MPI Collective Communications

Department of Computer Science
University of San Francisco

February 19, 2014

MPI provides a number of so-called “collective communication” functions. These are functions that must be called by all the processes in a communicator (unlike point-to-point communications). The following list contains information on the functions that are most likely to be useful in Programming Assignment 2. Note that for each of these functions the return value is an error code. For further information consult the online man pages.

- MPI_Allgather. Gather the contents of each process’ sendbuf onto each process.

  Syntax:

  ```c
  int MPI_Allgather(void* sendbuf /* in */,
  int sendcount /* in */,
  MPI_Datatype sendtype /* in */,
  void* recvbuf /* out */,
  int recvcount /* in */,
  MPI_Datatype recvtype /* in */,
  MPI_Comm comm /* in */);
  ```

  On each process gather the contents of every process’ sendbuf into recvbuf. Ordinarily the contents of process 0’s sendbuf is put first into recvbuf, then the contents of process 1’s, etc. Note that recvcount is the amount of data received from each process, not the total amount received. At this point you should use the same values for sendcount and recvcount and the same types in sendtype and recvtype.
• **MPI_Allreduce.** Perform a reduction on the contents of the input buffers on all the processes. The result is stored in `recvbuf` in all the processes in `comm`.

Syntax:

```c
int MPI_Allreduce(void* sendbuf /* in */, void* recvbuf /* out */, int count /* in */, MPI_Datatype datatype /* in */, MPI_Op op /* in */, MPI_Comm comm /* in */);
```

In general, the values in `count`, `datatype`, `op`, and `comm` must be the same on all the processes.

Some possibilities for `op` are

- `MPI_MAX`, `MPI_MIN`, `MPI_SUM`, and `MPI_PROD`

• **MPI_Barrier.** Block all the processes in a communicator until all the processes in the communicator have started executing the call.

Syntax:

```c
int MPI_Barrier(MPI_Comm comm /* in */);
```

• **MPI_Bcast.** Broadcast a message from one process to all the processes in a communicator.

Syntax:

```c
int MPI_Bcast(void* buf /* in/out */, int count /* in */, MPI_Datatype datatype /* in */, int root /* in */, MPI_Comm comm /* in */);
```

Send the contents of `buf` on the process with rank `root` to all the processes in `comm`. In general the `count`, `datatype`, and `root` arguments...
must be the same on all the processes. The buf arg is in on root and out on the other processes.

Note that all the processes in comm must call MPI_Comm. In particular, none of them should call MPI_Send or MPI_Recv.

- Allgather. Gather the contents of each process’ sendbuf into the recvbuf on the process with rank root in comm.

Syntax:

```c
int MPI_Gather(void* sendbuf /* in */,
               int sendcount /* in */,
               MPI_Datatype sendtype /* in */,
               void* recvbuf /* out */,
               int recvcount /* in */,
               MPI_Datatype recvtype /* in */,
               int root /* in */,
               MPI_Comm comm /* in */);
```

On the process with rank root gather the contents of every process’ sendbuf into recvbuf. Ordinarily the contents of process 0’s sendbuf is put first into recvbuf, then the contents of process 1’s, etc. Note that recvcount is the amount of data received from each process, not the total amount received. At this point you should use the same values for sendcount and recvcount and the same types in sendtype and recvtype.

- MPI_Reduce. Perform a reduction on the contents of the input buffers on all the processes. The result is stored only on the process with rank root.

Syntax:

```c
int MPI_Reduce(void* sendbuf /* in */,
                void* recvbuf /* out */,
                int count /* in */,
                MPI_Datatype datatype /* in */,
                MPI_Op op /* in */,
                int root /* in */,
                MPI_Comm comm /* in */);
```
Carry out the operation specified by op on the data in the processes’ sendbuf’s. Store the result in recvbuf on the process with rank root. In general, the values in count, datatype, op, root, and comm must be the same on all the processes.

Some possibilities for op are

MPI_MAX, MPI_MIN, MPI_SUM, and MPI_PROD

- MPI_Scatter. Distribute the contents of sendbuf from the process with rank root among the processes in comm.

Syntax:

```c
int MPI_Scatter(void* sendbuf /* in */,
                int sendcount /* in */,
                MPI_Datatype sendtype /* in */,
                void* recvbuf /* out */,
                int recvcount /* in */,
                MPI_Datatype recvtype /* in */,
                int root /* in */,
                MPI_Comm comm /* in */);
```

From the process with rank root distribute the contents of sendbuf among all the processes’ recvbuf’s. Ordinarily the first block of data in sendbuf goes to process 0, the next block goes to process 1, etc. Note that sendcount is the amount of data sent to each process, not the total amount sent. At this point you should use the same values for sendcount and recvcount and the same types in sendtype and recvtype.