MPI Communicators, Groups, Topologies

Salvatore Filippone

salvatore.filippone@uniroma2.it

Reusing software

So far we have never really discussed one ubiquitous argument: `MPI_COMM_WORLD`

Communicators

A communicator is used *both* to identify a group of processes *and* to separate messages

Why do you need a communicator?

> *Suppose you have a library, and you are invoking a function; suppose also that you are sending messages. How do you ensure your messages do not interfere with the library's messages?*

The communicator is part of the *envelope*; as such, messages sent using one communicator cannot be received with another (even if it's an otherwise identical copy).

Groups and Communicators

Communicators combine the concepts of

- Group: a set of processes, each one of which has a *rank* from 0 to the size of the group (minus one);
- Context: a partitioning of the communication space, such that messages in one context are separated from those in another.

For instance, all collective communication operations are executed in (and confined to) a certain group/context.

- MPI_COMM_WORLD Predefined communicator, comprising all processes participating in the program execution; available immediately after a call to `MPI_Init`;
- MPI_COMM_SELF Predefined communicator, comprising only the calling process;
- MPI_COMM_NULL Predefined invalid communicator.

Groups and Communicators

- `MPI_Comm_group(MPI_Comm comm, MPI_Group *group)`
  returns the group associated with a communicator;

-
  ```
  MPI_Comm_dup(MPI_Comm comm,
               MPI_Comm *newcomm)
  ```
  Duplicates an existing communicator;

-
  ```
  MPI_Group_incl(MPI_Group group, int n,
                 const int ranks[],MPI_Group *newgroup)
  ```
  Creates a new group from a subset of an existing one;

Groups and Communicators

-
  ```
  MPI_Comm_create(MPI_Comm comm, MPI_Group group,
                  MPI_Comm  *newcomm)
  ```
  Creates a new communicator from group, which is a subgroup of the group associated with comm; must be called by all processes in comm;

-
  ```
  MPI_Comm_create_group(MPI_Comm comm, MPI_Group group,
                        int tag,MPI_Comm *newcomm)
  ```
  Creates a new communicator from group, which is a subgroup of the group associated with comm; must be called by all processes in group;

Virtual Topologies

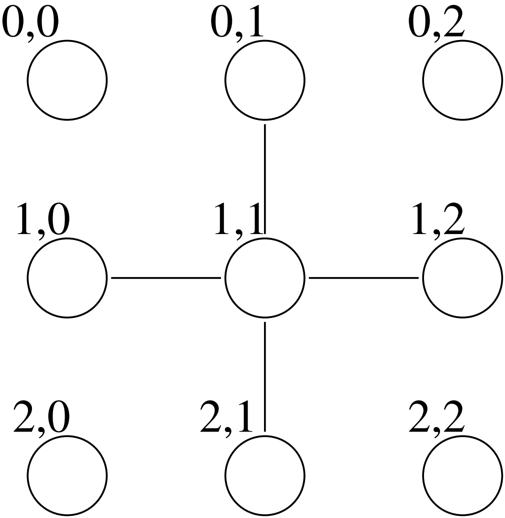
Nearest-neighbour communications are very common: they are essential in many scientific applications. MPI defines some *Neighbour collective communications*

-
  ```
  MPI_Neighbor_allgather(const void* sendbuf, int sendcount,
                         MPI_Datatype sendtype,
                         void* recvbuf, int recvcount,
                         MPI_Datatype recvtype, MPI_Comm comm)
  ```

-
  ```
  MPI_Neighbor_alltoall(const void* sendbuf, int sendcount,
                        MPI_Datatype sendtype,
                        void* recvbuf, int recvcount,
                        MPI_Datatype recvtype, MPI_Comm comm)
  ```

V variants.

Virtual Topologies

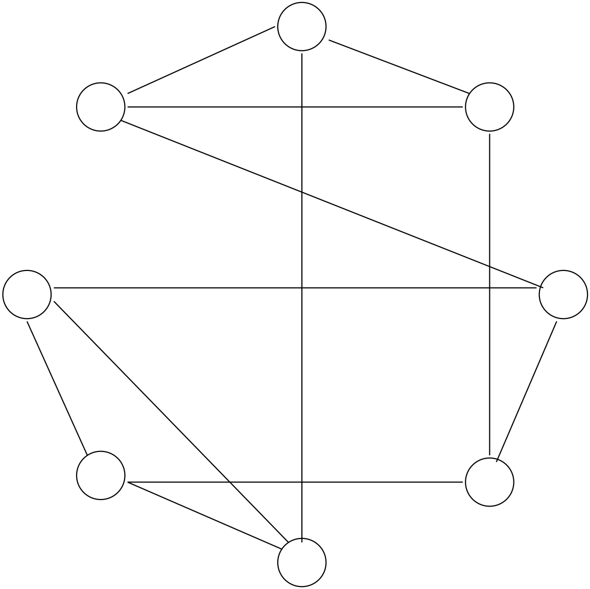


Virtual Topologies

-
  ```
  MPI_Cart_create(MPI_Comm comm_old, int ndims,
                  const int dims[],const int periods[],
                  int reorder, MPI_Comm *comm_cart)
  ```

-
  ```
  int MPI_Dims_create(int nnodes, int ndims, int dims[]),
  ```

Virtual Topologies



Virtual Topologies

-
  ```
  MPI_Graph_create(MPI_Comm comm_old, int nnodes,
                   const int index[],const int edges[],
                   int reorder, MPI_Comm *comm_graph)
  ```

-
  ```
  MPI_Dist_graph_create_adjacent(MPI_Comm comm_old, int indegree,
                  const int sources[], const int sourceweights[],
                  int outdegree,const int destinations[],
                  const int destweights[],MPI_Info info,
                  int reorder, MPI_Comm *comm_dist_graph)
  ```

Virtual Topologies

Exercises:

- implement a matrix-vector product with the matrix A distributed on a 2D cartesian topology;
- implement a matrix-matrix product with the matrices A, B and C distributed on a 2D cartesian topology;
- Implement a Jacobi sweep on a distributed cartesian mesh;