Communicators

Ned Nedialkov

Department of Computing and Software
McMaster University, Hamilton, Ontario
Canada

Outline

- Communicators, groups, contexts
- When to create communicators
- Some group and communicator operations
- Examples

Communicators, groups, contexts

- Processes can be collected into groups
- A group is an ordered set of processes
  - Each process has a unique rank in the group
  - Ranks are from 0 to $p-1$, where p is the number of processes in the group
- Each message is sent in a context, and must be received in the same context
- A communicator consist of a
  - group
  - context
- Every communicator has a unique context and every context has a unique communicator

Communicators, groups, contexts. Cont.

- A process is identified by its rank in the group associated with a communicator
- `MPI_COMM_WORLD` is a default communicator, whose group contains all initial processes
- A process can create and destroy groups at any time without reference to other processes—local to the process
- The group contained within a communicator is agreed across the processes at the time when the communicator is created
- Intra-communicator is a collection of processes that can send messages to each other and engage in collective communications
- Inter-communicator are for sending messages between processes of disjoint intra-communicators

When to create a new communicator

- To achieve modularity; e.g. a library can exchange messages in one context, while an application can work within another context

  Use of tags is not sufficient, as we need to know the tags in other modules

- To restrict a collective communication to a subset of processes
- To create a virtual topology that fits the communication pattern better

Some group and communicator operations

```
int MPI_Comm_group (MPI_Comm comm,
                    MPI_Group *group)
```

Returns a handle to the group associated with comm

```
int MPI_Group_incl(MPI_Group group, int n, int *ranks,
                   MPI_Group *new_group)
```

Creates a new group from a list of processes in old `group`

The number of processes in the new group is `n`

The processes to be included are listed at `ranks`

Process `i` in `new_group` has rank `rank[i]` in `group`

```
int MPI_Comm_create(MPI_Comm comm, MPI_Group new_group,
                    MPI_Comm *new_comm)
```

Associates a context with `new_group` and creates `new_comm`

All the processes in `new_group` belong to the group underlying `comm`

This is a collective operation

All process in `comm` must call `MPI_Comm_create`, so all processes choose a single context for the new communicator

```
int MPI_Comm_split(MPI_Comm comm, int color,
                   int key, MPI_Comm *comm_out)
```

Partitions the group associated with `comm` into disjoint subgroups, one for each value of `color`

Each subgroup contains all processes marked with the same color

Within each subgroup, processes are ranked in order defined by the value of `key`

Ties are broken according to their rank in the old group

A new communicator is created for each subgroup and returned in `comm_out`

Although a collective operation, each process is allowed to provide different values for `color` and `key`

The value of `color` must be greater than or equal to 0

Example

```
/* comm.c */
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"

#define NPROCS 8

int main(int argc, char *argv[])
{
  int rank, new_rank,
    sendbuf, recvbuf,
    numtasks;
  int ranks1[4]={0,1,2,3};
  int ranks2[4]={4,5,6,7};

  MPI_Group orig_group, new_group;
  MPI_Comm new_comm;

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
```

```
if (numtasks != NPROCS && rank==0)
  {
    printf("Must specify MP_PROCS = %d. Terminating.\n",
            NPROCS);
    MPI_Finalize();
    exit(0);
  }

/* store the global rank in sendbuf */
sendbuf = rank;

/* Extract the original group handle */
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);

/* Divide tasks into two distinct groups based upon rank */
if (rank < numtasks/2)
  /* if rank = 0,1,2,3, put original processes 0,1,2,3
     into new_group */
  MPI_Group_incl(orig_group, 4, ranks1, &new_group);
else
  /* if rank = 4,5,6,7, put original processes 4,5,6,7
     into new_group */
  MPI_Group_incl(orig_group, 4, ranks2, &new_group);
```

```
    /* Create new new communicator and then perform collective
       communications */
    MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);

    /* new_comm contains a group with processes 0,1,2,3
       on processes 0,1,2,3 */
    /* new_comm contains a group with processes 4,5,6,7
       on processes 4,5,6,7 */
    MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT,
                  MPI_SUM, new_comm);

    /* new_rank is the rank of my processs in the new group */
    MPI_Group_rank (new_group, &new_rank);

    printf("rank= %d newrank= %d recvbuf= %d\n",
           rank,new_rank,recvbuf);

    MPI_Finalize();

    return 0;
}
```

Example

Code adapted from P. Pacheco, PP with MPI

```
/* comm_split.c -- build a collection of q
   communicators using MPI_Comm_split
 * Input: none
 * Output:  Results of doing a broadcast across each of
            the q communicators.
 * Note:  Assumes the number of processes, p = q^2
 */
#include <stdio.h>
#include "mpi.h"
#include <math.h>

int main(int argc, char* argv[])
{
   int        p, my_rank;
   MPI_Comm   my_row_comm;
   int        my_row, my_rank_in_row;
   int        q, test;

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

```
    MPI_Comm_size(MPI_COMM_WORLD, &p);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    q = (int) sqrt((double) p);
    /* my_rank is rank in MPI_COMM_WORLD.
       q*q = p */
    my_row = my_rank/q;
    MPI_Comm_split(MPI_COMM_WORLD, my_row, my_rank,
                   &my_row_comm);

    /* Test the new communicators */
    MPI_Comm_rank(my_row_comm, &my_rank_in_row);
    if (my_rank_in_row == 0) test = my_row;
    else test = 0;

    MPI_Bcast(&test, 1, MPI_INT, 0, my_row_comm);

    printf("Process %d > my_row = %d,"
           "my_rank_in_row = %d, test = %d\n",
           my_rank, my_row, my_rank_in_row, test);

    MPI_Finalize();
    return 0;
}
```

How things may work

- A group can be represented by an array `group` such that `group[i]` is the address of process with rank `i` in `group`
- An intra-communicator can be represented by a structure with components `group`, `myrank`, `context`
- When a process posts a send with (`dest`, `tag`, `comm`), the address of the destination is computed as `comm.group[dest]`
- The message sent carries a header of the form (`comm.myrank`, `tag`, `comm.context`)
- When a process posts a receive with (`source`, `tag`, `comm`), then headers of incoming messages are matched by (`source`, `tag`, `comm.context`)

  (first two may be "don't care")