(HPSC **5576** ELIZABETH JESSUP)

HIGH PERFORMANCE SCIENTIFIC COMPUTING

:: Homework / 6

:: Student / Florian Rappl

2 problems / **15** points

Problem 1

Task:

Programming assignment 7.12.2 from Pacheco's PPMPI textbook: Creating an intercommunicator from intracommunicators (p. 133), *10 pts*

Write a short program that splits the processes in MPI_COMM_WORLD into two communicators: the processes with even ranks and the processes with odd ranks. (Do this using standard MPI groups and intracommunicators.)

Then, create an intercommunicator from these two communicators. Have each process in the odd-ranked communicator send a message to a process in the even-ranked communicator. Be sure to handle the case where there's an odd number of processes in MPI_COMM_WORLD.

Solution:

My solution is pretty straight forward programmed. First of all I make a group that will lead as the parent group for the intercommunication, i.e. the communication between the subgroups. Since *MPI_COMM_WORLD* is the root of the communication group tree I picked this one.

Then I tell MPI to recognize a new group and after that I fill this new group with the members that the one process that is instanced belongs to (either ODD or EVEN group). The odd group will always have floor(p/2) members, while the even group will have between floor(p/2) and floor(p/2)+1 members. To last case I handle with an *?*-operator and the modulo operation % 2.

In the next step I create the intra-communication and then the inter-communication. The group leaders are 0 and 1 (viewed from the parent communicator). Here I abuse the % 2 operator again.

To show the success I send a message between the communicators. I sent from every odd process (viewed from the parent communicator) to its counter-part in the other group (i.e. from process 1 in its own group to process 1 in the other group – those would be process 2 and 3 seen from the *MPI_COMM_WORLD* communicator). To do that I just use floor(my_rank / 2) as rank in the group.

The receiving part has to be implemented carefully since an odd number of processes could lead to a receive call that has no proper send call on the other side. Since the problem arises only if $p - 1$ is equal to an even processor I just had to exclude that case.

Program output:

```
D:\Documents\Visual Studio 2005\Projects\Whatever\Debug>mpiexec -n 9 intcom
Greetings from processor 1 (0,o) to 0 (0,e)
Greetings from processor 5 (2,o) to 4 (2,e)
Greetings from processor 3 (1,o) to 2 (1,e)
Greetings from processor 7 (3,o) to 6 (3,e)
```

Additionally to that output I included the possibility to print an intra-communication-broadcast – just to see if the group has been created the way it was intended. This feature is included in the *.tar.zip.

Code printout:

```
#include <stdio.h>
#include <string.h>
#include "mpi.h"
int main(int argc, char* argv[])
{
      int         my_rank;          /* rank of process      */
      int         p;                /* number of processes  */
      int         tag = 0;          /* tag for messages     */
      char        message[100];     /* storage for message  */
      int*        group_members;    /* local group members  */
      int         group_size;       /* local group size     */
      int         i, j;             /* loop counters        */
      MPI_Status  status;           /* return status for rec*/
      MPI_Group   comm_group;       /* COMM_WORLD group     */
      MPI_Comm    parent_comm;      /* WORLD_COMMUNICATION  */
      MPI_Group   intra_group;      /* COMM_INTRA group     */
      MPI_Comm    intra_comm;       /* INTRA_COMMUNICATION  */
      MPI_Comm    inter_comm;       /* INTER_COMMUNICATION  */
      /* Start up MPI */
      MPI_Init(&argc, &argv);
      /* Define a communicator for COMM_WORLD as parent */
      parent_comm = MPI_COMM_WORLD;
      /* Find out process rank  */
      MPI_Comm_rank(parent_comm, &my_rank);
      /* Find out number of processes */
      MPI_Comm_size(parent_comm, &p); //Cancel if p < 2
      /* Get the whole group of the COMM_WORLD */
      MPI_Comm_group(parent_comm, &comm_group);
      /* Build a new group (sub-group) of COMM_WORLD */
      group_size = p / 2 + (my_rank  % 2 == 0 ? p % 2 : 0);
      group_members = (int*)malloc(sizeof(int) * group_size);
      j = my_rank % 2;
      for(i = 0; i < group_size; i++)
            group_members[i] = j + i * 2;
      /* Set the group members of this subgroup */
      MPI_Group_incl(comm_group, group_size, group_members,
                        &intra_group);
      /* Create the new communicator with the def. above */
      MPI_Comm_create(MPI_COMM_WORLD, intra_group, &intra_comm);
      /* Create a proper inter-communication */
      MPI_Intercomm_create(intra_comm, 0, parent_comm,
                        (my_rank + 1) % 2,      tag, &inter_comm);
      /* Sending a message if odd process or receiving if even */
      if (my_rank % 2 == 0 && my_rank < p - 1)
      { /* Receiving - unless last && odd number of processes */
            MPI_Recv(&message, 100, MPI_CHAR, my_rank / 2, tag,
                        inter_comm, &status);
            printf("%s\n", message);
      }
      else if(my_rank % 2 == 1)
      { /* always sending */
            sprintf(message, "Greetings from proc. %d (%d,o) to %d (%d,e)",
                        my_rank, my_rank / 2, my_rank - 1, my_rank / 2);
            MPI_Send(&message, 100, MPI_CHAR, my_rank / 2, tag,
                        inter_comm);
      } //Printout Broadcast message to see group structure(s)
      /* Shut down MPI */
      MPI_Finalize();
      return 0;
```

```
60  } /* main */
```

Problem 2

Task:

Spawning multiple programs at the same time using mpirun/mpiexec , *5 pts*

With MPI-2, there are three ways to start or link multiple MPI executable into a single functional program:

- Multiple programs can be started at the same time using mpirun/mpiexec
- A MPI program can start child parallel programs using MPI_Comm_spawn
- Separately running programs can establish communication using "ports"

The MPI installation on Blacklight supports starting multiple programs at the same time from a single *mpiexec* command line. Make a PBS script with an *mpirun* line that runs both of the programs at the same time.

Solution:

My program just uses the code snipped for printing the process name and the process rank as stated in the assignment. Additionally I compiled two programs called *spawnONE* and *spawnTWO* with that code. Then I modified the script I used for Blacklight in the single-blade mode and executed the script. The modifications in the script lied in the *mpirun* command. The altered line is shown in the output.

Question:

If you start two programs on the same mpirun command line, how are the communicators configured? Do you get one MPI_COMM_WORLD communicator shared among both programs? Do you get two separate MPI_COMM_WORLDs linked with an intercommunicator?

Answer:

I got one shared *MPI_COMM_WORLD* communicator since the number of total processes has been $p_1 + p_2$ and the rank of processes in program number 2 has been $p_1 + r$, where r is the rank the process would have had normally.

Program output:

```
mpirun 5 ./spawnONE : 3 ./spawnTWO
Hello, I am 7 of 8 running ./spawnTWO on bl0.psc.teragrid.org
Hello, I am 0 of 8 running ./spawnONE on bl0.psc.teragrid.org
Hello, I am 2 of 8 running ./spawnONE on bl0.psc.teragrid.org
Hello, I am 1 of 8 running ./spawnONE on bl0.psc.teragrid.org
Hello, I am 3 of 8 running ./spawnONE on bl0.psc.teragrid.org
Hello, I am 4 of 8 running ./spawnONE on bl0.psc.teragrid.org
Hello, I am 6 of 8 running ./spawnTWO on bl0.psc.teragrid.org
Hello, I am 5 of 8 running ./spawnTWO on bl0.psc.teragrid.org
```

Code printout:

```
1  #include <stdio.h>
2  #include <string.h>
3  #include "mpi.h"
4
5  int main(int argc, char* argv[])
6  {
7      int         my_rank;       /* rank of process      */
8      int         p;             /* number of processes  */
9      int         source;        /* rank of sender       */
10     int         dest;          /* rank of receiver     */
11     int         tag = 0;       /* tag for messages     */
12     char        my_name[MPI_MAX_PROCESSOR_NAME];
13     int         my_name_len;   /* length of my_name    */
14     MPI_Status  status;        /* return status for rec*/
15     /* Start up MPI */
16     MPI_Init(&argc, &argv);
17     /* Find out process rank  */
18     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
19     /* Find out number of processes */
20     MPI_Comm_size(MPI_COMM_WORLD, &p);
21     /* Modified from Pacheco -- get machine name */
22     MPI_Get_processor_name( my_name, &my_name_len );
23       /* Print out info */
24     printf( "Hello, I am %i of %i running %s on %s\n", my_rank,
25                   p, argv[0], my_name);
26     /* Shut down MPI */
27     MPI_Finalize();
28     return 0;
29 } /* main */
```