Distributed memory computing with MPI

UROŠ LOTRIČ

Architecture: distributed memory

Multiple instructions, multiple data

Processors have their own memory

Other processors do not see memory changes

Processors exchange data by sending messages

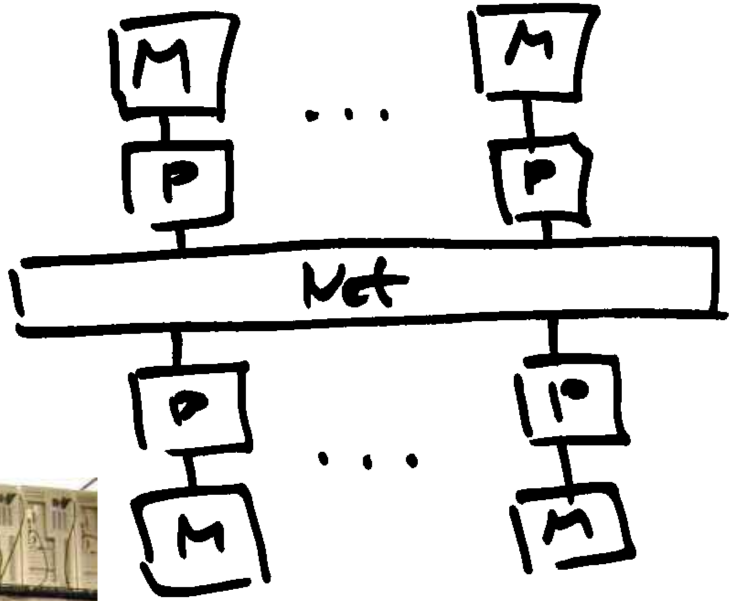
Slower compared to shared memory systems

More scalable

Focus on interconnections

Cost effective

- off-the-shelf hardware components



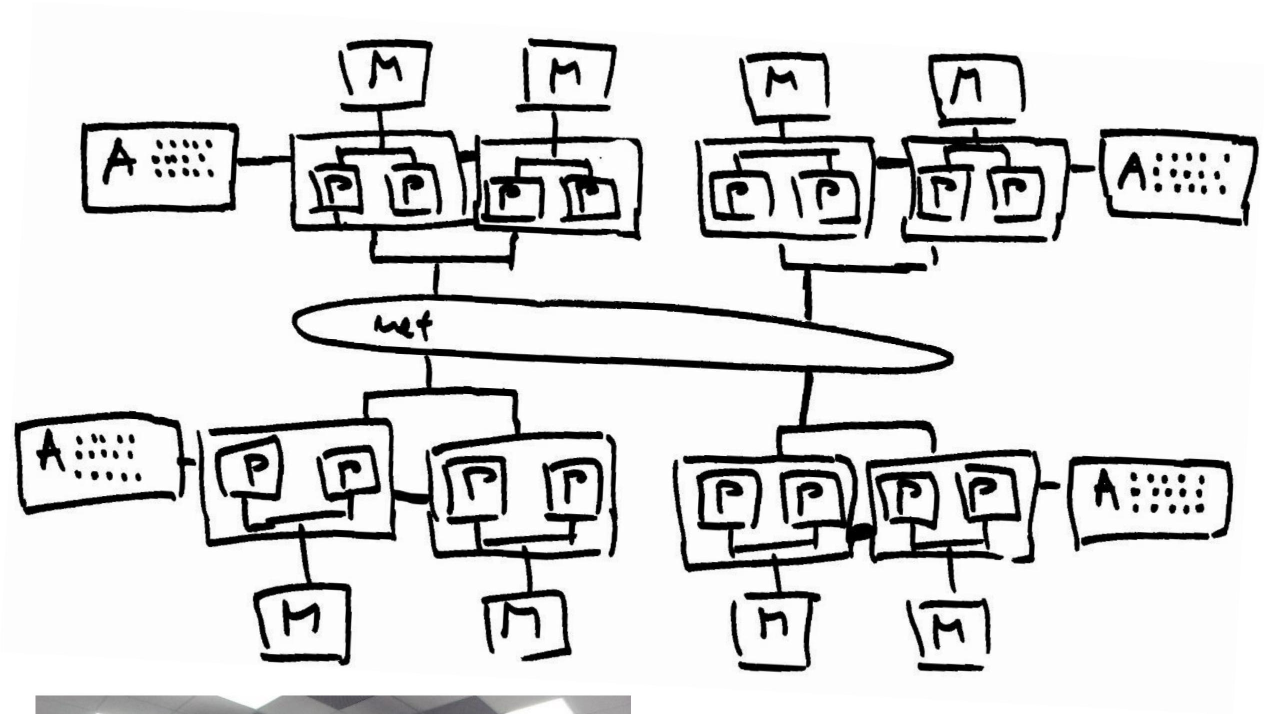
Architecture: modern systems

Modern computing resources

- Hierarchical organization
  - Connects many nodes
  - Shared memory within a node
  - Offload systems on some nodes
  - Message passing between nodes
- Heterogenous systems

Programming

- Reflects hardware organisation
- Different programming concepts
  - Programming languages
  - Libraries
  - Algorithms



Distributed memory systems

Network or interconnect is what distinguished distributed systems

- Distributed memory parallel computers are just regular computers, nodes programmed like any other
- Designing for and programming the distributed memory means thinking about how data moves between the compute nodes

Important characteristics

- How are the nodes connected together?
- How are the nodes attached to the network?
- What is the performance of the network?

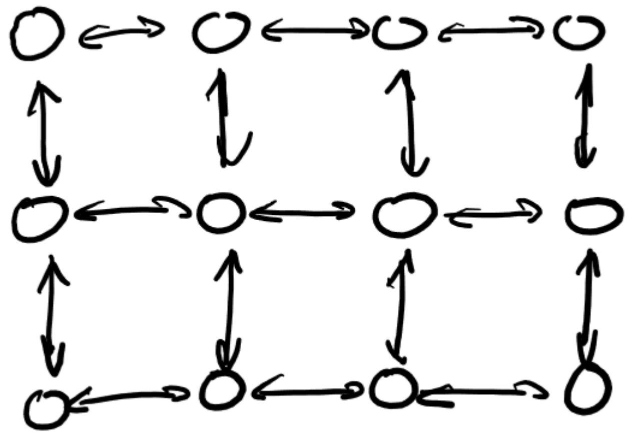
Direct and indirect topologies

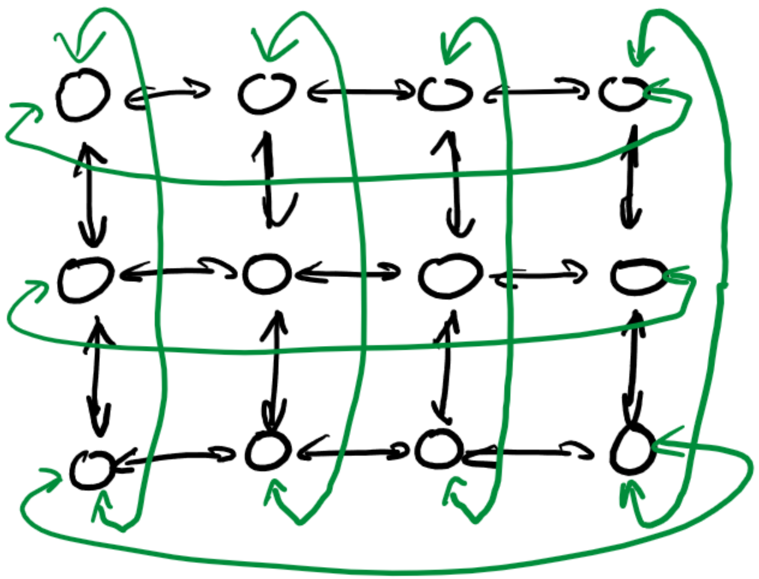
- Each node has a switch in direct topologies
- There are more switches than nodes in indirect topologies

Network

Direct topologies

- Mesh and torus
  - 1, 2, 3, and more dimensions
  - Torus links mesh ends together
  - Hypercube
- Communication is allowed among neighbouring nodes
- Constant cost to scale to more nodes
- Simple routing algorithms
- Easy to understand, simple to model
- Matches many problems well

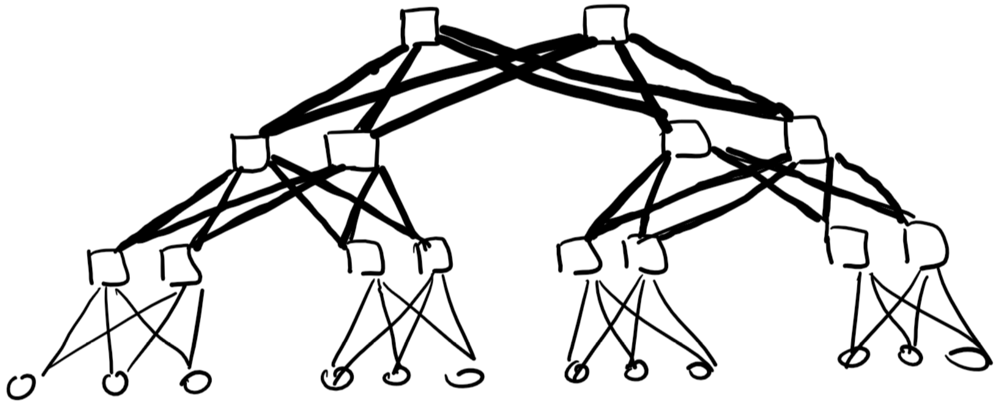


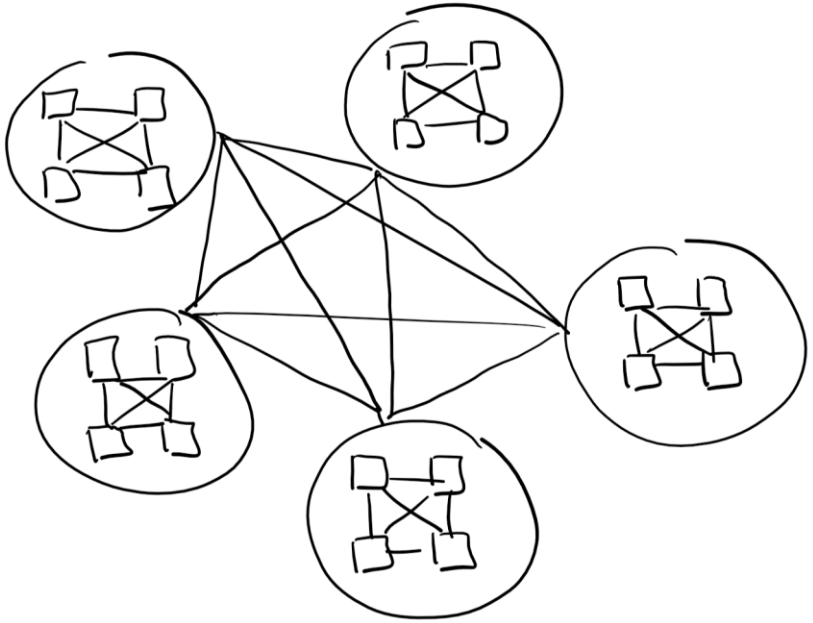


Network

Indirect topologies

- Multilevel networks
  - Fat tree network
  - Dragonfly (kačji pastir)
- Better throughput than direct topologies
  - Reduced number of hops
- Cost grows faster than linear with additional compute nodes
- More complex, harder to model

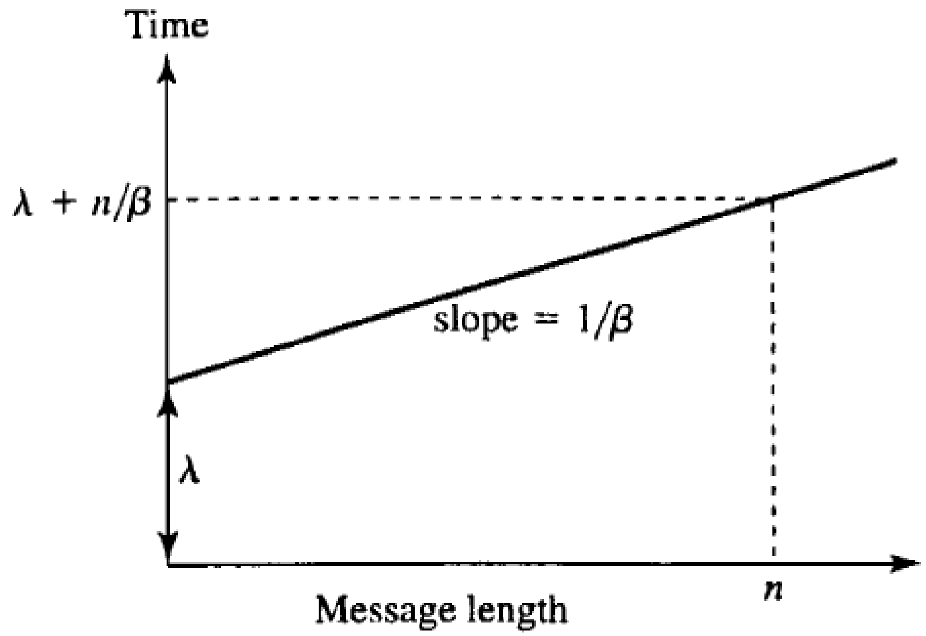




Network

Simple model

- Latency λ
- Bandwidth β
- Message length n



Improvements

- Account for number of hops
- Additional network characteristics
  - Topology, number of simultaneous connections, …

MPI

MPI

Message Passing Interface

Standard library interface specified by MPI forum

- Well recognized - each cluster has support for MPI
- All operations include routine calls

Implements message passing model

- Data transfer
- Synchronization

Support

- Official support for C/C++, Fortran
- Available other binding (Java, Python) but are not standard

MPI

Versions

- MPI-1.2 (mostly sufficient)
- MPI-2.1 (I/O, dynamic process management)
- MPI-3 (enhanced collectives, multithreaded programming, performance tools)

Implementations

- MPICH
- OpenMPI

MPI concepts

Usually we prepare one parallel program

- It consists of many processes which run in parallel

Each process has its own address space

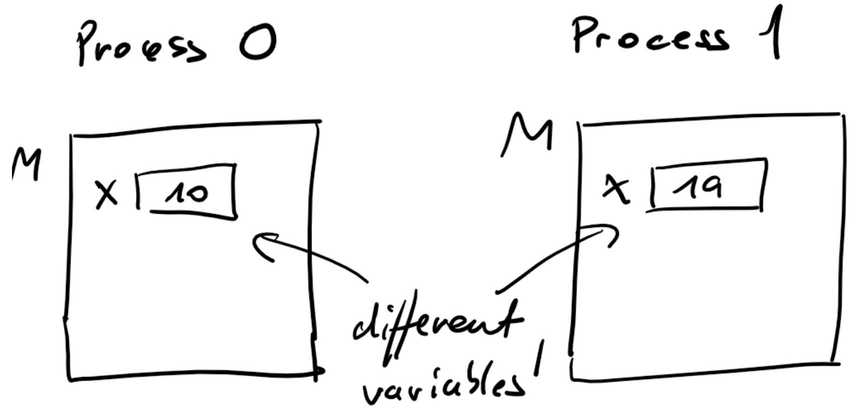
- Programmer takes care of
  data movement and placement

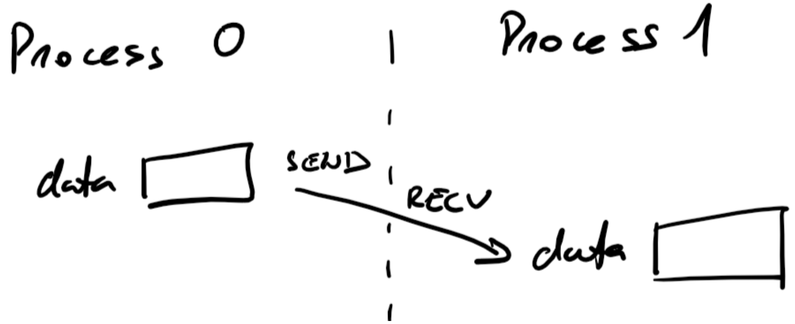
Data is sent explicitly among processes

- Programmer manages data distribution
- Programmer takes care of data transfers

Two types of transfers

- Point-to-point
- Collectives



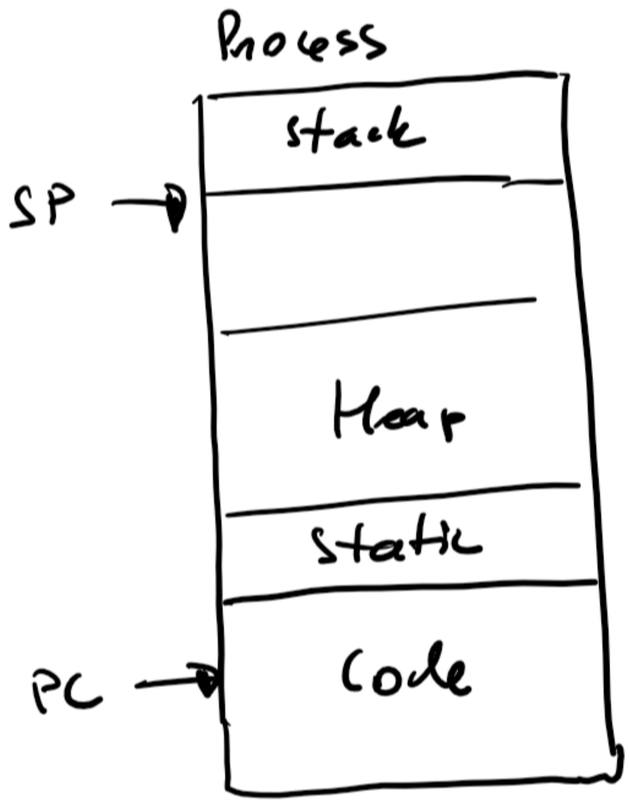


MPI concepts

Process
- Program code and program counter
- Stack and stack pointer
- Heap
- Static variables
- May have one or multiple threads sharing a single address space

MPI is for communication among processes which have separate address spaces
- Many processes can run on a single core or a single node

Inter-process communication consist od
- Data transfers
- Synchronization



MPI concepts

Processes are collected into groups

Each message is sent in a context and must be receive in the same context

Group and context form communicator

- Default communicator MPI_COMM_WORLD contains all processes
- Process inside a communicator is identified by its rank
  - Rank is a number in interval [0, *P*-1]

MPI concepts

Each message consists of

- communicator,
- source address,
- destination address,
- tag,
- data.

Each message is accompanied with user-defined integer tag to assist receiving process to identify the message

- communicator, source, destination, tag must match
- MPI_ANY_TAG does not care about tag in receiving message

MPI advantages

Recognized and standardized library

- Come implementations are free

Well understood

Tuned for all sorts of hardware

Used in many applications

Transferrable on code-level

Supports many useful functions

Quite simple to use

MPI Environment

Each process must see executable and data

- Cluster middleware (slurm) and network file system
- Processes must have permission to run over network

Executable mpirun (mpiexec) starts requested number of processes

Common approach is to use one executable on all nodes

- Differentiation is made inside the code

Programming MPI

In C/C++ we must include library “mpi.h”

Each function returns error code or MPI_SUCCESS

By default, an error causes all processes to abort

Programming MPI

Initialization

- MPI_Init(&argc, &argv)
  - MPI_Init_thread is recommended with MPI-2
    - MPI_THREAD_SINGLE has the same behaviour as MPI_Init
    - Additional arguments which take care of thread safety, needed to use OpenMP with MPI
- It must be the first MPI routine in a program

Finalization

- MPI_Finalize()
- In must be the last MPI routine in a program

Programming MPI

How many processes are running

- MPI_Comm_size(MPI_COMM_WORLD, &size)

Who am I?

- MPI_Comm_rank(MPI_COMM_WORLD, &myid)

Timing

- MPI_Wtime()
  - Returns wall-clock time in seconds
  - Difference between two points in time is only relevant
- MPI_Wtick()
  - Resolution of timer

Point-to-point communication

Point-to-point communication

Cooperative approach

- Data is explicitly send by one process and received by another
- Any change in receiver's memory is made with his participation
- Communication and synchronization are combined
- MPI_Send/MPI_Recv
- Commonly used

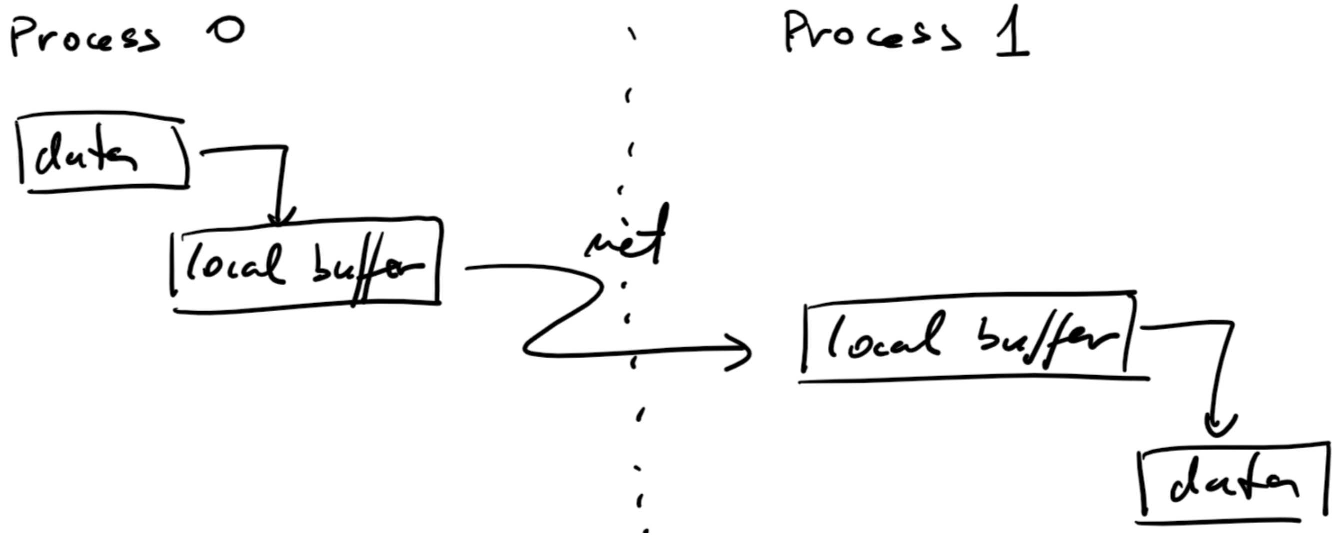
One sided operations

- Only one process takes care of data transfer
- Remote memory reads and writes
- Communication and synchronization are decoupled
- MPI_Put/MPI_Get
- Rarely used

Point-to-point communication

Buffer

- MPI_Send copies data to local buffer sending process continues executing
- MPI_Recv of receiving process gets data when it is ready
  - Some details about received message can be obtained from returned status (MPI_Status)
- Dead-lock prevention
- Smoother operation
- Limited size
  - If message is larger than buffer, buffer-less mode is used



Blocking mode

- MPI_Recv waits until it gets the message

Point-to-point communication

Synchronous (without buffer)

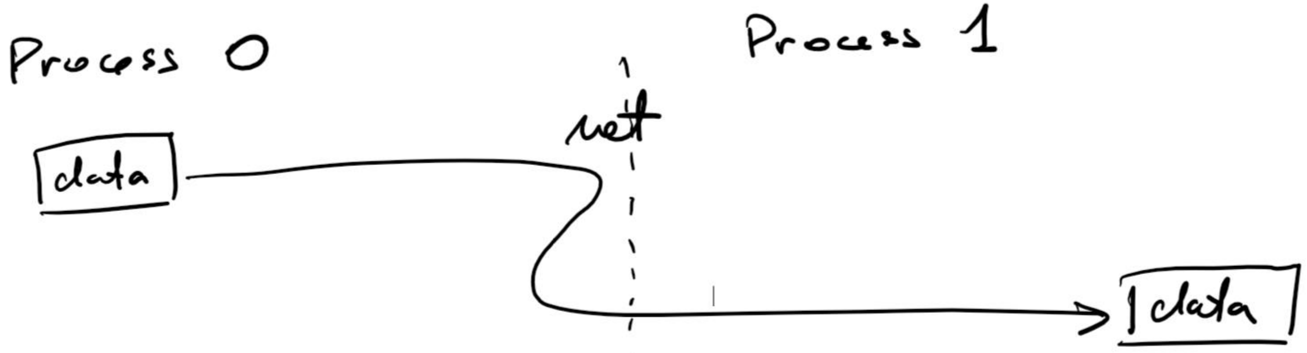
- It is better to avoid copying
- Sending process and receiving process must synchronously exchange data
- Not possible if MPI_Recv is not ready

Blocking communication

- MPI_Recv waits until in gets the message

Buffer or synchronous mode can be explicitly specified

- MPI_Bsend
- MPI_Ssend
- MPI_Recv remains the same



Point-to-point communication

Example 1

- Works in buffered mode
- Fails in synchronous mode
- Unsafe, it depends on the size of buffer

```
Process 0              |  Process 1
MPI_Send (data, 1)     |  MPI_Send (data, 0)
MPI_Recv (data, 1)     |  MPI_Recv (data, 0)
```

Example 2

- Each send has corresponding receive
- Works in both modes

```
Process 0              |  Process 1
MPI_Send (data, 1)     |  MPI_Recv (data, 0)
MPI_Recv (data, 1)     |  MPI_Send (data, 0)
```

Point-to-point communication

First example

- hello.c

Compiling on NSC

- module load mpi
- mpicc hello.c –o hello

Running on NSC

- srun --mpi=pmix -N <nodes> -n <procs> ./hello
- mpirun –n <nodes> –np <procs> ./hello

Non-blocking communication

Immediate operation

- Function returns immediately, send is performed by another thread
- On return function gives handler for testing on completion

It is safe to use

Not necessarily faster

- Lots of short messages, lots of handlers

Not necessarily concurrent/asynchronous

- Depends on MPI implementation

Non-blocking communication

Infrastructure

- Request
  - MPI_Request request
- Send/receive (one of arguments is request)
  - MPI_Isend, MPI_Irecv, MPI_Ibsend, MPI_Issend
- Inquiry (refers to request)
  - MPI_Wait, MPI_Test

Point-to-point communication

Blocking and immediate functions can be combined

In MPI function calls MPI datatypes are used for compatibility

It is not necessary that sending and receiving datatype match

Example: stencil

MPI has a rich list of functions

- MPI_Sendrecv
  - MPI_Send and MPI_Recv combined
  - Useful with stencil pattern

Conway's game of life in MPI

Collective communication

Collective communication

Involves all processes in communicator

- MPI_COMM_WORLD is default
- Can create own subsets
- MPI-2+ can create even bigger sets if dynamic process allocation is supported

Programs using only collective communication are easy to understand

- Every process does roughly the same thing
- No inventive communication patterns

Functions for collective communication are optimized

- Devised by experts
- Detailed implementation depends on infrastructure
  - Existing protocols in network infrastructure (broadcast)

Collective communication

All collective functions must be called by all processes in the communicator

Functions work with any number of processes from 1 onwards

All collective functions are blocking (MPI-1, MPI-2)

There is no tags

Basic datatypes (MPI-1, MPI-2)

Types of collectives

- Synchronization
- Data transfer
- Collective computation

Synchronization

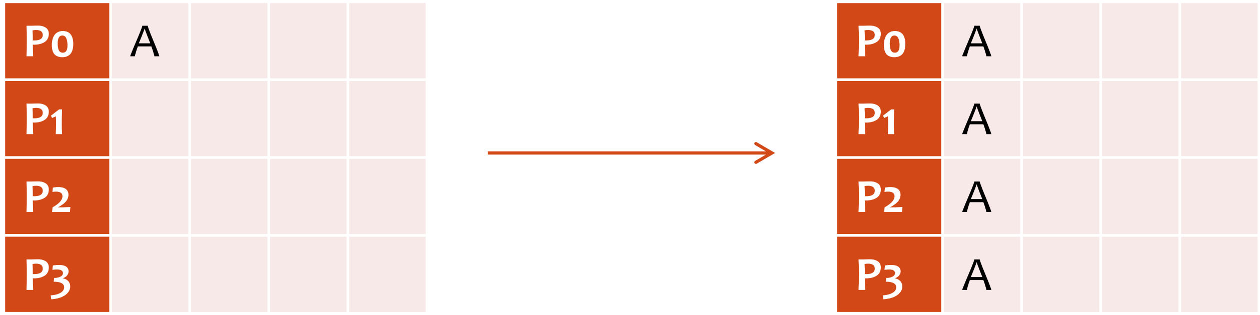
MPI_Barrier

- rarely used
- for performance measurements

Data transfer

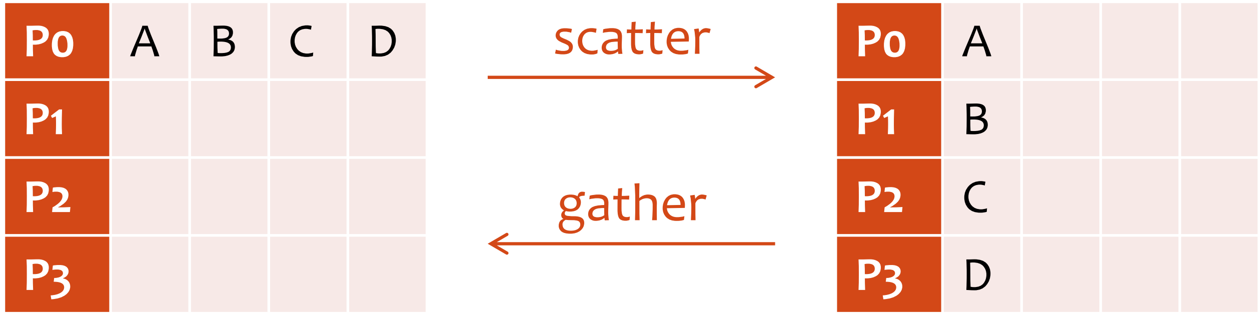
MPI_Bcast

- One to all



Data transfer

MPI_Scatter
MPI_Gather



Simple functions expect all data chunks to be of the same size

One data chunk is also on root process

Some parameters are valid on side of sender, some on side of receiver

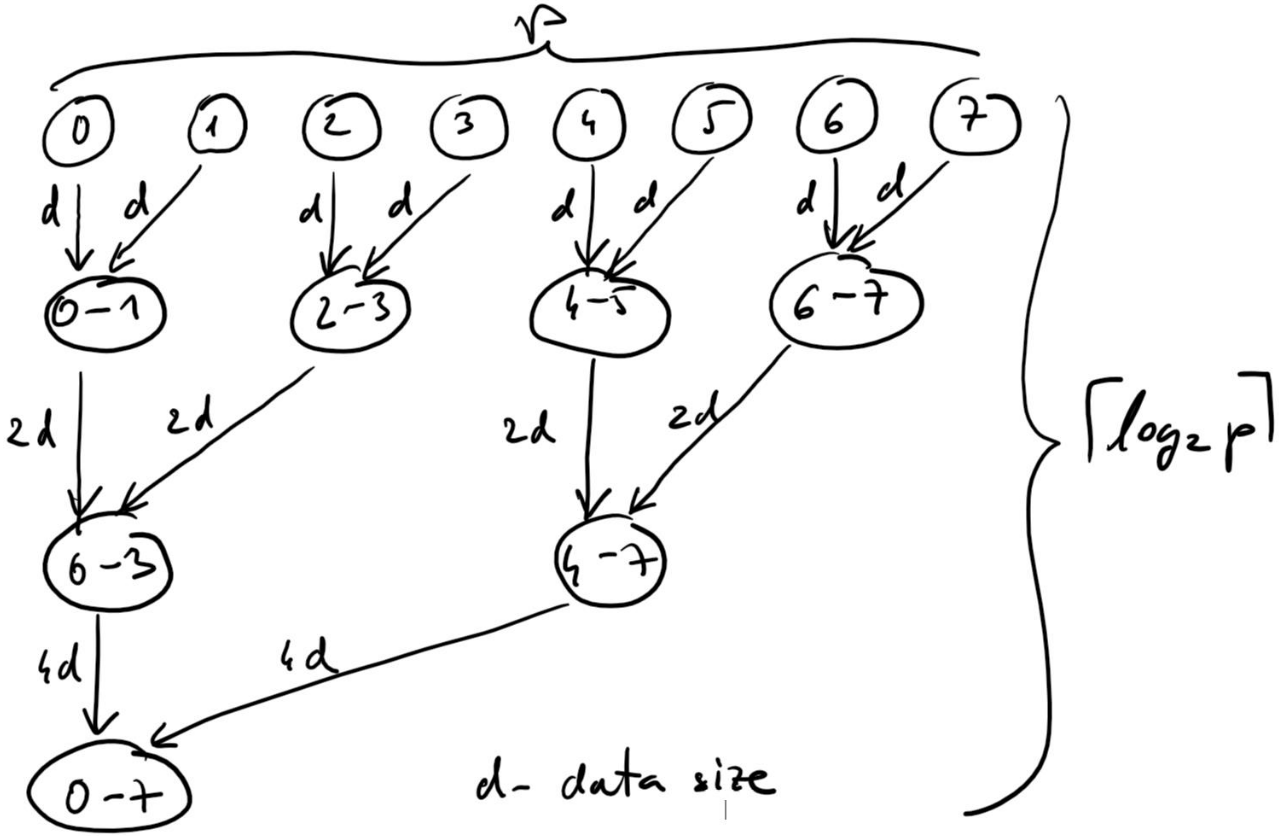
More general but slower vector functions can be used

- MPI_Scatterv
- MPI_Gatherv
- Variable size of data chunks

Data transfer

MPI_Gather

- Efficient implementation



Data transfer

MPI_Allgather

- MPI_Gather + MPI_Bcast
- Can be done in one pass of the tree

P0	A			
P1	B			
P2	C			
P3	D			

gather on all →

P0	A	B	C	D
P1	A	B	C	D
P2	A	B	C	D
P3	A	B	C	D

MPI_Alltoall

- Transpose of data
- Tricky to implement efficiently and in general

P0	A_0	A_1	A_2	A_3
P1	B_0	B_1	B_2	B_3
P2	C_0	C_1	C_2	C_3
P3	D_0	D_1	D_2	D_3

All to all →

P0	A_0	B_0	C_0	D_0
P1	A_1	B_1	C_1	D_1
P2	A_2	B_2	C_2	D_2
P3	A_3	B_3	C_3	D_3

Collective computation

MPI_Reduce

- Available reduce operations
  - MPI_MAX, MPI_MIN (minimum and maximum)
  - MPI_SUM, MPI_PROD (sum and product)
  - MPI_LAND, MPI_LOR, MPI_LXOR (logical)
  - MPI_BAND, MPI_BOR, MPI_BXOR (bit-wise)
  - MPI_MAXLOC, MPI_MINLOC (extreme value + process with extreme value)

Collective computation

MPI_Scan

P0	A
P1	B
P2	C
P3	D

→

P0	A
P1	A+B
P2	A+B+C
P3	A+B+C+D

MPI_Exscan

P0	A
P1	B
P2	C
P3	D

→

P0	Identity (0, 1)
P1	A
P2	A+B
P3	A+B+C

Collective computation

MPI_Allreduce

- MPI_Reduce + MPI_Bcast

P0	A
P1	B
P2	C
P3	D

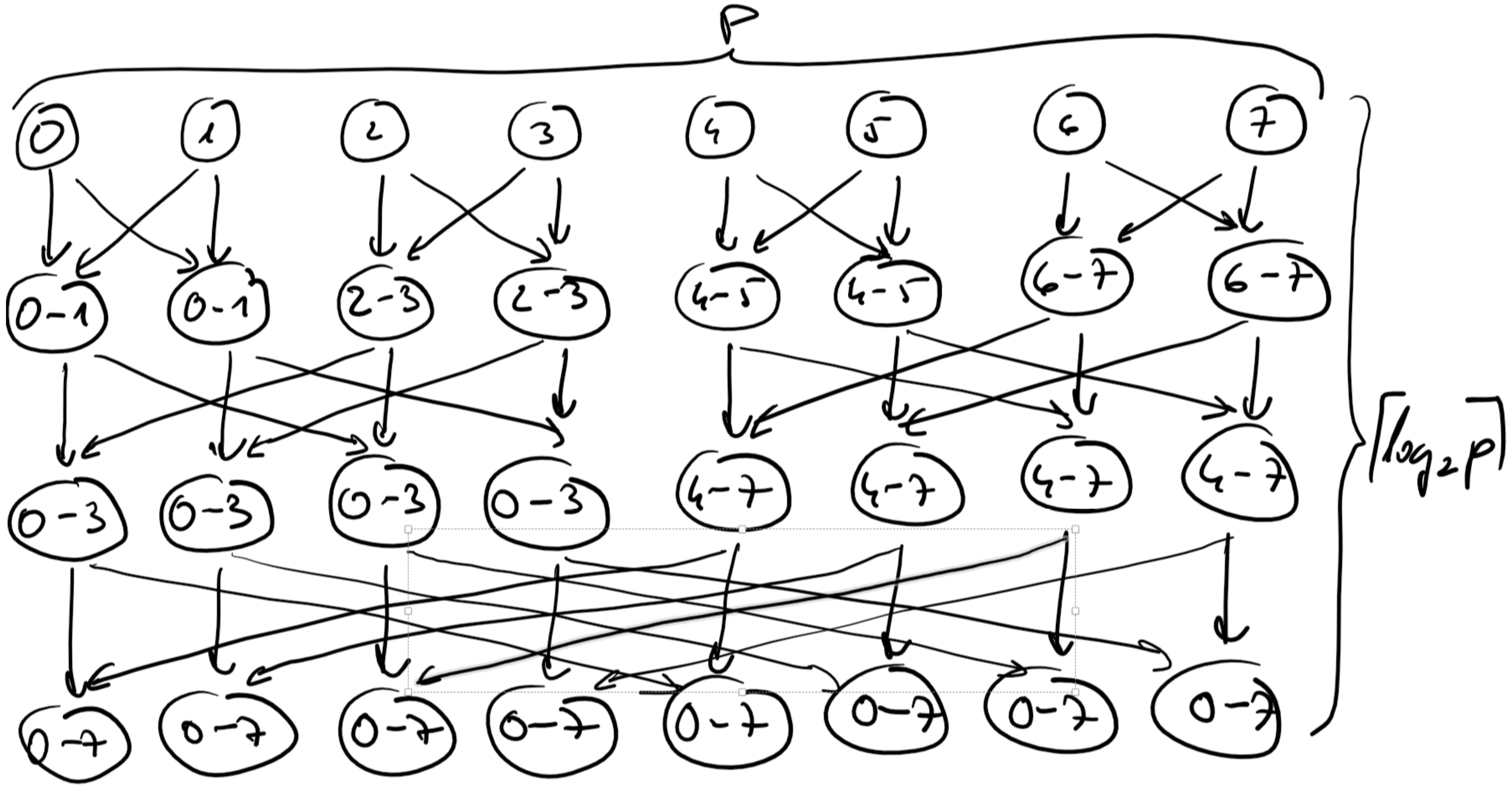
→

P0	A+B+C+D
P1	A+B+C+D
P2	A+B+C+D
P3	A+B+C+D

Collective computation

MPI_Allreduce

- Efficient computation



Collective computation

Determinism

- Roundoff error, truncation, depends on order of computation
- MPI does not guarantee the same result on the same input
  - Encouraged but not required
  - Not all applications need it
  - More efficient implementations of collectives are possible without it

Advanced features

Advanced features

Datatypes

Communicators

Virtual topology

- reflects actual system configuration
- Cartesian, graph

MPI-IO

Collective functions

- Neighbourhood
- Immediate