Collective Communications I

Ned Nedialkov

McMaster University
Canada

CS/SE 4F03
January 2016

Outline

Introduction

A collective communication involves all the processes in a communicator

Here we consider

- Broadcast
- Reduce

In part II, we consider

- Gather
- Scatter
- Allgather
- Allscatter

Broadcast

Broadcast: a single process sends data to all processes in a communicator

```
int Bcast(void *buffer, int count, Datatype datatype, int root,
    MPI_Comm comm)
```

`buffer`	starting address of buffer (in/out)
`count`	number of entries in buffer
`datatype`	data type of buffer
`root`	rank of broadcast root
`comm`	communicator

- `MPI_Bcast` sends a copy of the message on process with rank `root` to each process in `comm`
- Must be called in each process
- Data is sent in root and received by all other processes
- `buffer` is 'in' parameter in root and 'out' parameter in the rest of processes
- Cannot receive broadcasted data with `MPI_Recv`

Example: reading and broadcasting data

Code adapted from P. Pacheco, PP with MPI

```
/* getdata2.c */
/* Function Get_data2
 * Reads in the user input a, b, and n.
 * Input parameters:
 *     1.  int my_rank:  rank of current process.
 *     2.  int p:  number of processes.
 * Output parameters:
 *     1.  float* a_ptr:  pointer to left endpoint a.
 *     2.  float* b_ptr:  pointer to right endpoint b.
 *     3.  int* n_ptr:  pointer to number of trapezoids.
 * Algorithm:
 *     1.  Process 0 prompts user for input and
 *         reads in the values.
 *     2.  Process 0 sends input values to other
 *         processes using three calls to MPI_Bcast.
 */
```

```
#include <stdio.h>
#include "mpi.h"
void Get_data2(float* a_ptr, float* b_ptr, int* n_ptr,
               int my_rank)
{
  if (my_rank == 0)
    {
      printf("Enter␣a,␣b,␣and␣n\n");
      scanf("%f␣%f␣%d", a_ptr, b_ptr, n_ptr);
    }
  MPI_Bcast(a_ptr, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
  MPI_Bcast(b_ptr, 1, MPI_FLOAT, 0, MPI_COMM_WORLD);
  MPI_Bcast(n_ptr, 1, MPI_INT,   0, MPI_COMM_WORLD);
}
```

- Note: this code is not efficient
- It is more efficient, e.g., to store the data at `a_ptr`, `b_ptr`, and `n_ptr` in one array and broadcast a single message
- Then each process must extract these data

A more efficient version could be (by NN)

```
/* getdata3.c */
#include <stdlib.h>
#include <stdio.h>
#include "mpi.h"
void Get_data2(float* a_ptr, float* b_ptr, int* n_ptr,
               int my_rank)
{
  // buffer to store a, b, and n
  char *buf = malloc(2*sizeof(float)+sizeof(int));
  float *a  = (float*)buf;
  float *b  = a+1;
  int   *n  = (int*)(b+1);
  if (my_rank == 0)
    {
      printf("Enter␣a,␣b,␣and␣n\n");
      scanf("%f␣%f␣%d", a, b, n);
    }
```

```
  MPI_Bcast(buf, 2*sizeof(float)+sizeof(int), MPI_CHAR,
            0, MPI_COMM_WORLD);
  *a_ptr = *a;
  *b_ptr = *b;
  *n_ptr = *n;
  free (buf);
}
```

Reduce

Data from all processes are combined using a binary operation

```
int MPI_Reduce(void *sendbuf, void *recvbuf, int count,
    MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)
```

`sendbuf`	address of send buffer
`recvbuf`	address of receive buffer
`count`	number of entries in send buffer
`datatype`	data type of elements in send buffer
`op`	reduce operation; predefined, `MPI_MIN`, `MPI_MAX`, `MPI_SUM`, ..., or user defined
`root`	rank of root process
`comm`	communicator

Must be called in all processes in a communicator

Example: trapezoid with reduce

Code adapted from P. Pacheco, PP with MPI

```
/* redtrap.c */
#include <stdio.h>
#include "mpi.h"
extern void Get_data2(float* a_ptr, float* b_ptr,
                      int* n_ptr, int my_rank);
extern float Trap(float local_a, float local_b,
                  int local_n, float h);
int main(int argc, char** argv)
{
   int          my_rank, p;
   float        a, b, h;
   int          n;
   float        local_a, local_b, local_n;
   float        integral;  /* Integral over my interval */
   float        total;     /* Total integral            */
```

```
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  Get_data2(&a, &b, &n, my_rank);
  h = (b-a)/n;
  local_n = n/p;
  local_a = a + my_rank*local_n*h;
  local_b = local_a + local_n*h;
  integral = Trap(local_a, local_b, local_n, h);
  /* Add up the integrals calculated by each process */
  MPI_Reduce(&integral, &total, 1, MPI_FLOAT,
             MPI_SUM, 0, MPI_COMM_WORLD);
  if (my_rank == 0)
    {
      printf("With n = %d trapezoids, our estimate\n", n);
      printf("of the integral from %f to %f = %f\n",
             a, b, total);
    }
  MPI_Finalize();
  return 0;
}
```