MPI Collective Communication

Collective Communication

- Communication involving a group of processes
- Called by all processes in the communicator
- Communication takes place within a communicator
- Built up from point-to-point communications

Characteristics of Collective Communication

- Collective communication will not interfere with point-to-point communication
- All processes must call the collective function
- Substitute for a sequence of point-to-point function calls
- Synchronization not guaranteed (except for barrier)
- No non-blocking collective function. Function return implies completion
- No tags are needed
- Receive buffer must be exactly the right size

Types of Collective Communication

- Synchronization
  - barrier
- Data exchange
  - broadcast
  - gather, scatter, all-gather, and all-to-all exchange
  - Variable-size-location versions of above
- Global reduction (collective operations)
  - sum, minimum, maximum, etc

Barrier Synchronization

- Red light for each processor: turns green when all processors have arrived
- A process calling it will be blocked until all processes in the group (communicator) have called it

```c
int MPI_Barrier(MPI_Comm comm);
```

- comm: communicator whose processes need to be synchronized

Broadcast

- One-to-all communication: same data sent from root process to all others in communicator
- All processes must call the function specifying the same root and communicator

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

- buf: starting address of buffer (sending and receiving)
- count: number of elements to be sent/received
- datatype: MPI datatype of elements
- root: rank of sending process
- comm: MPI communicator of processors involved
Scatter

- One-to-all communication: different data sent to each process in the communicator (in rank order)
- Example: partition an array equally among the processes

```c
int MPI_Scatter(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)
```

- sbuf and rbuf: starting address of send and receive buffers
- scount and rcount: number of elements sent and received to/from each process
- stype and rtype: MPI datatype of sent/received data
- root: rank of sending process
- comm: MPI communicator of processors involved

Gather

- All-to-one communication: different data collected by the root process from other processes in the communicator
- Collection done in rank order
- MPI_Gather has same arguments as MPI_Scatter
- Receive arguments only meaningful at root
- Example: collect an array from data held by different processes

```c
int MPI_Gather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)
```

Scatter and Gather

![Diagram]

Example

- Matrix-vector multiply with matrix A partitioned row-wise among 4 processors. Partial results are gathered from all processors at end of computation

```c
double A[25][100], x[100], ypart[25], ytotal[100];
int root = 0;
for (i = 0; i < 25; i++) {
    for (j = 0; j < 100; j++)
        ypart[i] = ypart[i] + A[i][j]*x[j];
}
MPI_Gather(ypart, 25, MPI_DOUBLE, ytotal, 25, MPI_DOUBLE, root, MPI_COMM_WORLD);
```

All-Gather and All-to-All (1)

- All-gather
  - All processes, rather than just the root, gather data from the group
- All-to-all
  - All processes, rather than just the root, scatter data to the group
  - All processes receive data from all processes in rank order
- No root process specified
- Send and receive arguments significant for all processes

```c
int MPI_Allgather(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm)
```

- scount: number of elements sent to each process: for all-to-all communication, size of sbuf should be scount*p (p = # of processes)
- rcount: number of elements received from any process; size of rbuf should be rcount*p (p = # of processes)

All-Gather and All-to-All (2)

```c
int MPI_Alltoall(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, MPI_Comm comm)
```
Global Reduction Operations (1)

- Used to compute a result involving data distributed over a group of processes
- Result placed in specified process or all processes
- Examples
  - Global sum or product
  - Global maximum or minimum
  - Global user-defined operation

\[
D_j = D(0, j)^* D(1, j)^* D(2, j)^*...^* D(n-1, j)
\]

- \(D(i, j)\) is the \(j\)th data held by the \(i\)th process
- \(n\) is the total number of processes in the group
- \(^*\) is a reduction operation
- \(D_j\) is the result of reduction operation performed on the \(j\)th elements held by all processes in the group

Global Reduction Operations (2)

```c
int MPI_Reduce(void *sbuf, void *rbuf, int count,
                MPI_Datatype stype, MPI_Op op, int root,
                MPI_Comm comm);
int MPI_Allreduce(void *sbuf, void *rbuf, int count,
                  MPI_Datatype stype, MPI_Op op, MPI_Comm comm);
int MPI_Reduce_scatter(void *sbuf, void *rbuf, int *
                        rcounts, MPI_Datatype stype, MPI_Op op,
                        MPI_Comm comm);
```

Predefined Reduction Operations

<table>
<thead>
<tr>
<th>MPI name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_MAX</td>
<td>Maximum</td>
</tr>
<tr>
<td>MPI_MIN</td>
<td>Minimum</td>
</tr>
<tr>
<td>MPI_SUM</td>
<td>Sum</td>
</tr>
<tr>
<td>MPI_PROD</td>
<td>Product</td>
</tr>
<tr>
<td>MPI_BAND</td>
<td>Bitwise AND</td>
</tr>
<tr>
<td>MPI_BOR</td>
<td>Bitwise OR</td>
</tr>
<tr>
<td>MPI_LXOR</td>
<td>Logical exclusive OR</td>
</tr>
<tr>
<td>MPI_BXOR</td>
<td>Bitwise exclusive OR</td>
</tr>
<tr>
<td>MPI_MAXLOC</td>
<td>Maximum and location</td>
</tr>
<tr>
<td>MPI_MINLOC</td>
<td>Minimum and location</td>
</tr>
</tbody>
</table>

Minloc and Maxloc

- Designed to compute a global minimum/maximum and an index associated with the extreme value
- Common application: index is processor rank that held the extreme value
- If more than one extreme exists, index returned is for the first
- Designed to work on operands that consist of a value and index pair. MPI defines such special data types:
  - MPI_FLOAT_INT, MPI_DOUBLE_INT, MPI_LONG_INT, MPI_2INT, MPI_SHORT_INT, MPI_LONG_DOUBLE_INT

Example

```c
#include <mpi.h>
main (int argc, char *argv[]) {
    int rank, root;
    struct { double value; int rank; } in, out;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    in.value = rank+1;
    in.rank = rank;
    root = 0;
    MPI_Reduce(&in, &out, 1, MPI_DOUBLE_INT, MPI_MAXLOC, root,
               MPI_COMM_WORLD);
    if (rank == root) printf("PE %i: max = %f at rank %i\n", rank, out.value, out.rank);
    MPI_Finalize();
}
```

Output: PE 0: max = 5.0 at rank 4
Variable-Size-Location Collective Functions

- Allows varying size and relative locations of messages in buffer
- Examples: MPI_Scatterv, MPI_Gatherv, MPI_Allgatherv, MPI_Alltoally
- Advantages
  - More flexibility in writing code
  - Less need to copy data into temporary buffers
  - More compact code
  - Vendor’s implementation may be optimal
- Disadvantage: may be less efficient than fixed size/location functions

Scatterv and Gatherv

int MPI_Scatterv(void *sbuf, int *scount, int *displs, MPI_Datatype stype, void *rbuf, int rcount, MPI_Datatype rtype, int root, MPI_Comm comm)

int MPI_Gatherv(void *sbuf, int scount, MPI_Datatype stype, void *rbuf, int *rcount, int *displs, MPI_Datatype rtype, int root, MPI_Comm comm)

*scount and *rcount: integer array containing number of elements sent/received to/from each process
*displs: integer array specifying the displacements relative to start of buffer at which to send/place data to corresponding process