Parallel Programming course. Parallelism practice

Obolenskiy Arseniy, Nesterov Alexander

Nizhny Novgorod State University

November 9, 2024

Contents

Full list of tasks

- Producer-Consumer Problem
- Readers-Writers Problem
- Dining Philosophers Problem
- Sleeping Barber Problem
- Broadcast (one-to-all communication)
- Reduce (all-to-one communication)
- Allreduce (all-to-one communication and broadcast)
- Scatter (generalized one-to-all communication)
- Gather (generalized all-to-one communication)
- Topology: Line
- Topology: Ring
- Topology: Star
- Topology: Torus Grid
- Topology: Hypercube
- Horizontal Strip Scheme
- Vertical Strip Scheme
- Horizontal Strip Scheme (only matrix A partitioned)
- Horizontal Strip Scheme for A, Vertical Strip Scheme for B
- Gaussian Method - Horizontal Strip Scheme
- Gaussian Method - Vertical Strip Scheme
- Gauss-Jordan Method
- Iterative Methods (Jacobi)
- Iterative Methods (Gauss-Seidel)
- Simple Iteration Method
- Bubble Sort (Odd-Even Transposition Sort)
- Image Smoothing
- Contrast Enhancement

Tasks

Tasks in the course are split into the following subgroups:

- Classical Tasks of Parallel Programming
- Data Transfer Methods
- Topologies
- Matrix Multiplication
- Systems of Linear Algebraic Equations
- Sort
- Image Processing

Disclaimers

- All matrices should be stored in linear arrays (not `std::vector<std::vector<int> >`)
- Performance should be measured on big matrices/vectors
- Total execution time (per test) is more than 1 second
- Functionality should be preserved for wide range of processes count

Classical Tasks of Parallel Programming

- Producer-Consumer Problem
- Readers-Writers Problem
- Dining Philosophers Problem
- Sleeping Barber Problem

Warning: There is different testing mechanism. Time is not measured for these tasks. Tasks are measured with big number of processes (16+, up to physical limit)

Producer-Consumer Problem

We have multiple processes: **several producers and several consumers**.
Producer produces the data, consumer needs to read it.
The producer-consumer problem is a classic synchronization scenario where one or more producers generate data and place it into a buffer, and one or more consumers remove data from the buffer for processing. The challenge is to ensure that producers do not add data into a full buffer and consumers do not remove data from an empty buffer, all while maintaining data integrity and synchronization.
In the context of MPI we can implement the producer-consumer problem by using message passing between processes.
Source: https://en.wikipedia.org/wiki/Producer–consumer_problem

Readers-Writers Problem

Some processes may read and some may write, with the constraint that no process may access the shared resource for either reading or writing while another process is in the act of writing to it.
A readers-writer lock is a data structure that solves one or more of the readers-writers problems.
Source: https://en.wikipedia.org/wiki/Readers–writers_problem

Dining Philosophers Problem

Five philosophers dine together at the same table. Each philosopher has their own plate at the table. There is a fork between each plate. The dish served is a kind of spaghetti which has to be eaten with two forks. Each philosopher can only alternately think and eat. Moreover, a philosopher can only eat their spaghetti when they have both a left and right fork. Thus two forks will only be available when their two nearest neighbors are thinking, not eating. After an individual philosopher finishes eating, they will put down both forks. The problem is how to design a regimen (a concurrent algorithm) such that any philosopher will not starve; i.e., each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think (an issue of incomplete information).
Source: https://en.wikipedia.org/wiki/Dining_philosophers_problem

Sleeping barber problem

Imagine a hypothetical barbershop with one barber, one barber chair, and a waiting room with n chairs (n may be 0) for waiting customers. The following rules apply:

- If there are no customers, the barber falls asleep in the chair
- A customer must wake the barber if he is asleep
- If a customer arrives while the barber is working, the customer leaves if all chairs are occupied and sits in an empty chair if it's available
- When the barber finishes a haircut, he inspects the waiting room to see if there are any waiting customers and falls asleep if there are none

Source: https://en.wikipedia.org/wiki/Sleeping_barber_problem

Data Transfer Methods

- Broadcast (one-to-all communication)
- Reduce (all-to-one communication)
- Allreduce (all-to-one communication and broadcast)
- Scatter (generalized one-to-all communication)
- Gather (generalized all-to-one communication)

Data Transfer Methods: details

There is no specific task for this section. You should come up with your own:

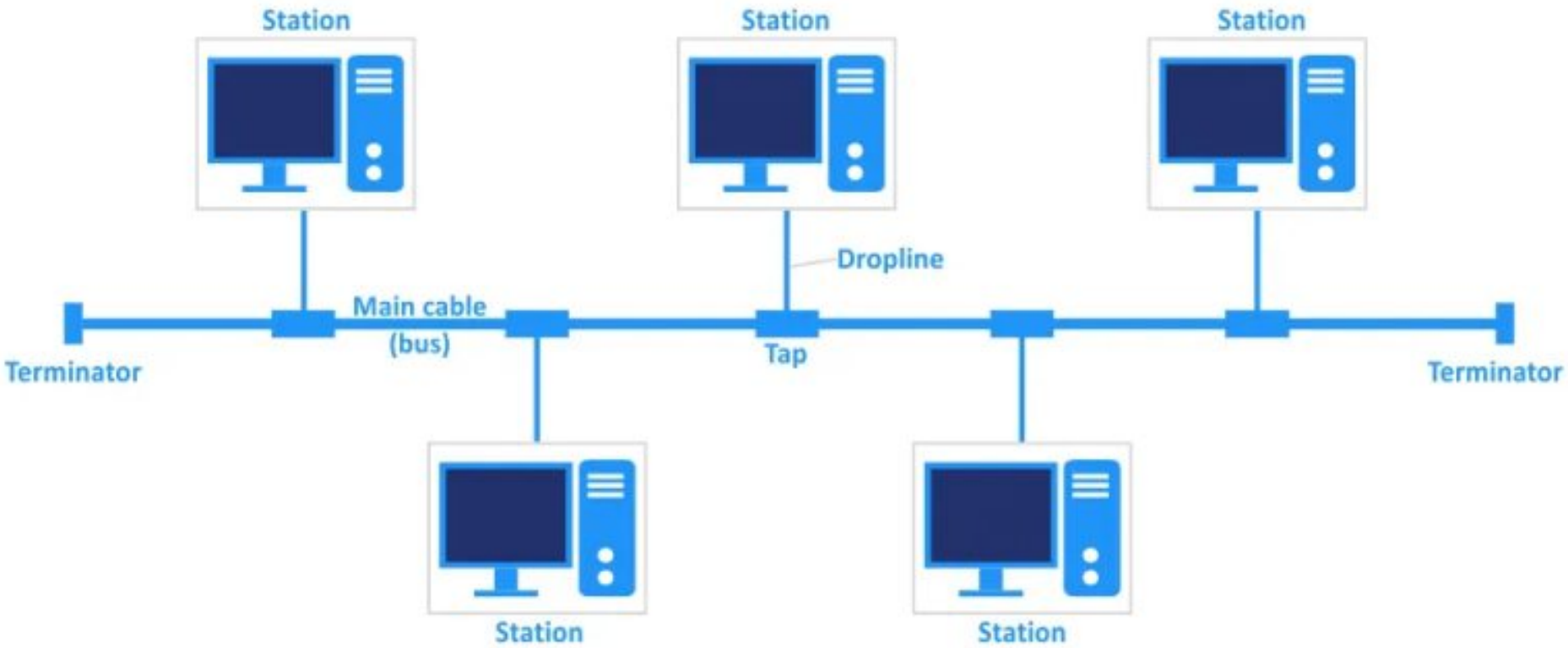
- Requirement: Put the task you have chosen in the description
- Reference implementation: original MPI function
- Tasks size should be big. Broadcast should send more that one element (vector)
- Consider using binary trees to distribute data across different processes

Topologies

- Topology: Line
- Topology: Ring
- Topology: Star
- Topology: Torus Grid
- Topology: Hypercube

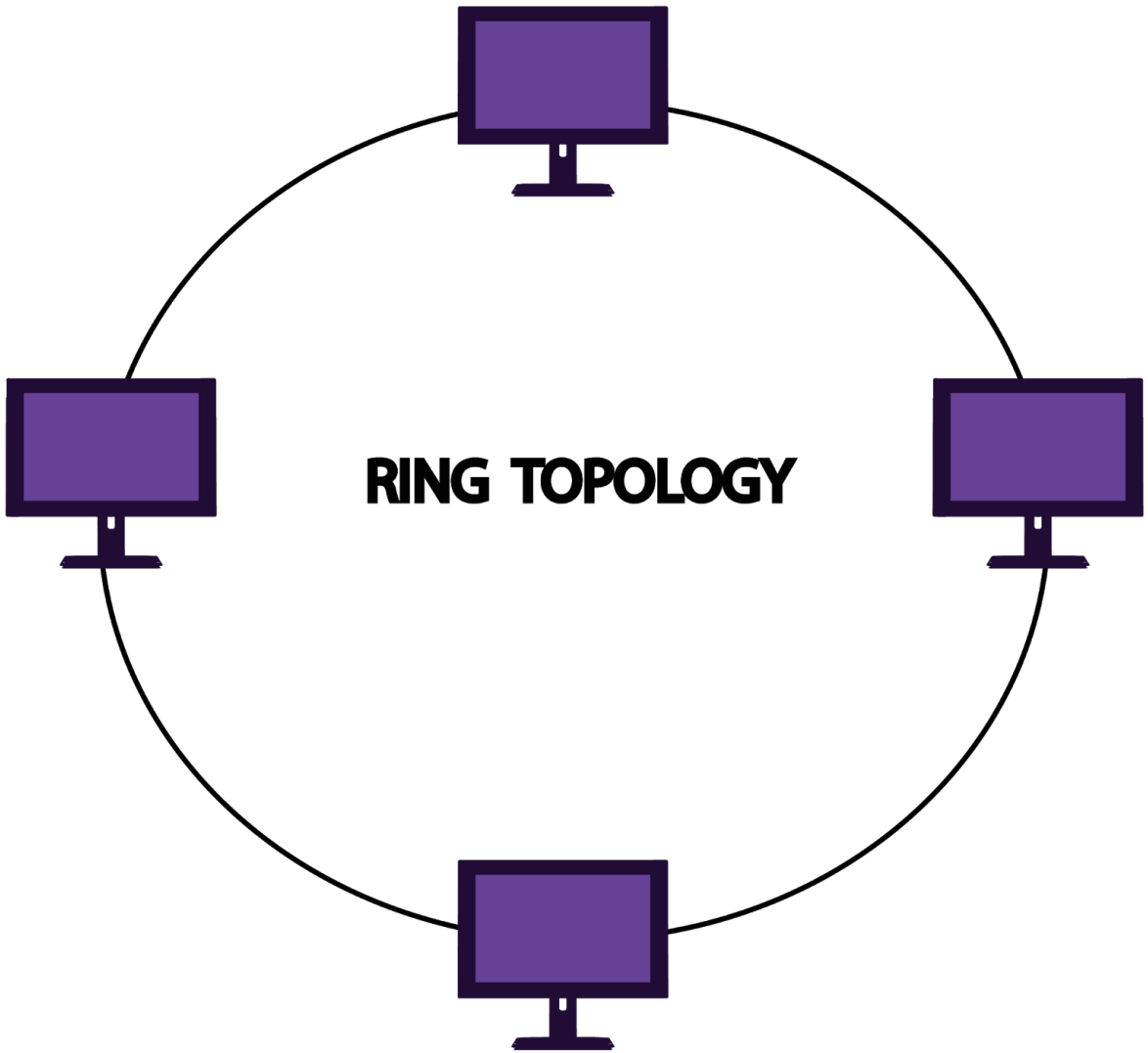
Warning: There is different testing mechanism. Time is not measured for these tasks. Tasks are measured with big number of processes (16+, up to physical limit)
Data is transferred from process 0

Line topology

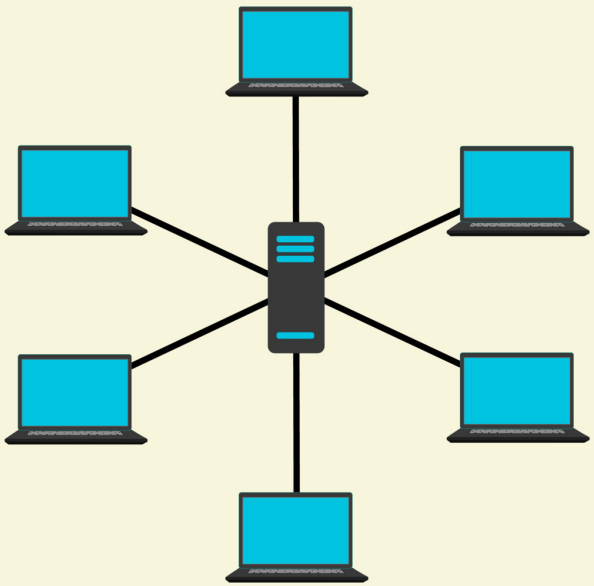


Line Topology

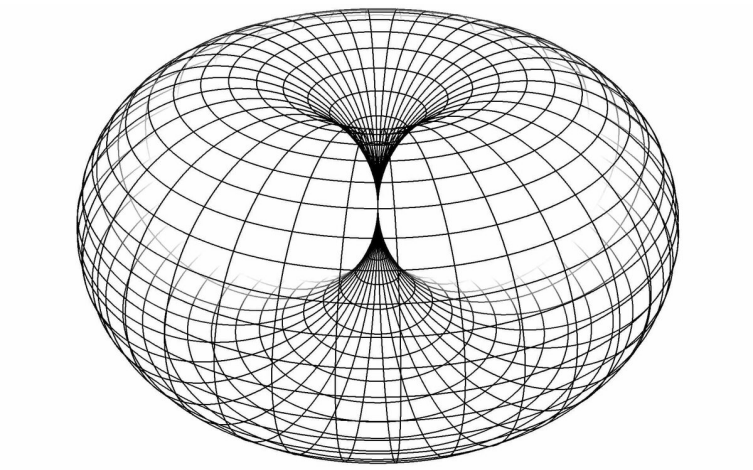
Ring topology

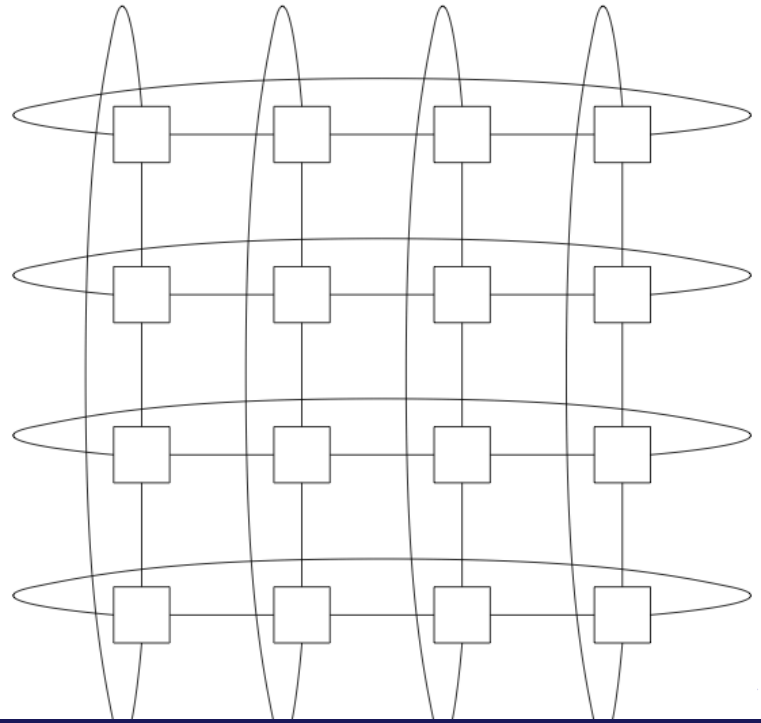


Star topology

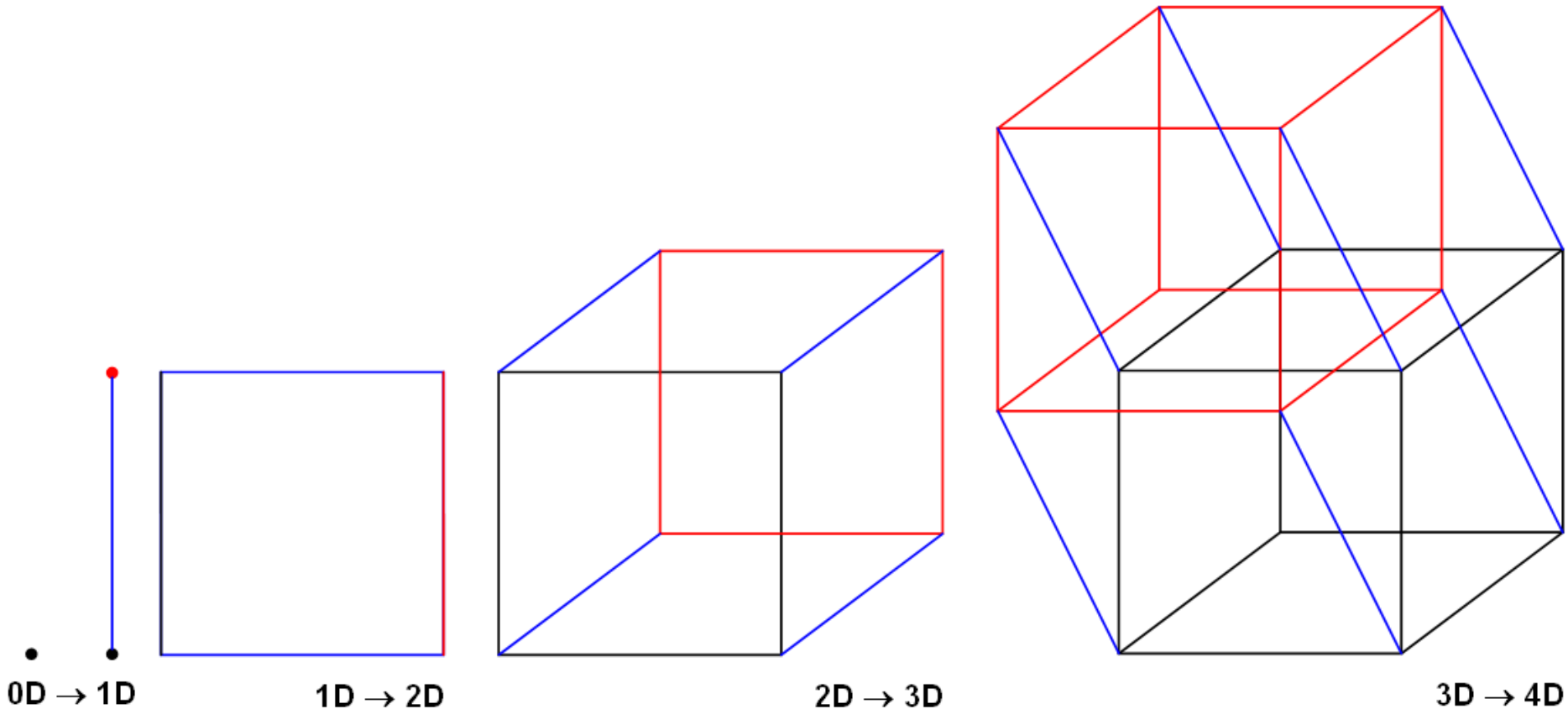


Torus Grid topology





Hypercube topology



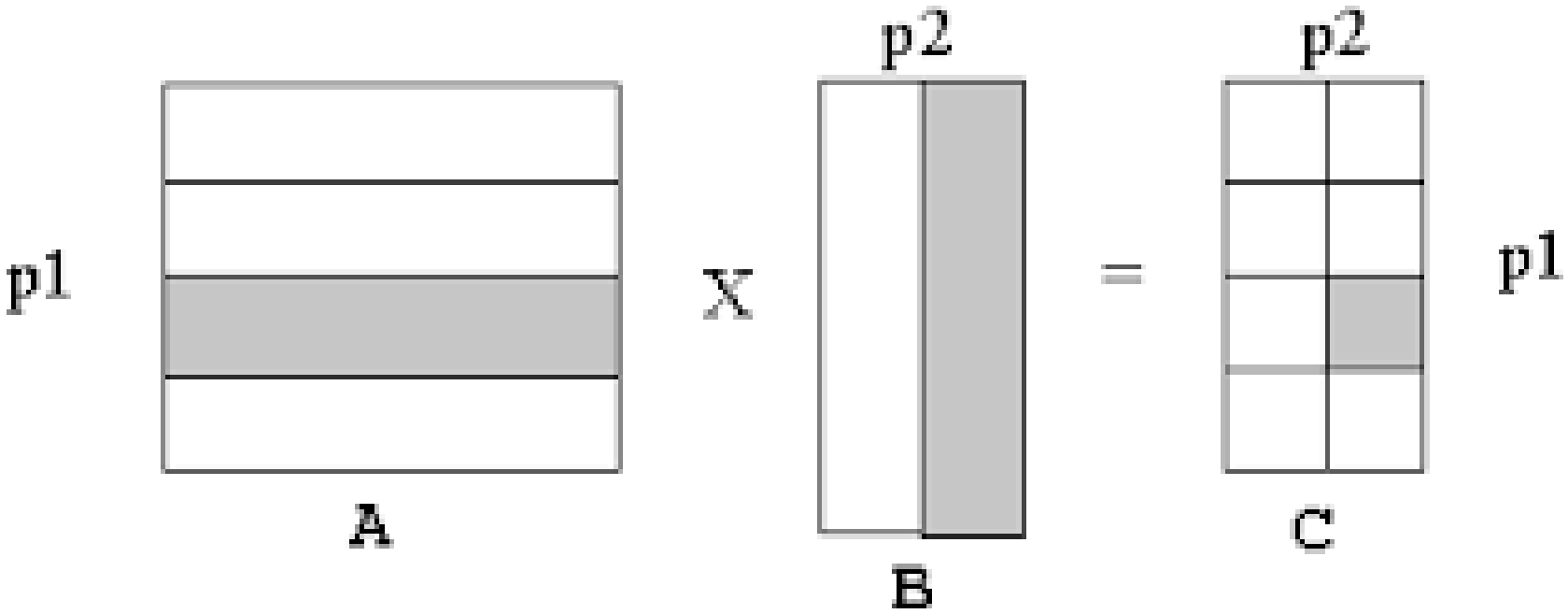
Source: https://en.wikipedia.org/wiki/Hypercube_internetwork_topology

Matrix Multiplication

- Horizontal Strip Scheme
- Vertical Strip Scheme
- Horizontal Strip Scheme (only matrix A partitioned)
- Horizontal Strip Scheme for A, Vertical Strip Scheme for B

Expected perf gain: Validate shapes (if matmul is possible for these shapes)

Matrix Multiplication



Systems of Linear Algebraic Equations

- Gaussian Method - Horizontal Strip Scheme
- Gaussian Method - Vertical Strip Scheme
- Gauss-Jordan Method
- Iterative Methods (Jacobi)
- Iterative Methods (Gauss-Seidel)
- Simple Iteration Method

Generate matrix that fits conditions Validate:

- Determinant, ... (you can use boost library for validation step)
- Use data that produces only 1 solution
- For the last 3 tasks, check convergence conditions (otherwise there will be an infinite loop)

Iterative methods: max 10%, you can parallelize only inside iteration

Sort

- Bubble Sort (Odd-Even Transposition Sort)

Notes:

- Do not use std::sort for time comparison!
- Validation of functionality is OK

Image Processing

- Image Smoothing
- Contrast Enhancement

Notes:

- Use RGB/BGR (color) linearized matrix
- Allowed to verify functionality using boost
- Compare time metric with sequential implementation

Thank You!

References