HICREST

UNITRENTO

# Introduction to Parallel Computing

a.y. 2025 – 2026

Flavio Vella

Deliverable 2

# Outline

UNITRENTO

Distributed-Memory Programming

- Introduction & Distributed-Memory Architecture
- MPI Point-to-Point Communication
- MPI Collectives & Distributed Data Patterns
- Distributed Algorithms & Performance Optimization

Algorithm design, data decomposition

Libraries and Primitives

Programming models and implementations

Run-time and Compilers

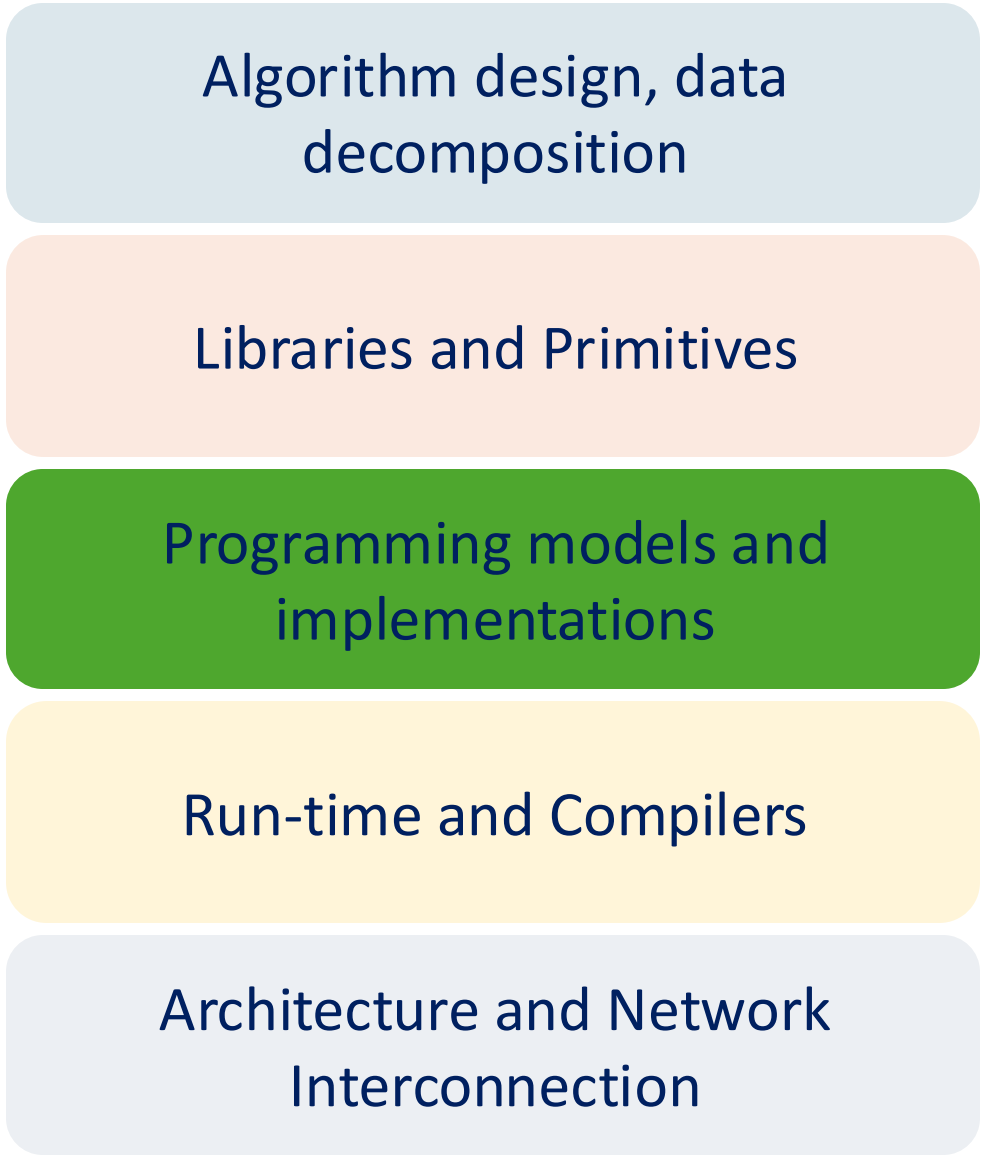
Architecture and Network Interconnection

# Outline

UNITRENTO

Distributed-Memory Programming

- Introduction & Distributed-Memory Architecture
- MPI Point-to-Point Communication
- MPI Collectives & Distributed Data Patterns
- Distributed Algorithms & Performance Optimization



# Deadline and Instructions

# Instructions

UNITRENTO

**When**

Send the report by the day before the exam sessions:

1. Jan 14 at 23.59
2. Feb 9 at 23.50

**Where and What**

1. Google Registration Form (the form will open the 8th of Jan)
2. Upload the **report Upload the deliverable 2** or **Final Project** PDF file

If you have the **project** the day after, you need to prepare 10 minute slide presentation (no need to upload the slide)

# Distributed SpMV

# Foster's Design Methodology

Foster’s Methodology is a systematic, four-stage approach to designing parallel programs.

It provides a high-level framework that guides developers from problem definition to efficient parallel implementation.

It is composed of four stages

1. Partitioning
2. Communication
3. Aggregation
4. Mapping

# Foster's Methodology Partitioning

UNITRENTO

**Break the problem into a set of small, independent tasks.**

This is the most important step: good partitioning determines whether communication will be low or high.

**Two common forms:**

1. Domain Decomposition
   Divide the data (e.g., matrix blocks, grid cells) into subdomains.
2. Functional Decomposition
   Divide work into tasks or operations (e.g., stages of a pipeline).

# Foster's Methodology Communication

UNITRENTO

**Determine how tasks exchange data.**

Questions addressed:

- Which tasks depend on each other?
- What data must be communicated?
- How often?
- What communication pattern arises?

**Examples**

- Neighbour exchanges (stencils)
- Broadcast and reduction (collectives)
- Global communication (FFT, sorting)

# Foster's Methodology Aggregation

UNITRENTO

**Combine fine-grained tasks into larger units of work.**

- Too many tasks cause communication overhead.
- Some tasks are naturally grouped.

**Agglomeration balances:**

- task granularity
- communication cost
- programmer complexity
- load balancing

**Common techniques**

- **Combine small blocks into larger tiles**
- **Merge pipeline stages**
- **Bundle halo regions**

# Foster's Methodology Mapping

UNITRENTO

Assign tasks to physical processors efficiently.

Mapping should aim to:

- Balance the workload
- Minimize communication cost
- Exploit topology (e.g., map neighbors to nearby ranks)
- Consider the network (fat-tree, torus, dragonfly)

**Principle**

Tasks that communicate frequently should be mapped close to each other.

# Deliverable 2 Overview

UNITRENTO

**Distributed Sparse Matrix–Vector Multiplication (SpMV) with MPI**

- Implement a distributed-memory SpMV using MPI
- Focus on data distribution, communication, and scalability
- Optional advanced features for bonus points

By completing this assignment, you will learn how to:

- Read and distribute sparse matrices in Matrix Market format
- Design 1D or 2D data partitioning strategies
- Implement distributed-memory SpMV algorithms
- Analyze strong and weak scalability

# Matrix-Vector Product

- The Matrix-Vector product is a special case of general matrix multiplication (GEMM), where the second operand is a vector.

- This means the output of $\mathrm{M} \cdot \vec{v}$ is a vector $\vec{c}$ such that:

$$\vec{c}_i = < M_{i,\cdot}, \vec{v} > = \sum_{j=1}^{m} M_{i,j} \cdot \vec{v}_j$$

| | | | |
|---|---|---|---|
| **0** | **1** | **2** | **3** |
| **10** | **11** | **12** | **13** |
| **20** | **21** | **22** | **23** |
| **30** | **31** | **32** | **33** |

$\times$

| |
|---|
| 1 |
| 1 |
| 1 |
| 1 |

$=$

| |
|---|
| 6 |
| 46 |
| 86 |
| 126 |

# Matrix Reading

UNITRENTO

**Baseline (required):**

- Rank 0 reads the entire Matrix Market file
- Rank 0 distributes matrix entries to all processes

**Bonus:**

- Parallel reading: each rank reads its own file chunk
- MPI-IO implementation (MPI_File_read_at_all)
- Must handle header and line-boundary alignment correctly

**Matrix Market Format can be challenging to parse and read**

https://math.nist.gov/MatrixMarket/mmio-c.html

# Data Partitioning

UNITRENTO

**Required distribution**: 1D modulo (cyclic) partitioning

- Row ownership rule: owner(i)=imodP where P is the number of Processes and i is the row index
- Each process stores all nonzeros belonging to its rows

**Bonus:**

- Local + global index representations
- **2D partitioning** (process grid)

# 1D vs 2D partitioning

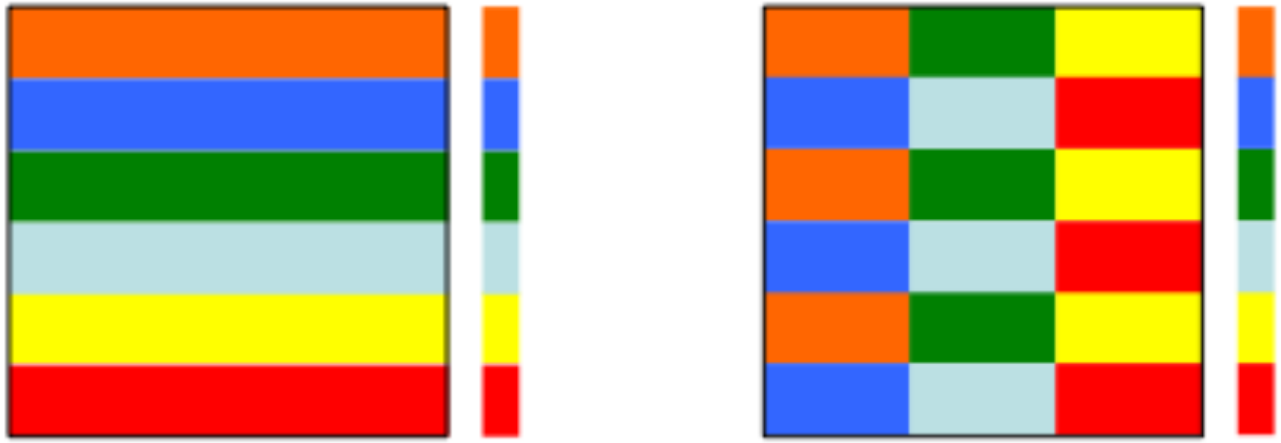


**Figure 2: 1D and 2D block layouts for 6 processes. Each color represents a process.**

Erik G. Boman, Karen D. Devine, and Sivasankaran Rajamanickam. 2013. Scalable matrix computations on large scale-free graphs using 2D graph partitioning. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13). Association for Computing Machinery, New York, NY, USA, Article 50, 1–12. https://doi.org/10.1145/2503210.2503293

# Data Structure

UNITRENTO

**Global vs Local Data Structures**

- Each process should build a **local sparse representation**, e.g.: COO → CSR conversion

Local row index:

- Compute the local index: e.g., global index (index of the entire matrix) / number of processes (integer part)

**You can use the local SpMV computation implemented for Deliverable 1**

**Bonus**

- Use Multi-threading for the local SpMV
- This paradigm is called MPI+X

**Check the paper:**
*G. Schubert et al "Parallel Sparse Matrix-Vector Multiplication as a Test Case for Hybrid MPI+OpenMP Programming*

# 1D SpMV

UNITRENTO

## Communication Pattern (1D SpMV)

Typical steps:

- Each rank computes $y_i$ for its owned rows
- Communication required to fetch remote $x_j$ entries if any

- Identify required remote column indices (“ghost” entries)
- Exchange vector values with other ranks
- Perform local SpMV
- (Optional) gather or keep distributed result

# Performance Evaluation

UNITRENTO

**Strong scaling**

- Fixed matrix size
- Increase number of processes up to 128
- Use **real matrices** (e.g., SuiteSparse)

**Weak scaling**

- Increase matrix size with number of processes
- Use **synthetic matrices (Random)**

**Bonus: Load Balance & Structure**

- NNZ per rank (min/avg/max)
- Communication volume per rank
- Structured vs unstructured matrices

# Performance Evaluation

UNITRENTO

**Discuss**

- When 1D works well
- When 2D partitioning is advantageous, if any
- When your implementation is bound by the interconnect
- What is the largest matrix size you can manage with your approach

# Metrics

UNITRENTO

**Minimum required:**

- Execution time per SpMV
- Speedup and efficiency (per rank)
- FLOPs

**Optional but encouraged:**

- Communication vs computation breakdown
- Memory footprint per rank

# Other Bonus

UNITRENTO

**Bonus: Advanced MPI Features**

Choose one:

- **Virtual topology** (Cartesian communicator for the 2D representation)
- **MPI one-sided (RMA)** for vector exchange

Explain:

- Why it helps
- How it compares to two-sided MPI

# References

Corresponding Chapters of An Introduction to Parallel Programming :

**MPI Point-to-Point Communication**

- Ch.3: Basic MPI Routines
- Ch.4: MPI Datatypes
- Ch.5: A Parallel Trapezoidal Rule
- Ch.6: Point-to-Point Communication

# References

Corresponding Chapters of An Introduction to Parallel Programming :

**Collectives & MPI Programming Patterns**

- Ch.3: Collectives
- Ch.7: Collective Communication
- Ch.8: MPI Groups and Communicators
- Ch.9: Process Topologies

**Distributed Algorithms & Performance**

- Ch.10 Parallel Sorting
- Ch.11: Graph Algorithms
- Ch.12: Matrix-Vector & Matrix-Matrix algorithms
- Ch.13: One-Sided Communication (RMA)
- Ch.14: Performance Considerations