MPI
part 2

CSC 5001

Octobre 2018

Credits

These slides were originaly created by Patrick Carribault from CEA as part of INF560 (Algorithmique Parallèle et Distribuée) at Ecole Polytechnique. They were (slightly) adapted to fit this class format.

Lecture Outline

- MPI Collective Communication
  - Synchronization
  - Data exchange
  - Non-blocking communication

Collective Communication

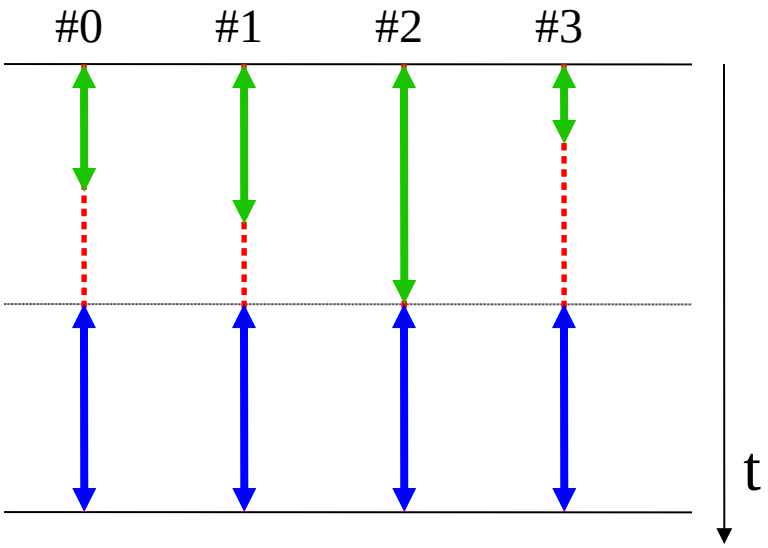
Synchronization

Barrier

- Synchronize all processes belonging to target communicator

```
int MPI_Barrier( MPI_Comm comm ) ;
```

```
MPI_Init(&argc, &argv);

/* Work 1 */

MPI_Barrier(MPI_COMM_WORLD);

/* Work 2 */

MPI_Finalize();
```



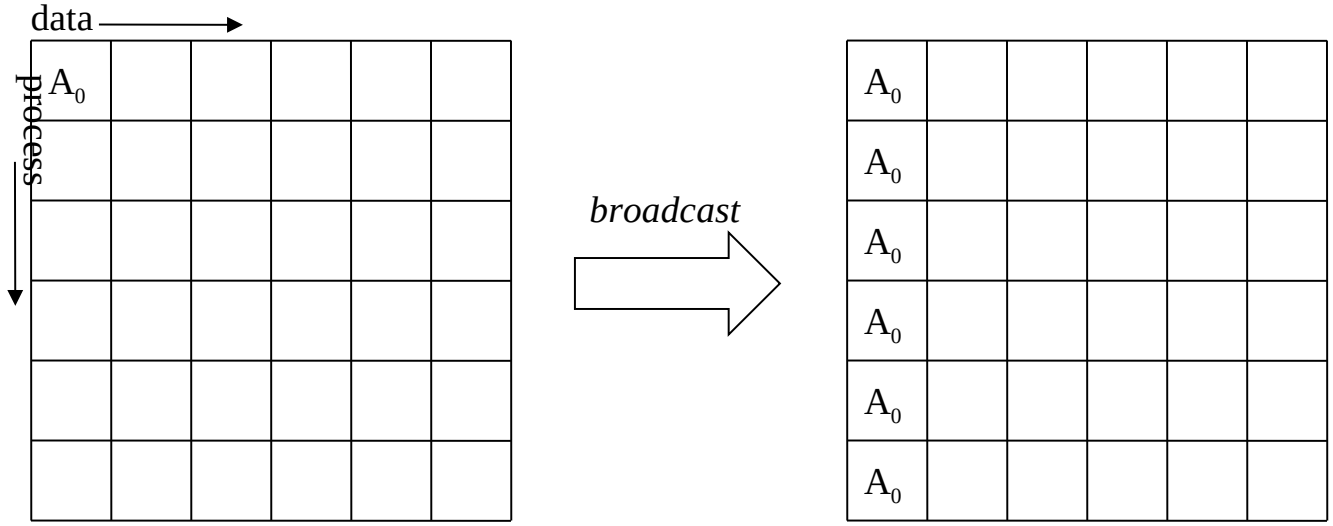
Collective Communication

Data Exchange

Broadcast

- Send data owned by one process to all other processes inside target communicator
- Process emitting data = *root*

- One-to-all collective communication



Broadcast

```
int MPI_Bcast (
        void *buf(inout),
        int count(in),
        MPI_Datatype datatype(in),
        int root(in),
        MPI_Comm comm(in),
    );
```

- rank == **root** ␣ address of memory zone to send
- rank != **root** ␣ address where to store broadcasted data

Output memory segment should be allocated by user.

Size of broadcasted data (number of elements of type `datatype`).

Broadcast

```
int MPI_Bcast (
   void *buf(inout),
        int count(in),
        MPI_Datatype datatype(in),
        int root(in),
        MPI_Comm comm(in),
    );
```

Root rank.

This rank is valid inside comm communicator.

All processes involved in this collective should have the same root.

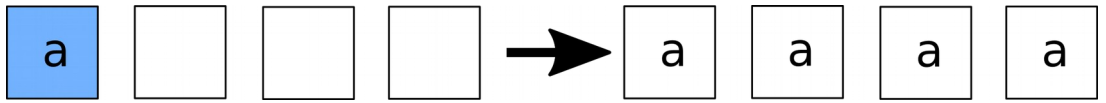
Broadcast

```
int me, root;
float pi = 0.0 ;

root = 0; /* Process 0 is the root */
…
MPI_Comm_rank(MPI_COMM_WORLD, &me);
…
if (me == root)
   pi = 3.14; /* Only root has the right initial value */

/* All processes have to call MPI_Bcast */
MPI_Bcast(&pi, 1, MPI_FLOAT, root, MPI_COMM_WORLD);

printf("P%d: pi = %f\n", me, pi);
```

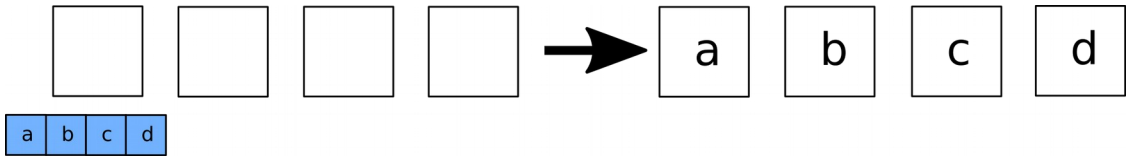
```
% mpirun –n 4 ./a.out
P0: pi = 3.14
P3: pi = 3.14
P1: pi = 3.14
P2: pi = 3.14
%
```

Collective communication

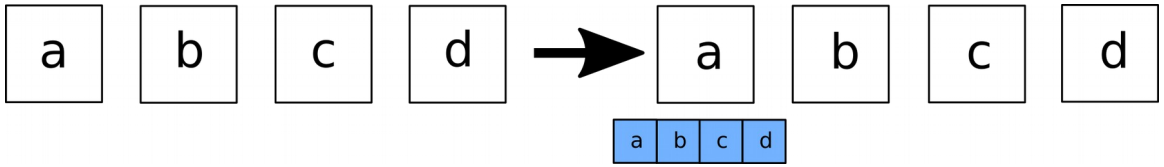
- MPI_Bcast



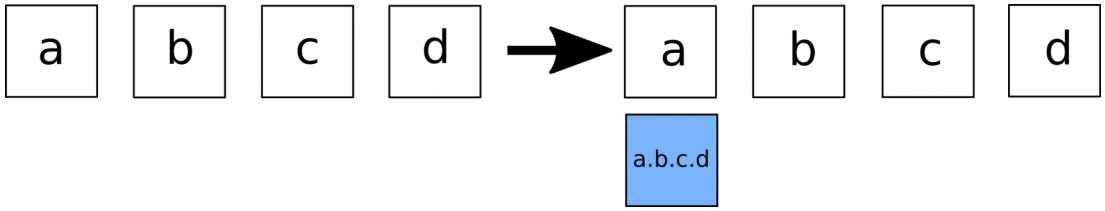
- MPI_Scatter



- MPI_Gather

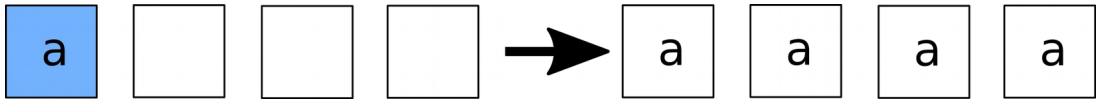


- MPI_Reduce

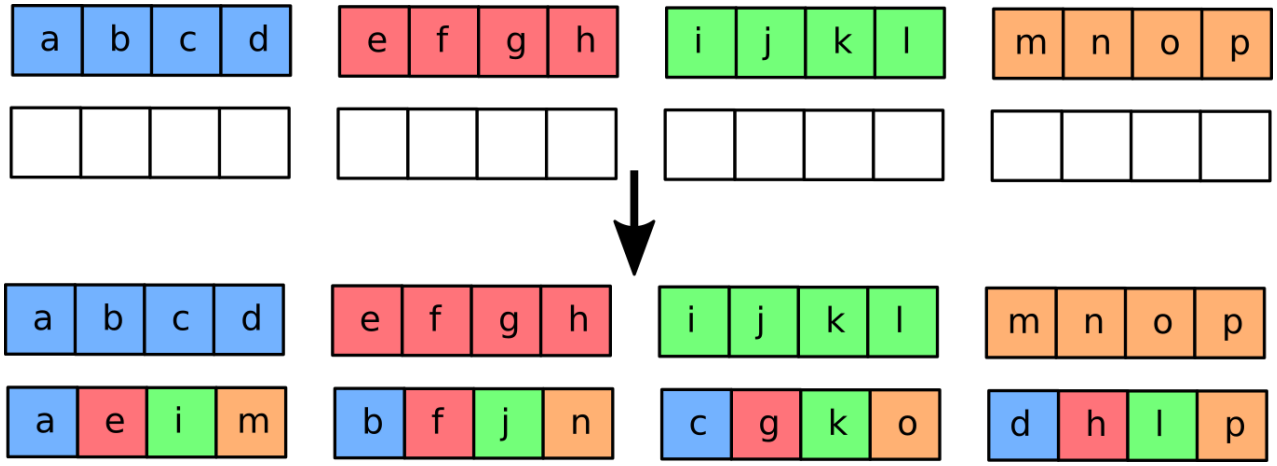


Collective communication
all-to-all broadcast

- Diffusion 1 to *n* (MPI_*Bcast*)

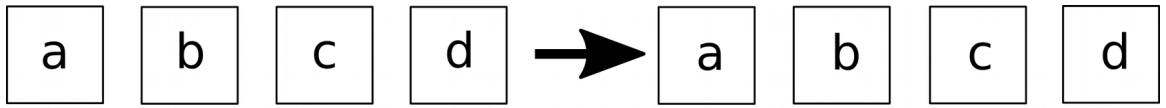


- Diffusion *n* to *n* (MPI_Alltoall)

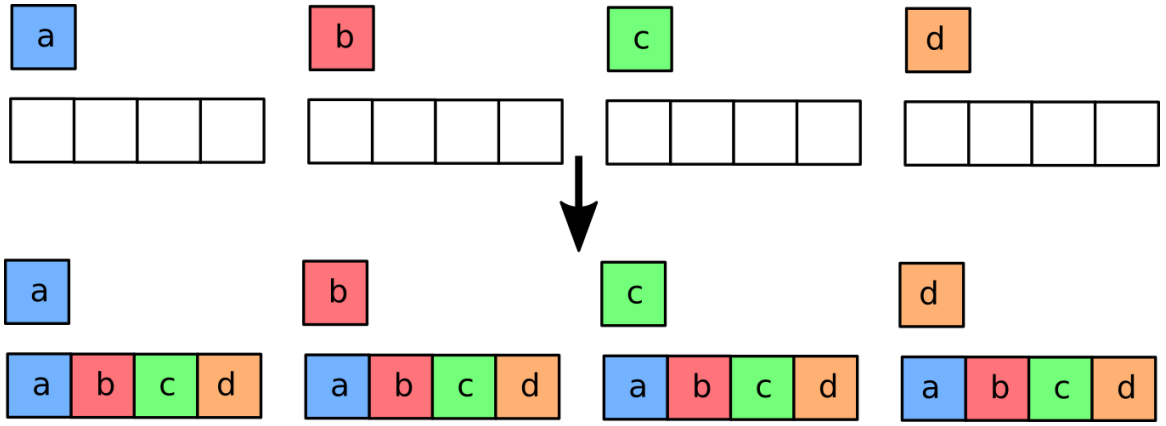


Collective communication
all-to-all gather

- Collecte *n* to 1 (MPI_*Gather*)

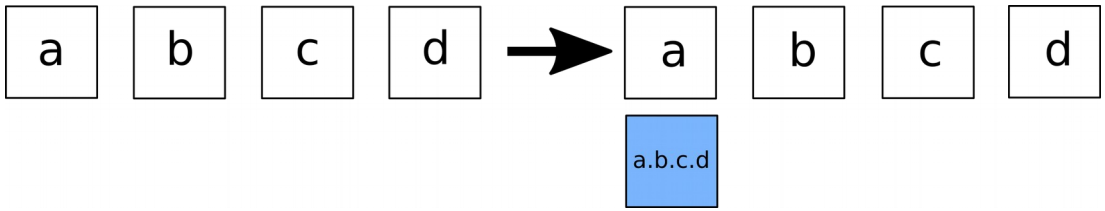


- Collecte *n* to *n* (MPI_*Allgather*)

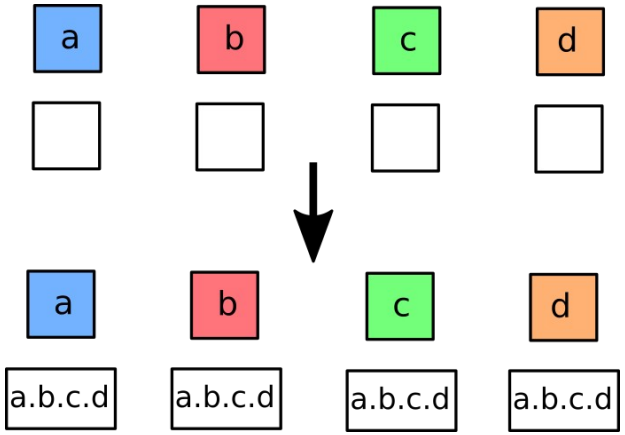


Collective communication
all-to-all reduction

- Reduction *n* to 1 (MPI_*Reduce*)



- Reduction *n* to *n* (MPI_All*reduce*)



Per-Rank Data Size

- Some collective communications propose multiple versions including one to handle different size for different ranks.
  - E.g., Broadcast, Gather
- Corresponding names have the suffix v
  - v = variant
- Examples
  - `MPI_Gather` ⫢ `MPI_Gatherv`
  - `MPI_Allgather` ⫢ `MPI_Allgatherv`
  - `MPI_Scatter` ⫢ `MPI_Scatterv`
  - `MPI_Alltoall` ⫢ `MPI_Alltoallv`

Collective Communication

Non-blocking communication

Non-blocking collective communication

- Since MPI 3.0, collective communication can be non-blocking
- Additional parameter (`MPI_Request*`) to each blocking collective function
  - eg.
    - **`int MPI_Barrier( MPI_Comm comm )`**
    - **`int MPI_Ibarrier(MPI_Comm comm, MPI_Request *request)`**
- Communication completion can be checked with `MPI_Wait`, `MPI_Test`, etc.