



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



# Message Passing Interface (MPI)

Summer School 2017 – Effective High Performance Computing

Tim Robinson, CSCS

July 19–20, 2017

# Previous course summary

- Point-to-point communication
- Blocking and non-blocking communication
- Transfer modes

# Course Objectives

- The understanding of a collective operations
- Knowledge of the different collective operations

# General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
- Topology
- Datatypes

# General Course Structure



- An introduction to MPI
- Point-to-point communications
- Collective communications
  - Collective communications
  - Barrier
  - Broadcast
  - Scatter/Gather
  - All to all
  - Reduction
  - Global collective operations
  - Non-blocking coll-op
- Topology
- Datatypes



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich

# Collective communications

---

# Collective operations

Communications involving a group of processes part of a communicator.  
Different algorithms:  $1 \rightarrow N$ ,  $N \rightarrow 1$  or  $N \rightarrow N$  ( $1 \rightarrow 1 = \text{pt2pt}$ ).

Example:

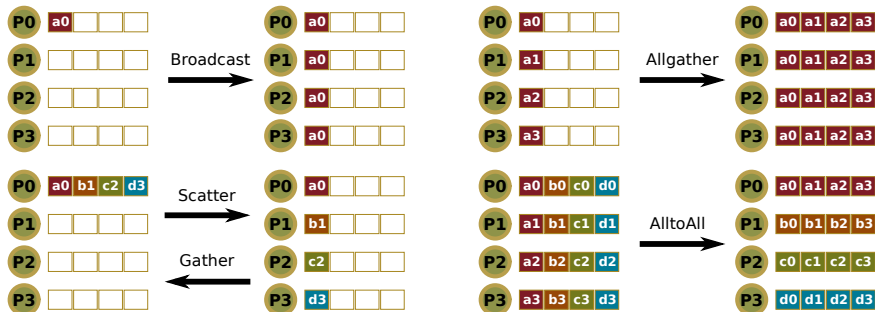
- Barrier Synchronization
- Broadcast
- Gather/Scatter
- AlltoAll
- Reduction (sum, max, prod, ...)

Features:

- All processes must call the collective routine, one is the root
- No tags

The MPI library should use the most efficient communication algorithm for the particular platform.

# Collective operations schemes



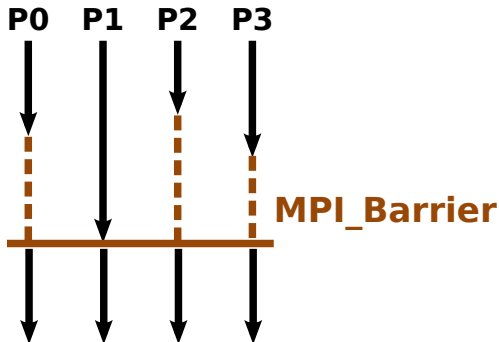


# Barrier

Stop processes until all processes within a communicator reach the barrier.

Pseudo-code

```
MPI_Barrier(comm)
```



# Broadcast

One-to-all communication: same data sent from root process to all other processes in the communicator.

Pseudo-code

```
MPI_Bcast(buf, count, type, root, comm)
```

**root**    rank being the initiator of the collective operation

# Scatter

One-to-all communication: different data sent from the root process to all other processes in the communicator.

Pseudo-code

```
MPI_Scatter(sndbuf, sndcount, sndtype,  
           rcvbuf, rcvcount, rcvtype, root, comm)
```

<b>sndcount</b>	number of elements sent to each process, <b>not the size of <code>sndbuf</code></b> , that should be <code>sndcount</code> times the number of process in the communicator
<b>rcvcount</b>	number of element in the receive buffer

The sender arguments are meaningful only for root.

# Gather

All-to-one communication: different data collected by the root process, from all others processes in the communicator.

Pseudo-code

```
MPI_Gather(sndbuf, sndcount, sndtype,  
          rcvbuf, rcvcount, rcvtype, root, comm)
```

**rcvcount** the number of elements collected from each process,  
**not the size of rcvbuf**, that should be rcvcount times  
the number of process in the communicator

**sndcount** number of element in the send buffer

The receive arguments are meaningful only for root.

# Global exchange: All to All

All-to-all communication: global exchange, all processes exchange their data. Useful for data transposition.

Pseudo-code

```
MPI_Alltoall(sndbuf, sndcount, sndtype,  
             rcvbuf, rcvcount, rcvtype, comm)
```

# Reduction

The reduction operation allows to:

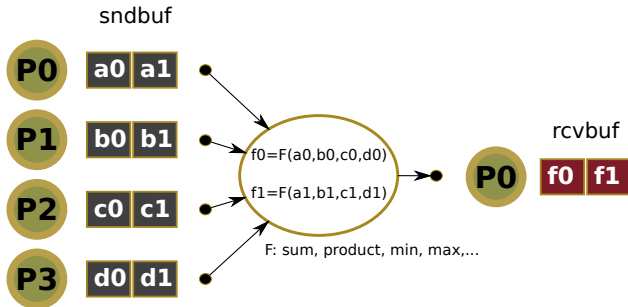
- Collect data from each process
- Reduce the data to a single value
- Store the result on the root processes
- Store the result on all processes
- Overlap communication and computation

# Reduction

Pseudo-code

```
MPI_Reduce(sndbuf, rcvbuf, count, type, op, root, comm)
```

**op** parallel operation to perform



# Reduction operators

MPI op	Operation
MPI_MAX	Maximum
MPI_MIN	Minimum
MPI_SUM	Sum
MPI_PROD	Product
MPI_LAND	Logical AND
MPI_BAND	Bitwise AND
MPI_LOR	Logical OR
MPI_BOR	Bitwise OR
MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR
MPI_MAXLOC	Maximum and location
MPI_MINLOC	Minimum and location



# Global collective operations

The result of the one-to-all operation is known by all ranks at the end of the operation.

Pseudo-code

```
MPI_Allgather(sndbuf, sndcount, sndtype,  
              rcvbuf, rcvcount, rcvtype, comm)  
  
MPI_Allreduce(sndbuf, rcvbuf, count, type, op, comm)
```

The argument **root** is missing, the result is stored in all processes.

# Non-blocking collective operations

All collective operations have a non-blocking version.

Example:

Pseudo-code

```
MPI_Ibcast(buf, count, type, root, comm, request)
```

Other functions:

Pseudo-code

```
MPI_Ibarrier, MPI_Igather, MPI_Ireduce, MPI_Iscatter,  
MPI_Iallgather, MPI_Iallreduce, MPI_Ialltoall
```

# Other functions

- Operations with different buffer sizes:

```
MPI_AlltoAllv, MPI_Gatherv, MPI_Scatterv, MPI_Allgatherv
```

- Neighbor operations, based on topology:

```
MPI_Neighbor_gather, MPI_Neighbor_alltoall
```

- Cumulative per rank reduction:

```
MPI_Scan, MPI_Exscan
```

- Create your own operator:

```
MPI_Op_create, MPI_Op_free
```

# Practicals

## Exercise: 03.MPI\_Coll

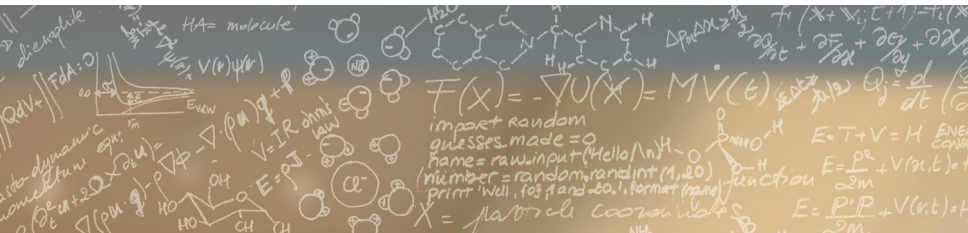
1. Read from the terminal and broadcast the input
2. Initialise an array and scatter it
3. Reduction operation
4. Reduction with results stored in all ranks (allreduce)



**CSCS**

Centro Svizzero di Calcolo Scientifico  
Swiss National Supercomputing Centre

**ETH** zürich



**Thank you for your attention.**