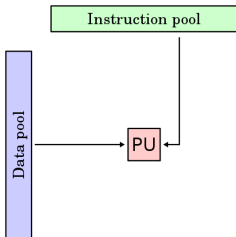
- ▶ **Data parallelism** assigns data elements to multiple processing units that perform the same operation simultaneously on different data
- ▶ **Instruction parallelism** refers to the simultaneous execution of different operations on different processing units

Both kinds of parallelism can coexist and can be applied at the same time

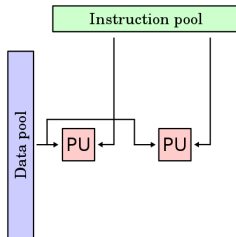
- ▶ Leading to various architectures implementing different combinations

Flynn's Taxonomy

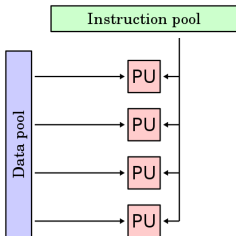
SISD



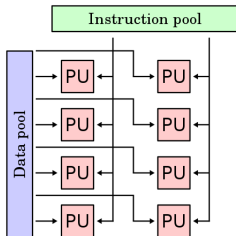
MISD



SIMD



MIMD



Single Program Multiple Data

There is also another division besides single and multiple instruction

Multiple processors can simultaneously execute the same program but at independent points (program counters)

- ▶ A single program is duplicated and distributed to each processing unit
- ▶ It is then executed individually by each unit, usually on different data

This kind of architecture is called **Single Program Multiple Data** (SPMD), in contrast to MPMD where multiple programs are also allowed

Shared Memory vs Distributed Memory

Shared memory allows processors to access a common memory space to read, store and share data and results

- ▶ Communication is simple since all the elaborated data are located in one place directly accessible to all processing units
- ▶ The simultaneous access to data requires critical care to ensure correct synchronization and avoid race conditions

Distributed memory provides each processor with its own private memory

- ▶ Each processing unit can only operate on its local data
- ▶ To access remote data communication is required, accomplished by **message passing** via point-to-point interconnection links

Architectural Choices

The design of a parallel algorithm depends foremost on the architecture on which the algorithm is intended to execute

- ▶ The number of processing units, w.r.t. the size of the problem
- ▶ The kind of memory space, shared or distributed, provided to the processors and the way in which they interact and communicate
- ▶ Whether the control unit is global or local (\sim instruction parallelism)
- ▶ The physical organisation and connections of the processors

All these choices will affect how the parallel algorithm must operate

Size and Memory

The number of processing units can be fixed or dependent on the size of the problem

In the case of distributed memory, the same choice is encountered for the size of the local memory space of each processor

In most of the architectures adopted in the course

- ▶ the number of processors P will **grow** with the size of the problem N
- ▶ each provided with a local memory of **constant** size $M = O(1)$

Input Distribution

Having many processing units with local memory requires to decide how to divide and distribute the input data among them

- ▶ Depending on the problem to be solved
- ▶ Considering the tradeoff between duplication and communication
- ▶ Promoting ease and clarity of algorithm and implementation

Another approach is to have inputs delivered to processors and then flow through connections at regular intervals, known as **systolic computation**

- ▶ Time is partitioned into steps by a *global clock*
- ▶ At each cycle each processing unit *receives*, *elaborates* (as indicated by its local control), and *passes* data to the next unit

Example: Distribution of a Square Matrix

To distribute a $N \times N$ matrix to P processors, with N multiple of P , two different kinds of division can be employed

- ▶ **Block distribution** on rows, columns, or both: each processor receives a block of N/P rows or columns, or a square block of side N/\sqrt{P}
- ▶ **Cyclic distribution** on rows, columns, or both: the rows, columns, or their combination, respectively, are cyclically assigned one at a time to the processors

The two ways can also be combined

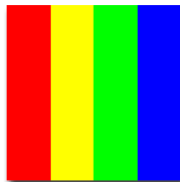
- ▶ E.g.: block distribution on rows and cyclic on columns

Example: Distribution of a Square Matrix

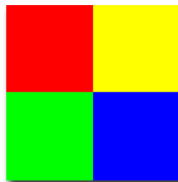
Examples with 4 processing units (identified by the colours) and $N = 16$



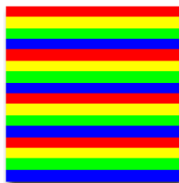
BLOCK, *



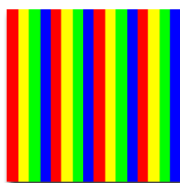
***, BLOCK**



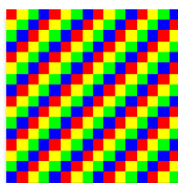
BLOCK, BLOCK



CYCLIC, *



***, CYCLIC**



CYCLIC, CYCLIC

Example: Square Matrix Multiplication

Naive Algorithm

Problem: multiply two matrices A and B of size $N \times N$

Architecture: P processing units, with N multiple of P

Block distribution on rows of A , *duplication* of B

Algorithm

Every processing unit i :

1. receives $A(i)$, i.e. the i -th block of N/P rows of A , and a copy of B
2. computes $C(i) = A(i) \cdot B$
3. returns $C(i)$

Example: Square Matrix Multiplication

Without Duplication

Block distribution on rows of A and of B

Algorithm

Every processing unit i :

1. receives $A(i)$ and $B(i)$, i.e. the i -th blocks of N/P rows of A and B
2. initialises $C(i) = A(i, i) \cdot B(i)$, where $A(i, j)$ is the j -th block of N/P columns of $A(i)$
3. for $k = 1 \dots P-1$:
 - a. sends $B(i)$ to processor $i+1 \bmod P$
 - b. overwrites $B(i) = B(i-1)$ received from processor $i-1 \bmod P$
 - c. computes $C(i) = C(i) + A(i, i-k \bmod P) \cdot B(i)$
4. returns $C(i)$

Performance Analysis of Parallel Algorithms

Parallel algorithms can be analysed from more angles than sequential ones

Given the number of processing units P and the size of the problem N

The local memory M required by each processing unit can be seen as **space complexity**

The parallel execution time $\mathcal{T} = \mathcal{T}_P(N)$ corresponds to **time complexity**

But there is more...

Performance Analysis of Parallel Algorithms

Work and Communications

The **work** is the total processing effort needed to run the algorithm

- ▶ It includes processors being idle or performing communication
- ▶ $\mathcal{W} = \mathcal{T} \cdot P$

Sometimes it is useful to analyse the **communication overhead**, i.e. the ratio of communications to total work

- ▶ $\mathcal{C} = \mathcal{W}^C / \mathcal{W}$ where \mathcal{W}^C is the total number of communications
- ▶ It is assumed that in one time step a single communication can occur through each physical connection

Performance Analysis of Parallel Algorithms

Speedup and Efficiency

The **speedup** measures the gain in speed obtained through parallelization

- ▶ It is the ratio of the execution time Γ of the fastest sequential algorithm for the problem, to that of the parallel algorithm
- ▶ $\mathcal{S} = \Gamma/\mathcal{T} \leq \mathcal{T}_1(N)/\mathcal{T} \leq \mathcal{W}/\mathcal{T} = P$

The speedup per processor measures the **efficiency** with which each processor is utilized

- ▶ $\mathcal{E} = \mathcal{S}/P = \Gamma/\mathcal{W} \leq 1$

The best parallel algorithms are both *fast* and *efficient*:

- ▶ \mathcal{T} as small as possible and \mathcal{E} as close to 1 as possible

Asymptotic Notation

For the previous measures we will often use the following notation

► $f(n) = O(g(n))$ if $\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}$ s.t. $f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$

► $f(n) = \Omega(g(n))$ if $\exists c \in \mathbb{R}^+, n_0 \in \mathbb{N}$ s.t. $f(n) \geq c \cdot g(n) \quad \forall n \geq n_0$

► $f(n) = \Theta(g(n))$ if $\exists c, c' \in \mathbb{R}^+, n_0 \in \mathbb{N}$ s.t.
 $c \cdot g(n) \leq f(n) \leq c' \cdot g(n) \quad \forall n \geq n_0$

Example: Square Matrix Multiplication

Performance Analysis with $P = N \geq 2$

Naive algorithm:

- ▶ $M = N^2 + 2N = \Theta(N^2)$
- ▶ $\mathcal{T} = \mathcal{T}_N(N) = \Theta(N^2)$
- ▶ $\mathcal{W} = \mathcal{T} \cdot N = \Theta(N^3)$
- ▶ $\mathcal{W}^C = 0 \Rightarrow \mathcal{C} = 0$
- ▶ $\mathcal{S} = \frac{\Gamma}{\mathcal{T}} = \frac{O(N^{2.37})}{\Theta(N^2)} \leq \frac{\mathcal{T}_1(N)}{\mathcal{T}}$
- ▶ $\mathcal{E} = \frac{\Gamma}{\mathcal{W}} = \frac{O(N^{2.37})}{\Theta(N^3)}$

Algorithm without duplication:

- ▶ $M = 3N = \Theta(N)$
- ▶ $\mathcal{T} = 3N^2 - 2N = \Theta(N^2)$
- ▶ $\mathcal{W} = 3N^3 - 2N^2 = \Theta(N^3)$
- ▶ $\mathcal{W}^C = N^3 - N^2 \Rightarrow \mathcal{C} \rightarrow 1/3$
- ▶ $\mathcal{S} = \frac{O(N^{2.37})}{\Theta(N^2)}$
- ▶ $\mathcal{E} = \frac{O(N^{2.37})}{\Theta(N^3)}$

Networks of Processors

In the course we will use distributed memory architectures where processing units are physically organised in an **interconnection network**

- ▶ Each unit is linked to a (usually *small*) subset of the others
- ▶ Links can be *unidirectional* or *bidirectional* and are used by units to pass messages (data)

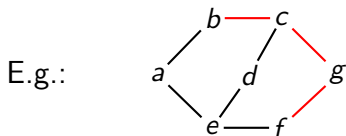
An interconnection network is a **graph** $G = (V, E)$, where nodes V are processors and edges E are links (directed if unidirectional)

- ▶ Properties and metrics of graphs can be used to evaluate a network, providing lower bounds for algorithms that run on it

Diameter

The **distance** $\text{dist}_G(u, v)$ of two nodes $u, v \in V$ is the length of the *shortest path* from u to v

The **diameter** $\text{diam}(G)$ of the graph G is the maximum distance of any pair of nodes: $\text{diam}(G) = \max_{u, v \in V} \text{dist}_G(u, v)$



$$\text{diam}(G) = 3$$

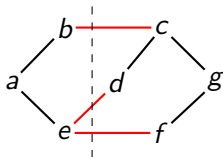
Bandwidth

Given a subset $S \subset V$ of nodes, the **bandwidth** $\delta_G(S)$ is the number of edges connecting the partitions S and $V \setminus S$ of the graph

- ▶ $\delta_G(S) = |\{e \in E \mid e \in S \times (V \setminus S)\}|$
- ▶ It provides an upper bound to the communication that can occur simultaneously between the two partitions

The bandwidth of the entire network is given by its **bisection bandwidth**

$$b(G) = \min_{S \subset V, |S| = \lfloor \frac{|V|}{2} \rfloor} \delta_G(S) = 3$$



Impact on Algorithms Performance

The diameter of the network is often a lower bound to the execution time of an algorithm

- ▶ When the information in input to or computed by a processor might be needed by any other

The bisection bandwidth is another critical factor in determining the speed with which the network can perform a calculation

- ▶ Data contained or computed by one half of the network may be needed by the other half to carry out the overall computation

Example: Square Matrix Multiplication

Interconnection Network

For the algorithm without duplication

- ▶ How many processing units should each one be linked to?
- ▶ Should links be unidirectional or bidirectional?
- ▶ How should the graph of processors be organised?

```
⋮  
3. for  $k = 1 \dots P-1$ :  
    a. sends  $B(i)$  to processor  $i+1 \bmod P$   
    b. overwrites  $B(i) = B(i-1)$  received from processor  $i-1 \bmod P$   
⋮
```