MPI Groups and Communicator Management

Table of Contents

1. Overview
2. MPI Routines
3. Forest Dynamics Model
   3.1 Problem
   3.2 Steps
   3.3 Code
   3.4 Alternate Approach

References  Lab Exercises  Quiz  Evaluation  Navigation Guide

1. Overview
Thus far in the workshop, we have assumed that all processors are available for all communication events. This is the effect of using the MPI_COMM_WORLD communicator, which includes all processors, for message passing activity.

To limit communication to a subset of the processors, the programmer can create a group, and associate a communicator with that group. The new communicator can be used in point-to-point or collective communication routines. Both groups and communicators are MPI objects (stored in system space) accessed by handles (returned from or passed to MPI routines).

A group is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group.

A communicator is a mechanism for creating a self-contained communication "universe". A message sent with a given communicator is only eligible to be received by a process specifying the same communicator.

A communicator incorporates an instance of a group, and also includes contexts. A context can be thought of as a unique tag that is allocated by the system. For a message to be received, it must match on context and on message tag. A communicator includes two separate contexts: one for point-to-point communication and one for collective communication. These operations can therefore occur simultaneously within one communicator, without the point-to-point receiving the collective message, or vice versa.

Communicators and contexts are particularly important for libraries. There is no way to ensure that a library developer will choose a message tag that is not used by the application programmer. If the library creates its own communicator, it will be guaranteed a unique context and no interference from other communication events.

There are two additional issues related to communicators which are beyond the scope of this talk. First, a communicator can cache attribute information, to be carried over between subsequent subroutine calls. MPI virtual topologies are implemented using cached attributes. Second, in addition to intra-communicators (discussed in this talk), inter-communicators can be created to allow point-to-point communication between groups.

2. MPI Routines

MPI includes routines for accessing information on groups or communicators, for creating new groups or communicators from existing ones, and for deleting groups or communicators. A list follows. Please see the man pages or the MPI standard for information on syntax. The commands that will be illustrated later in this module and lab are italicized.

Communicator creation routines are collective. They require all processes in the input communicator to participate, and may require communication amongst processes. All other group and communicator routines are local. As will be discussed later, it often makes sense to
have all members of an input group call a group creation routine, if a communicator will later be created for that group.

**Group Accessors**

- **MPI_Group_size**: returns number of processes in group
- **MPI_Group_rank**: returns rank of calling process in group
- **MPI_Group_translate_ranks**: translates ranks of processes in one group to those in another group
- **MPI_Group_compare**: compares group members and group order

**Group Constructors**

- **MPI_Comm_group**: returns the group associated with a communicator
- **MPI_Group_union**: creates a group by combining two groups
- **MPI_Group_intersection**: creates a group from the intersection of two groups
- **MPI_Group_difference**: creates a group from the difference between two groups
- **MPI_Group_incl**: creates a group from listed members of an existing group
- **MPI_Group_excl**: creates a group excluding listed members of an existing group
- **MPI_Group_range_incl**: creates a group according to first rank, stride, last rank
- **MPI_Group_range_excl**: creates a group by deleting according to first rank, stride, last rank

**Group Destructors**

- **MPI_Group_free**: marks a group for deallocation

**Communicator Accessors**

- **MPI_Comm_size**: returns number of processes in communicator’s group
- **MPI_Comm_rank**: returns rank of calling process in communicator’s group
- **MPI_Comm_compare**: compares two communicators

**Communicator Constructors**

- **MPI_Comm_dup**: duplicates a communicator
- **MPI_Comm_create**: creates a new communicator for a group
- **MPI_Comm_split**: splits a communicator into multiple, non-overlapping communicators

**Communicator Destructors**

- **MPI_Comm_free**: marks a communicator for deallocation

---

### 3. Forest Dynamics Model

#### 3.1 Problem
In forest dynamics models, leaf-area profile growth is dictated by the amount of available light. In some models, diffuse light at a given grid point depends on data at grid points in cardinal directions (N,E,S,W). If one views the processors laid out in a 2-dimensional grid, one would like to exchange data among processors sharing the same row and processors sharing the same column. This can be accomplished by using groups of processors.

Suppose we have 12 processors laid out in a 3x4 grid, where the number in the box is the processor number.

**Forest Simulation**

**3x4 grid**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>8</td>
<td>11</td>
</tr>
</tbody>
</table>

Since data must be exchanged among rows and columns of processors, we must create a group for each row and a group for each column. In our example, 7 different groups must thus be generated. However, a particular processor will belong to only 2 of these groups -- one row group and one column group.

### 3.2 Steps

This module will demonstrate two methods for creating the groups for the forest dynamics model. The first method is the most generally applicable, but not the most compact.

It should first be noted, however, that MPI does not provide a mechanism to build groups from scratch; they can only be developed from previously defined ones. This leads to an obvious question: How can one build the first group? You must start with the base group of all processes, which is associated with the MPI_COMM_WORLD communicator.

Here are the steps for the first method:

http://www.tc.cornell.edu/Services/Edu/Topics/MPI/Groups/more.asp
1. Access the base group of all processes via a call to MPI_COMM_GROUP
2. Create the new group via a call to MPI_GROUP_INCL
3. Create the communicator for the new group via a call to MPI_COMM_CREATE

This last step is essential because all communication events within the group are activated by a communicator for the group, not by the group object itself.

### 3.3 Code

The following snippet of Fortran code shows how to set up groups for rows and columns.

Click here for C version.

```fortran
subroutine set_group(row_comm, col_comm)
  include 'mpif.h'
  parameter (NROW=3, NCOL=4)
  integer row_list(NCOL), col_list(NROW), base_grp, grp
  integer temp_comm, row_comm, col_comm

  ! Get base group from MPI_COMM_WORLD communicator
  call MPI_COMM_GROUP(MPI_COMM_WORLD, base_grp, ierr)

  ! Establish the row and column to which this processor belongs
  call MPI_COMM_RANK(MPI_COMM_WORLD, irank, ierr)
  irow = MOD(irank, NROW) + 1
  icol = INT(irank/NROW) + 1

  ! Build row groups
  row_list(1) = 0
  do i=2, NCOL
    row_list(i) = row_list(i-1) + NROW
  end do
  do i=1, NROW
    call MPI_GROUP_INCL(base_grp, NCOL, row_list, grp, ierr)
    call MPI_COMM_CREATE(MPI_COMM_WORLD, grp, temp_comm, ierr)
    if (irow .eq. i) row_comm = temp_comm
    do j=1, NCOL
      row_list(j) = row_list(j)+1
    end do
  end do

  ! Build column groups
```

MPI Groups and Communicator Management

10/8/2002

http://www.tc.cornell.edu/Services/Edu/Topics/MPI/Groups/more.asp
You might wonder why every processor is involved in the creation of all groups, even though it belongs to only 2 of them. This is because the MPI standard states that the MPI_COMM_CREATE call must be executed by all processes in the input communicator (in our case, MPI_COMM_WORLD), and that all processes must pass the same value for the group argument (grp), even if they do not belong to the new group.

The group communicators row_comm and col_comm can be used in a number of ways. Returning to the forest dynamics model again, each processor calculates the maximum tree height for trees in its domain. This information must be exchanged with all other processors sharing the same row and column, with the result that each processor can then calculate the maximum tree height in the entire row and column. If maxht is the local maximum tree height, then the following snippet of code illustrates how one can determine the row and column maximum:

```fortran
include 'mpif.h'
parameter (NROW=3, NCOL=4)
integer row_hgt(NCOL), col_hgt(NROW), max_row, max_col
call MPI_ALLGATHER(maxht,1,MPI_INTEGER,row_hgt,1,MPI_INTEGER,
1    row_comm,ierr)
call MPI_ALLGATHER(maxht,1,MPI_INTEGER,col_hgt,1,MPI_INTEGER,
1    col_comm,ierr)
max_row = row_hgt(1)
do i=2,NCOL
   if (row_hgt(i) .gt. max_row) max_row=row_hgt(i)
end do
max_col = col_hgt(1)
do i=2,NROW
   if (col_hgt(i) .gt. max_col) max_col=col_hgt(i)
end do
```

For collective communication, all processes in the specified communicator must participate. Processes not in the communicator must not call the routine. In this example, all processes belong to a valid row_comm and col_comm, so all call MPI_ALLGATHER. Some codes will require making the collective communication call conditional upon membership in the communicator's group. In this situation, all processes can call MPI_GROUP_RANK to learn their rank in the group. Those whose rank equals MPI_UNDEFINED should not call the collective communication routine.
3.4 Alternate Approach

An alternate approach is to use MPI_COMM_SPLIT, which partitions one communicator into multiple, non-overlapping communicators. All manipulation of groups is done behind the scenes. This approach is less generally applicable than using MPI_COMM_CREATE, but produces more compact code for regular decomposition problems, such as the forest dynamics problem.

In the subroutine below, the first call to MPI_COMM_SPLIT divides MPI_COMM_WORLD into three communicators corresponding to the three rows in the 3x4 grid. One communicator includes all processes for which color = 1 (row 1), another includes all processes for which color = 2 (row 2), and a third includes all processes for which color = 3 (row 3). Within the new communicators, processes are ranked in order of increasing key value (in this case, according to increasing rank in the original communicator MPI_COMM_WORLD).

Click here for C version.

```fortran
subroutine set_group(row_comm, col_comm)
   include 'mpif.h'
   parameter (NROW=3, NCOL=4)
   integer row_comm, col_comm, color, key
   ! Establish the row and column to which this processor belongs
   call MPI_COMM_RANK(MPI_COMM_WORLD,irank,ierr)
   irow = MOD(irank,NROW) + 1
   icol = INT(irank/NROW) + 1
   ! -------------------------
   ! Build row communicators
   ! -------------------------
   color = irow
   key = irank
   call MPI_COMM_SPLIT (MPI_COMM_WORLD, color, key, row_comm, ierr)
   ! ---------------------------
   ! Build column communicators
   ! ---------------------------
   color = icol
   call MPI_COMM_SPLIT (MPI_COMM_WORLD, color, key, col_comm, ierr)
   return
end
```

References

Take a multiple-choice quiz on this material, and submit it for grading.

Lab exercise for MPI Groups and Communicator Management

Tutorial for MPI Groups and Communicator Management

Please complete this short evaluation form. Thank you!