Communicators and Virtual Topologies
Collective Communication in MPI

Communicators

- The processes of a parallel program are united into groups.
- A process can be part of more than one group.
- The communicator in MPI is a special object, which unites within itself a group of processes and a number of complementary parameters (context) used for carrying out data transmission operations.
- Point-to-point data transmission operations are carried out for the processes that belong to the same communicator.
- Collective operations are applied simultaneously to all the processes of the communicator.

Virtual topologies

- Processes in a group can be arranged according to different topologies.
- By default, MPI considers the processes as being arranged in a one-dimensional topology.
- In many parallel programs, processes are naturally arranged in higher-dimensional topologies (e.g. a two-dimensional grid).
- Most commonly used topologies in MPI programs are one-, two-, or higher-dimensional grids, that are also referred to as **Cartesian topologies**.

Creating new communicators

- Create a communicator using a subset of the processes of an existing communicator:
  int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newcomm)
  (man page $\longrightarrow$ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Comm_split.html`)
  The function must be called by all processes in the ***comm*** communicator.

- Example: If we consider p processes organized in a $q \times q$ grid, we can create a separate communicator for processes in each row of the grid

```
MPI_Comm comm;
int rank, row;
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
row = rank/q;
MPI_Comm_split(MPI_COMM_WORLD,row,rank,&comm);
```

Working with Cartesian topologies (1)

- Create the topology:
  int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder, MPI_Comm *comm_cart)

  (man page $\longrightarrow$ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Cart_create.html`)

- Determine the Cartesian process coordinates according to its rank:
  int MPI_Cart_coords(MPI_Comm comm_cart, int rank, int maxdims, int *coords)

  (man page $\longrightarrow$ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Cart_coords.html`)

Working with Cartesian topologies (2)

- Determine the process rank according to its Cartesian coordinates:
  int MPI_Cart_rank(MPI_Comm comm_cart, int *coords, int *rank)
  (man page ⟶ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Cart_rank.html`)

- Split the grid into subgrids of smaller dimension:
  int MPI_Card_sub(MPI_Comm comm, int *subdims, MPI_Comm *newcomm)
  (man page ⟶ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Cart_sub.html`)

Data broadcasting

- ***int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int source, MPI_Comm comm)***
  - sends the data stored in the buffer buf of process source to all the other processes in the group
  - must be called by all processes in the group
  - data received by each process is stored in the buffer buf
  - the amount of data sent by the source process must be equal to the amount of data that is being received by each process

(man page → `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Bcast.html`)

All-to-one reduction

- ***int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int target, MPI_Comm comm)***
  - combines the elements stored in the buffer sendbuf of each process in the group, using the operation specified in op, and returns the combined values in the buffer recvbuf of the process with rank target
  - all the processes must call MPI_Reduce with the same value for count, datatype, op, target, and comm

  (man page $\longrightarrow$ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Reduce.html`)

Scattering data

- ***int MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, int source, MPI_Comm comm)***
  - source process sends a different part of the send buffer sendbuf to each processes, including itself
  - process $i$receives sendcount contiguous elements of type senddatatype starting from the i * sendcount location of the sendbuf of the source process
  - sendcount is the number of elements sent to each individual process
  - all the processes must call MPI_Scatter with the same value for sendcount, senddatatype, recvcount, recvdatatype, source, and comm

  (man page $\longrightarrow$ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Scatter.html`)

  see also: MPI_Scatterv

Gathering data

- ***int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, int target, MPI_Comm comm)***
  - each process, including the target process, sends the data stored in the array sendbuf to the target process
  - process $i$receives sendcount contiguous elements of type senddatatype starting from the i * sendcount location of the sendbuf of the source process
  - recvcount specifies the number of elements received by each process and not the total number of elements it receives
  - recvcount must be the same as sendcount and their datatypes must be matching

  (man page $\longrightarrow$ `http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Gather.html`)

  see also: MPI_Gatherv, MPI_Allgather, MPI_Allgatherv

All-to-all data transmission

- ***int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, MPI_Comm comm)***

  - each process sends a different portion of the sendbuf array to each other process, including itself

  - each process sends to process i sendcount contiguous elements of type senddatatype starting from the i * sendcount location of its sendbuf array

  - each process receives from process i recvcount elements of type recvdatatype and stores them in its recvbuf array starting at location i * recvcount

- MPI_Alltoall must be called by all the processes with the same values for the sendcount, senddatatype, recvcount, recvdatatype and comm

- sendcount and recvcount are the number of elements sent to, and received from, each individual process

(man page $\longrightarrow$ http://www-unix.mcs.anl.gov/mpi/www/www3/MPI_Alltoall.html)
see also: MPI_Alltoallv

Bibliography

Grama, A., Gupta, A., Karypis, G. and Kumar, V., “Introduction to Parallel Computing”, Addison-Wesley, 2003.