Parallel Algorithms and Programming

Collective operations in message-passing systems

Thomas Ropars

Email: thomas.ropars@univ-grenoble-alpes.fr

Website: tropars.github.io

In this lecture

- Communication models
- Logical topologies
    - Unidirectional ring

- Collective operations
    - Broadcast, Scatter, Gather, Reduce, etc.
- Algorithms for collective operations
    - Binomial tree
    - Recursive doubling
- Pipelining

Introduction

Message passing vs shared memory

In the previous lectures

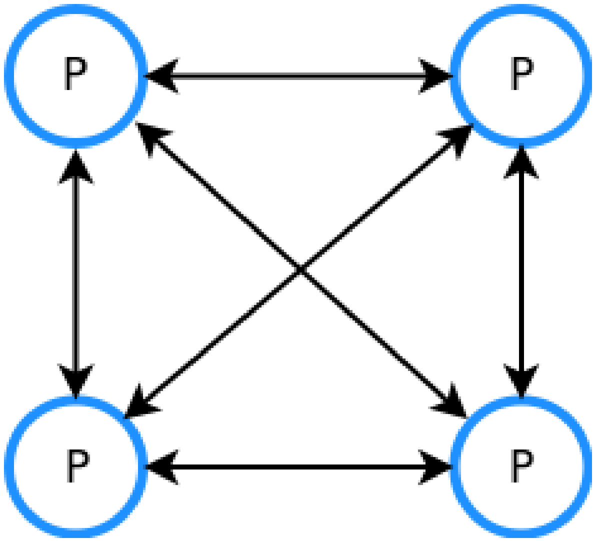
- Shared memory systems
- All processing elements have access to a shared address space

In this lecture

- Distributed memory model
- System composed of multiple nodes connected through a network

Message passing

- Each processor has access to a local private memory
  - We talk about **nodes** in message-passing systems
- The nodes communicate by exchanging messages over a network



Collective operations

- Some communication patterns are very common in message-passing parallel applications
  - These patterns should be studied and optimized

```
*Collective operations* are communication patterns that involve
groups of nodes
```

- In practice, collective operations are implemented by libraries
  - MPI (The *Message Passing Interface*)
- We will study them:
  - To understand their semantic
  - To analyze some algorithms and techniques to achieve good performance

Execution model

Need for an execution model

Objectives

- Being able to reason about the performance of an algorithm
- Capturing major performance trends

Constraints

- *A simple-enough model*: to allow reasoning about it
- *A complete-enough model*: to ensure that the designed algorithm will work well in practice
    - Capture the main characteristics of the execution platform

Our basic model

Time for transferring a message `M` of size `m` over one network link:

$$t = L + m/B$$

- `L` is the latency (or delay)
  - The time to transfer one word from the source to the destination.
  - `L` is independent of the message size.
- `B` is the bandwidth:
  - The rate at which words can be inserted on the network link.

Additional information about the model

About the communication

- **Store-and-forward** model for nodes that need to re-transmit messages
  - A node receives the full message `M` before transmitting it to the next node.
- **Full-duplex** communication channels
  - Data can transit in both directions at the same time on a network link

About the execution

- A processor can send data, receive data, and perform computation on local data at the same time

Description of Collective Operations

List of common operations

One-to-all

- Broadcast
- Scatter

All-to-one

- Gather
- Reduce

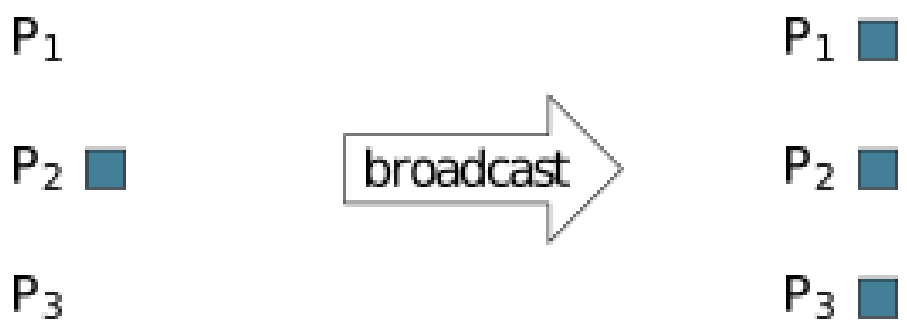
All-to-all

- All-gather / All-reduce
- All-to-all

For the description, we assume a set of `n` processors: $P_1, P_2, \ldots P_n$

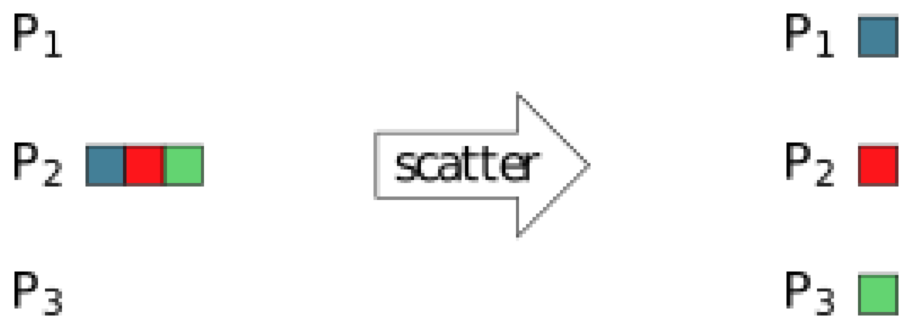
Broadcast

- *One-to-all* communication pattern.
- One processor P_k sends a message to all processors.



Scatter

- *One-to-all* communication pattern
- One processor P_k has a vector of `n` data blocks
- Each processor receives one of the blocks
- All blocks have the same size and type

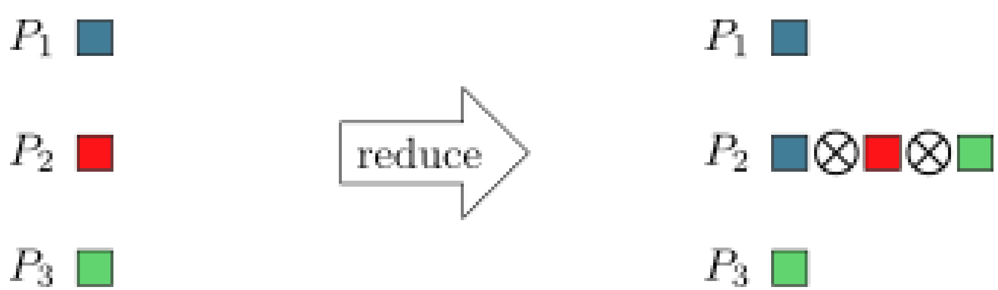


Gather

- *All-to-one* communication pattern.
- Each processor has a block of data.
- A vector of blocks is constructed on one processor P_k.
    - Block i in the vector in the block from process P_i.
- All blocks have the same size and type.

Reduce

- *All-to-one* communication pattern.
  - Also called **accumulation**
- Same as *Gather* but:
  - A reduction operation ($\otimes$) is applied to *accumulate* the data blocks
  - $\otimes$ can be min, max, sum, product, and, or, etc.



All-gather

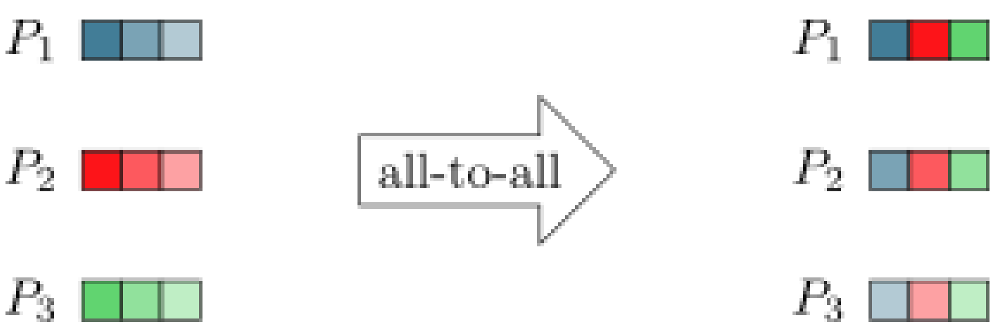
- *All-to-all* communication pattern
- Same as *Gather* except that all processors receive the constructed vector

All-reduce

- *All-to-all* communication pattern
- Same as *Reduce* except that all processors receive the computed block

All-to-all

- *All-to-all* communication pattern
- Initially:
  - Each processor has a vector of n blocks
  - Block i should be sent to processor P_i
- After the operation:
  - Each processor has a vector of n blocks
  - Block i was received from processor P_i



Implementation in a fully-connected network

Network topology

```
The *network topology* defines the distance in number of hops
between any two processors in the distributed system.
```

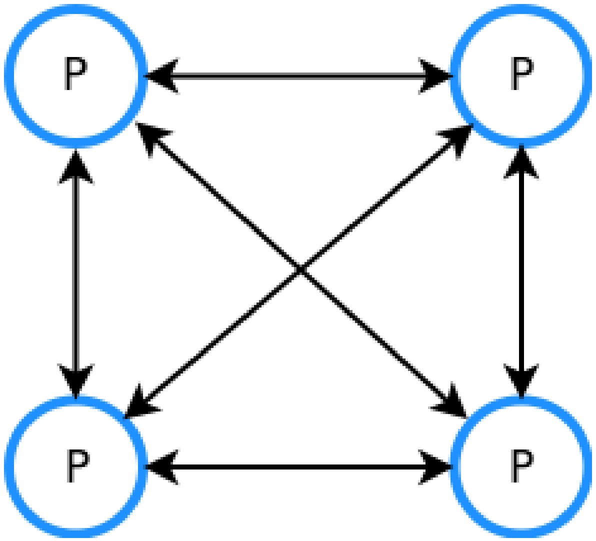
- 1-hop distance: direct link between two processors
- n-hop distance: $n - 1$ processors on the path from source to destination.

Physical vs logical topology

- **Physical topology**: The way the physical network is organized on the target platform
- **Logical topology**: An arbitrary topology that is used to reason about algorithms

Fully connected network

- Each node has a direct connection with every other nodes in the system
- We assume this is the logical topology of our network
    - Allows reasoning about the theoretically most efficient algorithms



Efficiency of collective operations

2 main metrics: latency and throughput

```
The *latency* is the time from when the collective operation is
initiated until when it is completed on all processes.

The *throughput* (or effective bandwidth) is the amount of data
that can be processed by the collective operation per time unit.
```

Recall: Our performance model for a point-to-point communication

$$t = L + m/B$$

- We assume that L and B are the same for all network links.

Broadcast algorithm

A simple algorithm

- The source sends the message to all destinations

Is it a good solution?

Broadcast algorithm

A simple algorithm

- The source sends the message to all destinations

Is it a good solution?

Time taken to transmit a message of size m, assuming a system with n nodes:

$$T_{bcast}(m) = (n-1) \times (L + \frac{m}{B})$$

Broadcast algorithm

A simple algorithm

- The source sends the message to all destinations

Is it a good solution?

Time taken to transmit a message of size m, assuming a system with n nodes:

$$T_{bcast}(m) = (n-1) \times (L + \frac{m}{B})$$

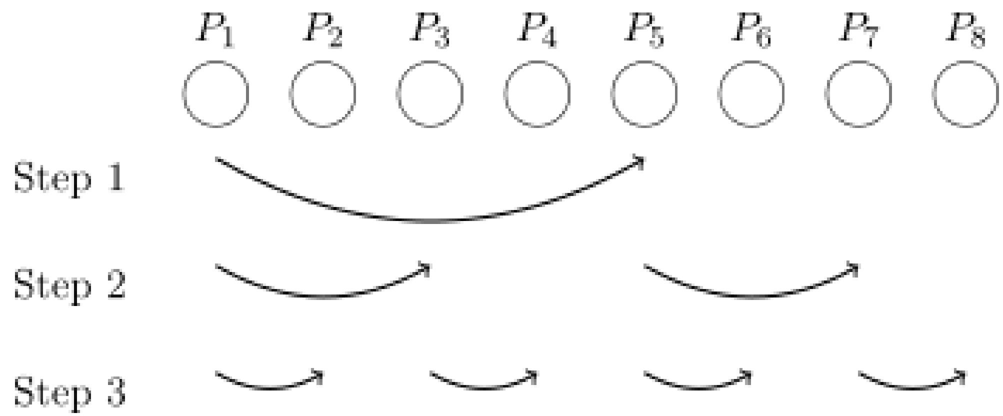
Bad performance at large scale:

- The latency and the throughput terms are proportionals to the number of nodes.

Other solutions?

Algorithm based on a binomial tree

Broadcast from P_1

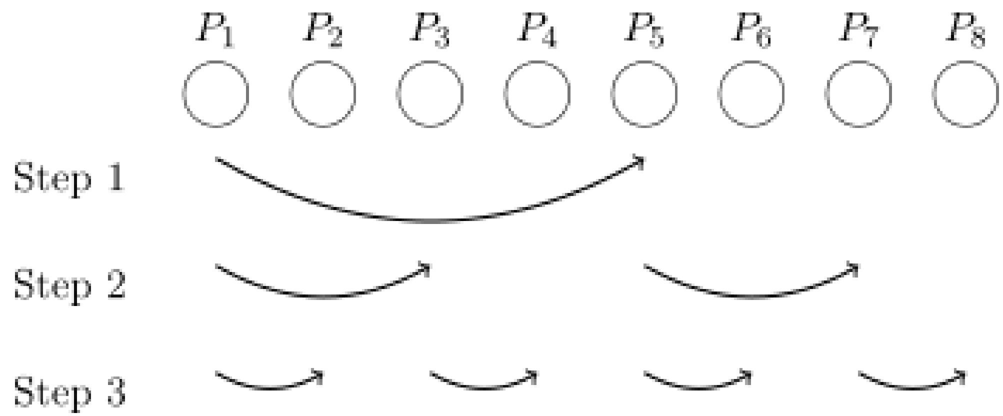


The number of *senders* is multiplied by 2 at every step of the algorithm.

Performance

Algorithm based on a binomial tree

Broadcast from P_1



The number of *senders* is multiplied by 2 at every step of the algorithm.

Performance

$$T_{bcast}(m) = log(n) \times (L + \frac{m}{B})$$

This new algorithm scales much better.

Additional comments

Considering latency

- The binomial-tree algorithm is optimal from latency point of view (assuming power-of-two number of processes)

Considering throughput

- A better solution: Algorithm by Van de Geijn *et al.*
    - Step 1: Scatter
    - Step 2: All-gather

All-gather algorithm

(Recall) Definition

- Constructs an array out of the contributed block of each node
- Result shared between all nodes

Why focusing on this operation?

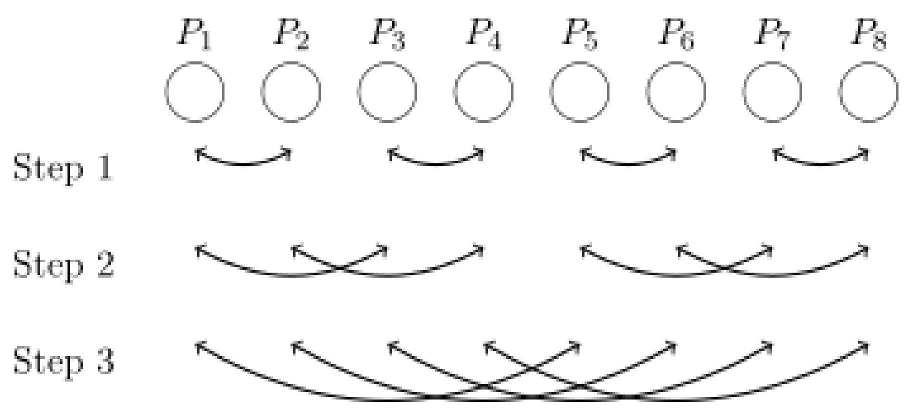
- To illustrate a strategy called **recursive doubling**
    - Used in the implementation of several collective operations
    - Complementary operation: *recursive halving*

Recursive doubling algorithm

Basic idea

At each step:

1. Each processor exchanges with another processor
2. The distance of the processor to exchange with is multiplied by 2

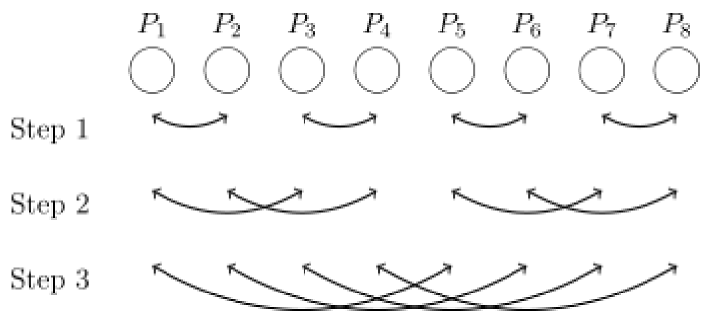


The algorithm requires $log(n)$ steps to complete

- Good at large scale

Recursive doubling algorithm

Let's check if it really works!



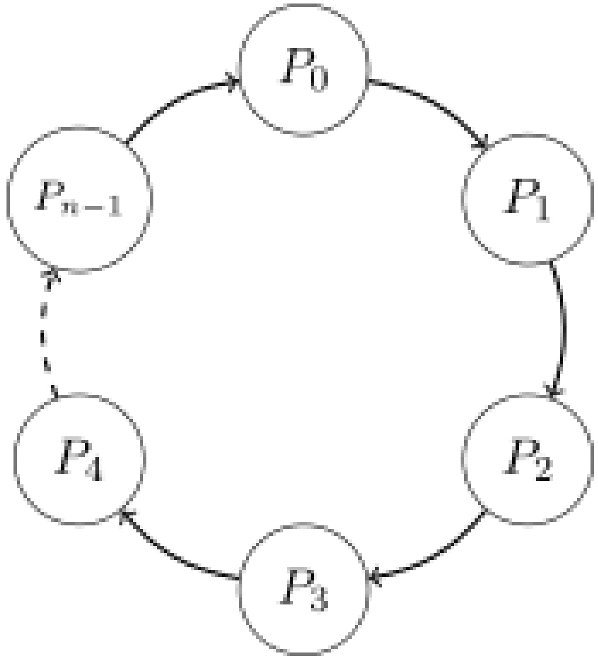
Let us consider the processor P_6:

- At step 1, P_6 receives the block of P_5.
- At step 2, P_6 receives from P_8 the blocks of P_7 and P_8.
- At step 3, P_6 receives from P_2 the blocks of P_1, P_2, P_3 and P_4.

Implementation in an unidirectional ring

A different topology

A unidirectional ring of n processors



A processor can:

- receive messages from the preceding processor
- send messages to the following processor

Broadcast algorithm

```
# broadcast of msg M, source is processor Pk
broadcast_ring(M, k):
    if my_id != k:
        Receive M from processor (my_id-1) mod n

    if my_id != k-1 mod n:
        Send M to processor (my_id+1) mod n
```

Performance

Broadcast algorithm

```
# broadcast of msg M, source is processor Pk
broadcast_ring(M, k):
    if my_id != k:
        Receive M from processor (my_id-1) mod n

    if my_id != k-1 mod n:
        Send M to processor (my_id+1) mod n
```

Performance

$$T_{bcast-ring}(m) = (n-1) \times (L + \frac{m}{B})$$

About ring-based logical topologies

Why introducing such a topology?

- Simplifies the reasoning on algorithms

Theoretical performance of ring-based algorithms

- Bad latency
    - Especially with a large number of nodes
- But with large messages, latency is not the important metric

About ring-based logical topologies

In practice

- Such algorithms can perform well in some cases
    - When the number of processors is not a power-of-two
    - When the messages are big (pipelining)

Case of distributed deep learning

- Several distributed deep-learning solutions make use of such an approach
    - All-reduce operation run with large messages
    - At most a few tens of processors
    - A ring-based algorithm works well

Improving performance through pipelining

Limit of the presented algorithm

- No parallelism is the data transfers

Solution: Pipelining

Improving performance through pipelining

Limit of the presented algorithm

- No parallelism is the data transfers

Solution: Pipelining

- Split a large message into multiple chunks on the source process
- Send the chunks one after the other

Improving performance through pipelining

Illustration:

Assuming that the source is P_0:

1. P_0 sends the first packet to processor P_1
2. P_0 sends the second packet to P_1 while
    - P_1 sends the first packet to P_2
3. P_0 sends the third packet to P_1 while
    - P_1 sends the second packet to P_2
    - P_2 sends the first packet to P_3
4. etc.

Performance of the Broadcast algorithm with pipelining

Assuming that the message is split into r packets:

Time for the first packet to reach the last node in the ring

$$(n-1) \times (L + \frac{m}{r} \times \frac{1}{B})$$

Total time for the r packets

- Observation: After the first packet has arrived, we are still waiting for $r-1$ packets
- Total execution time:

$$T_{bcast-ring-pipelined}(m) = (n+r-2) \times (L + \frac{m}{r} \times \frac{1}{B})$$

Some references

- See the lecture notes