COMS3008A: Parallel Computing
Introduction to MPI IV

Hairong Wang

School of Computer Science & Applied Mathematics
University of the Witwatersrand, Johannesburg

2021-10-26

WITS
UNIVERSITY

Contents

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

WITS UNIVERSITY

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

WITS UNIVERSITY

Some Important MPI Features

MPI provides the following features

- Groups
- Contexts
- Communicators

to make the message passing libraries effective so that

- A safe point-to-point communication without interference from other point-to-point communication;
- Collective operations within a group of processes while non-participants can still continue their work;
- Process ranks within a group or abstract names for processes on virtual topologies.

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

WITS UNIVERSITY

Groups

- A *group* is an ordered set of processes.
- A group is used within a communicator to describe the participants in a communication “universe” and to rank such participants.
- Special predefined group: `MPI_GROUP_EMPTY` — a group with no members.
- Predefined constant: `MPI_GROUP_NULL` — a value used for invalid group handles. For example, `MPI_GROUP_NULL` is returned when a group is freed.
- `MPI_GROUP_EMPTY` is a valid group handles. `MPI_GROUP_NULL` is invalid group handles.

WITS UNIVERSITY

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

Contexts

- A *context* is the communication environment.
- A *context* is a property of communicators that allows partitioning of the communication space.
- A message sent in one context cannot be received in another context. Separate contexts are entirely independent, or two distinct communicators have different contexts.
- Contexts are not explicit MPI objects; they appear only as part of the realization of communicators.
- A context is essentially a *system-managed* tag (distinct communicators use distinct contexts) that is associated with a group in a communicator; it makes a communicator safe for point-to-point and MPI-defined collective communication.

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

Communicators

- Communicators bring together the concepts of group and context.
- MPI communication operations reference communicators to determine the scope and the “communication universe” in which a point-to-point or collective operation is to operate.
- Each communicator contains a group of valid participants.
- For collective communication, the intra communicator specifies the set of processes that participate in the collective operation.
  - Intracommunicator: Refers to the regular communicators of communication within a group.
  - Intercommunicator: Communicators target group-to-group communication

WITS UNIVERSITY

Communicators cont.

- Predefined intracommunicator `MPI_COMM_WORLD` of all processes the local process can communicate with after initialization is defined once `MPI_Init` has been called.
- Predefined `MPI_COMM_NULL` is the value for invalid communicator handle. Used as an error result from some functions.
- Predefined `MPI_COMM_SELF` includes only the process itself.
- Avoid using two communicators that overlap.
- You always start with an existing communicator and subdivide it to make one or more new ones.

Why go beyond MPI_COMM_WORLD

- To use collective communication on only some processes
- Need to do a task on only some processes
- Want to do several tasks in parallel

WITS UNIVERSITY

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

Example 1

- If the radius of a circle is 1, then the area is π, and the area of the square around the circle, with the same center point as the circle, is 4.
- The ratio r of the area of the circle to that of the square is $\frac{\pi}{4}$.
- Compute r by generating random points (x, y) in the square and counting how many of them turn out to be in the circle $(x^2 + y^2 < 1)$.

Example 1 cont.

- Use only one process (called `server`) to generate the random numbers, and distribute these over the other processes.
- We want the processes other then the `server` to compute the ratio — need to use collective communication.
- We need to have two communicators.

Example 1 cont.

```
/* declare two communicators */
 MPI_Comm world=MPI_COM_WORLD, workers;
 /* declare two groups */
 MPI_Group world_group, worker_group;
 /* an array with 1 element (a scalar can be used)*/
 int ranks[1];
 MPI_Init(&argc, &argv);
 MPI_Comm_size(world, &numprocs);
 MPI_Comm_rank(world, &myid);
 server = numprocs - 1; /* the last process */
 MPI_Comm_group(world, $world_group); /* exatract the group */
 ranks[0] = server;
 /* created a new group without the 'server' process */
 MPI_Group_excl(world_group, 1, ranks, &worker_group);
 /* create a communicator for the 'worker_group' */
 MPI_Comm_create(world, worker_group, &workers);
 ......
 MPI_Group_free(&worker_group); /* free-up group */
 MPI_Group_free(&world_group);
 MPI_Comm_free(&workers); /* free-up communicator */
```

(See the following slides for the new functions used.)

Example 1 cont.

The program may continue in the following way:

- The `server` process (possibly with rank 0 or any other process) receives requests from workers for chunks of random numbers, generate these numbers, and then sends a unique chunk of random numbers to each worker who sent their requests.
- A worker process sends a request for random numbers to the server, receives the numbers, proceeds to test whether a pair of points fall in the circle or not, and accumulate the number of points fall in the circle, and those fall outside the circle, respectively.
- After computing a chunk of random numbers, the workers do a collective communication - `MPI_Allreduce` in this case, to compute an estimation of number π. If the estimation is not good enough, a worker needs to send another round of request to the server for a new chunk of random numbers.
- The stopping criteria is the error of estimated number π is less than a threshold, or a total number of random points to be inspected.

WITS UNIVERSITY

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

Group Management

- Group Accessors
  - `int MPI_Group_size(MPI_Group group, int *size)`
    Returns the number of processes in the group.
  - `int MPI_Group_rank(MPI_Group group, int *rank)`
    Returns the rank of calling process in the group

Note that the difference here with `MPI_Comm_size` and `MPI_Comm_rank` is that a group can be manipulated outside of communicators, but a group can be only used for message passing inside of a communicator.

Group Management Cont.

- Group Constructors
  - ```
    int MPI_Group_incl(
        MPI_Group group,
        int n,
        const int ranks[],
        MPI_Group *newgroup)
    ```
    Creates a group `newgroup` that consists of the `n` processes in `group` with ranks `ranks[0], ..., ranks[n-1]`. If `n = 0`, then `newgroup = MPI_GROUP_EMPTY`.
  - `int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)`
    Returns in `group` a handle to the group of `comm`.

WITS UNIVERSITY
    ```

Group Management cont.

- Group Constructors
  - ```
    int MPI_Group_excl(
        MPI_Group group,
        int n,
        const int ranks[],
        MPI_Group *newgroup)
    ```

    This function creates a group of processes `newgroup` that is obtained by deleting from `group` those `n` processes with ranks `ranks[0], . . ., ranks[n-1]`.
    ```

Group Management cont.

More group constructing functions: construct new groups from existing groups using various set operations.

- ```
  int MPI_Group_union(
      MPI_Group group1,
      MPI_Group group2,
      MPI_Group* newgroup)
  ```
- ```
  int MPI_Group_intersection(
      MPI_Group group1,
      MPI_Group group2,
      MPI_Group* newgroup)
  ```
- ```
  int MPI_Group_difference(
      MPI_Group group1,
      MPI_Group group2,
      MPI_Group* newgroup)
  ```
  ```

Group Management cont.

Set operations in group construction

- Union: Returns in `newgroup` a group consisting of all processes in `group1` followed by all processes in `group2`, with no duplication
- Intersection: Returns in `newgroup` all processes that are in both groups, ordered (rank) as in `group1`
- Difference: Returns in `newgroup` all processes in `group1` that are not in `group2`, ordered as in `group1`

Group Management cont.

- Group Destructors

```
int MPI_Group_free(MPI_Group *group)
```

WITS UNIVERSITY

Communicator constructors

- Communicator constructors
    - ```
      int MPI_Comm_create(
          MPI_Comm comm,
          MPI_Group group,
          MPI_Comm *newcomm)
      ```
    - Collective routine within the communicator `comm`
    - Creates a new communicator which is associated with `group`
    - `MPI_COMM_NULL` is returned to processes not in `group`
    - All `group` arguments must be the same on all calling processes
    - `group` must be a subset of the group associated with `comm`.
      ```

MPI_Comm_create() Example

Example 2

Consider dividing the processes in the `MPI_COMM_WORLD` into two groups and create a new communicator for each group.

Example 2

```c
#define NPROCS 8

int rank, new_rank, sendbuf, recvbuf, comm_sz;
/* Divide the processes by their ranks into two groups
   */
int ranks1[4]={0,1,2,3}, ranks2[4]={4,5,6,7};
MPI_Group orig_group, new_group;
MPI_Comm new_comm;
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
sendbuf = rank;
/* Extract the original group handle */
MPI_Comm_group(MPI_COMM_WORLD, &orig_group);
```

Example 2 cont.

```
/* Divide processes into two distinct groups based on rank */
if (rank < NPROCS/2)
  MPI_Group_incl(orig_group, NPROCS/2, ranks1, &new_group);
else
  MPI_Group_incl(orig_group, NPROCS/2, ranks2, &new_group);

/* Create new communicator and then perform collective communications */
MPI_Comm_create(MPI_COMM_WORLD, new_group, &new_comm);
MPI_Allreduce(&sendbuf, &recvbuf, 1, MPI_INT, MPI_SUM, new_comm);

MPI_Group_rank(new_group, &new_rank);
printf("rank = %d new_rank = %d recvbuf = %d\n", rank, new_rank, recvbuf);
/* clean-up follows */
```

Note that afetr calling `MPI_Comm_create` in the above code, there will be 2 `new_comm` communicators created, each for a distinct group of processes from `MPI_COMM_WORLD`.

WITS UNIVERSITY

MPI_Comm_free()

- Communicator Destructor

```
int MPI_Comm_free(MPI_Comm *comm)
```

When you have finished using a communicator, free (delete/destroy) it.

Outline

1. Groups and Communicators
   - Groups
   - Contexts
   - Communicators
   - Example: Monte Carlo Computation of Π
   - Group Management
   - Communicator Management

Communicator Management

- Communicator accessors
  - `int MPI_Comm_size(MPI_Comm comm, int *size)`
  - `int MPI_Comm_rank(MPI_Comm comm, int *rank)`

Communicator Constructors

```
MPI_Comm_dup(
    MPI_Comm oldcomm,
    MPI_Comm *newcomm)
```

Creates a new communicator that is an exact replica of an existing communicator.

MPI_Comm_split()

- Communicator constructors
  - ```
    int MPI_Comm_split(
        MPI_Comm comm,
        int color,
        int key,
        MPI_Comm *newcomm)
    ```
    - Partitions the group associated with the given communicator into disjoint subgroups.
    - `color` controls the subset assignment,
    - `key` controls the rank assignment.
    ```

MPI_Comm_split() cont.

- A collective operation.
- All the processes that pass in the same value of `color` will be placed in the same communicator, and that communicator will be the one returned to them.
- The `key` argument is used to assign ranks to the processes in the new communicator.
  - If all processes passing the same `color` value also pass the same `key` value, the order of the ranks in the new communicator will be the same as in the old one. Note that `color` ≥ 0.
  - If they pass in different values for `key`, then these values are used to determine their order in the new communicator.
  - For simplicity, `key = 0` — you don't care about the order.
  - `MPI_UNDEFINED` is used as the `color` for processes not to be included in any of the new groups.

MPI_Comm_split() cont.

- `MPI_Comm_split` creates several new communicators but each process is given access only to one of the new communicators.
- Assume that a collective call to `MPI_Comm_split` is executed in a group of 12 processes, with the arguments `color` and `key` given in the table below.

Rank	0	1	2	3	4	5	6	7	8	9	10	11
Process	a	b	c	d	e	f	g	h	i	j	k	l
Color	U	3	1	1	3	2	3	3	1	2	U	2
Key	0	1	2	3	1	9	3	8	1	0	0	0

The call generates 3 new communication domains (communicators) with following groups: {`b`, `e`, `g`, `h`}, {`c`, `d`, `i`}, {`f`, `j`, `l`}. Both process `a` and `k` are returned with `MPI_COMM_NULL` in `newcomm`, as their corresponding `color` values are both `MPI_UNDEFINED`.
What will be the order of ranks like in each group? (See previous slide on the use of `key` argument.)

MPI_Comm_split() Example

Example 3

Suppose we create NROW $\times$ NCOL number of processes, and the processes are arranged in a virtual 2D array topology. We want to form a communicator for the processes in each row, and do some computation using each communicator. How? Similarly, we want also formulate a communicator for the processes in each column.

Example 3 cont.

```c
#define NROW 3
#define NCOL 4

int irow, icol, color, key, rank_in_world;
MPI_Comm row_comm, col_comm;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank_in_world);

irow = rank_in_world%NROW;
icol = rank_in_world/NROW;
// Build row communicators
color = irow;
key = rank_in_world;
MPI_Comm_split(MPI_COMM_WORLD, color, key, &row_comm);
// Build column communicators
color = icol;
MPI_Comm_split(MPI_COMM_WORLD, color, key, &col_comm);
```

Example 3 cont.

```
int row_procs[NCOL], col_procs[NROW];
int max_rank_row, max_rank_col, my_max;
my_max = rank_in_world;

MPI_Allgather(&my_max,1,MPI_INT,row_procs,1,MPI_INT,
    row_comm);
MPI_Allgather(&my_max,1,MPI_INT,col_procs,1,MPI_INT,
    col_comm);

max_rank_row = row_procs[0];
for(int i = 1; i < NCOL; i++)
   if(row_procs[i] > max_rank_row) max_rank_row =
      row_procs[i];
max_rank_col = col_procs[0];
for(int i = 1; i < NROW; i++)
   if(col_procs[i] > max_rank_col) max_rank_col =
      col_procs[i];
```

WITS UNIVERSITY

Examples

Complete all the examples in the slides. Some example codes for `MPI_Comm_split` and MPI group management are given in `mpi_split_example1.c` and `mpi_group_example1.c`.

References

- Using MPI-1: Portable Parallel Programming with the Message Passing Interface, William Gropp, Ewing Lusk, and Anthony Skjellum. MIT Press Cambridge, London, England.
- Parallel Programming in C with MPI and OpenMP, Michael J. Quinn, Chapter 4–5, McGraw-Hill Education Group, 2003. `http://epcc.sjtu.edu.cn/wordpress/wp-content/uploads/2013/05/parallel-programming-in-c-with-mpi-and-openmp.pdf`
- A. Grama, A. Gupta, G. Karypis and V. Kumar, Introduction to Parallel Computing, 2nd Edition, Chapter 6.
- `https://computing.llnl.gov/tutorials/mpi/`
- MPI: The Complete Reference (`http://www.mpi-forum.org/docs/docs.html`)

WITS UNIVERSITY