MPI: Barrier

- A barrier can be used to synchronize all processes in a communicator. Each process wait till all processes reach this point before proceeding further.

```
MPI_Barrier(communicator)
```

MPI: Scatter

• Given an array, divide it into equal contiguous parts and send to nodes, one part each. This is equivalent to n sends. The 0th process gets the first part, 1st processor the second part, and so on. Number of data elements to given to each node is specified in `send count`.

```
MPI_Scatter(send buffer, send count, send type,
recv buffer, recv count, recv type, root,
communicator)
```

MPI: ScatterV

• This is a vector variant of `MPI_Scatter`. Here the user decides on how the send buffer is to be divided.

```
MPI_Scatterv(send buffer, send count (array),
array of displacements, send type, recv buffer,
recv count, recv type, root, communicator)
```

◦ `Displacements` and `send count` arrays specify which and how large a data chunk is being sent to a given process.

◦ Multiple reading of any location in `send buffer` is not allowed.

◦ Different sized chunks can be sent and different chunks need not be contiguous, e.g., we can start with a matrix and scatter only the upper (or lower) triangular part.

MPI: Gather

- Given a small array of same size with each of the processes, `MPI_Gather` collects these in order of increasing process rank and combines it on the root node. This is equivalent to n receives. Number of data elements to be collected from each node is specified in `send count`.

```
MPI_Gather(send buffer, send count, send type,
recv buffer, recv count, recv type, root,
communicator)
```

MPI: GatherV

- This is a vector variant of `MPI_Gather`. Here the user decides on the size of send count from each process.

```
MPI_Gatherv(send buffer, send count, send type,
recv buffer, recv counts (array), displacements
(array), recv type, root, communicator)
```

MPI: All Gather

- Gather to all. This routine gives the result of the gather operation on all nodes.

```
MPI_AllGather(send buffer, send count, send
type, recv buffer, recv count, recv type,
communicator)
```

- There is also a vector variant of all gather.

```
MPI_AllGatherv(send buffer, send count, send
type, recv buffer, recv counts (array),
displacements (array), recv type, communicator)
```

MPI: AlltoAll

- Scatter from all to all. Each process has some data, this is scattered to all processes.

```
MPI_AlltoAll(send buffer, send count, send type,
recv buffer, recv count, recv type,
communicator)
```

- There is also a vector variant of all to all.

```
MPI_AlltoAllV(send buffer, send counts (array),
send displacements (array), send type, recv
buffer, recv counts (array), receive
displacements (array), recv type, communicator)
```

MPI: AlltoAllW

- The most general variant of all to all allows the user to send different data types as well.

```
MPI_AlltoAllW(send buffer, send counts (array),
send displacements (array), send types (array),
recv buffer, recv counts (array), receive
displacements (array), recv types (array),
communicator)
```

Problems

- Use `MPI_ScatterV` to send first two columns of an 4×4 matrix to four processes.

- Use `MPI_ScatterV` to send a row of an upper triangular matrix to each process.

- Use MPI_Gather to collect rows of an 8×8 matrix scattered on 4 processors.

- Check that using `MPI_AlltoAll` on a square matrix, where every process has one row of the matrix, leads to each process getting a column of the matrix.

MPI: All Reduce

• This version of reduce is a combination of reduce and broadcast, the final result is available on all the processes.

```
MPI_AllReduce(send buffer, recv buffer, count,
data type, operation, communicator)
```

○ Check the time taken by all reduce, and a combination of reduce and broadcast. Which is faster?

MPI: Reduce Scatter

• This is a combination of reduce and scatter.

```
MPI_Reduce_Scatter(send buffer, recv buffer, recv
counts (array), data type, operation,
communicator)
```

○ The buffer is divided into disjoint sets with sizes given by the array `recv counts`.

○ It is possible to define new operations for the `reduce` family of functions/subroutines. A user defined function can be used instead of operations in MPI.

MPI: User Defined Data Types

- Create an array out of an existing array.
- Create a structure of different data types.
- Duplicate a derived data type.
- Make an array of a derived data type.

MPI: Create Vector

- Starting with an array, create a new array that contains equal sized chunks (`block size` specified by programmer) separated by a given stride. The `block size` and `stride` can be different.

```
MPI_Type_Vector(count, block length, stride, old
type, new type)
```

- ○ A new data type must be committed before it can be used.

```
MPI_Type_Commit(data type)
```

MPI: Derived Data Types

- To use the derived data type, we simply use it in place of the old data type in communication routines.

```
MPI_Send(buffer, .  .  , data type, .  .  )
```

- It is necessary to free the derived data types once we are through using these.

```
MPI_Type_Free(data type)
```

MPI: Indexed Type

```
MPI_Type_Indexed(count, block lengths (array),
displacements (array), old type, new type)
```

- Displacements are measured from the first element of the array.

○ Use this command to create a derived data type to represent:

1. Upper triangular matrix.
2. Lower triangular matrix.
3. A sparse matrix (8×8) with 11 non-zero elements.

MPI: Structures as derived data types

```
MPI_Type_Create_Struct(count, block lengths
(array), displacements (array), old data types
(array), new data type)
```

- Structures in C or FORTRAN can be sent as derived data types.

MPI: Data Packing

- Similar in concept to structures, these come in handy for sending several small variables as one packet. Very useful on low latency networks.

```
MPI_Pack(in buffer, in count, data type, out
buffer, out size, position, communicator)
```

- This function/subroutine call should be visible to all processes that may use pack/unpack data.

```
MPI_Unpack(in buffer, in size, position, out
buffer, out count, data type, communicator)
```

MPI: Groups & Communicators

• The set of processes that belong to a communicator forms a group. There is a one to one correspondence between groups and communicators. The following function/subroutine returns the group corresponding to the communicator.

```
MPI_Comm_group(communicator, group)
```

• There are many functions available for manipulating groups and group members. Functions equivalent to `Comm_size` (`Group_size`) and `Comm_rank` (`Group_rank`) are available.

MPI: Groups

• Union of groups, intersection of groups, comparison of groups, etc. are available. A group can be created as a subset of an existing group.

```
MPI_Group_incl(group, n, ranks (array), new
group)
```

○ n processes with ranks given by the array `ranks` are members of the new group.

• It is important to free a group after its use is over.

MPI: Communicators and Groups

• Communicators can be duplicated, split, etc. More importantly, a communicator can be constructed from a group.

```
MPI_Com_create(communicator, group, new
communicator)
```

◦ This creates a new communicator from a group, the group is a subgroup of the group corresponding to the original communicator.

MPI: Process Topologies

• Processes can be arranged in a virtual, Cartesian topology. The Cartesian grid of processes can be periodic, or aperiodic.

```
MPI_Cart_create(old communicator, ndims, dims,
periods, reorder, cartesian communicator)
```

• One can get the "coordinates" of a process (`MPI_Carts_coords`), or that of a neighboring processes (`MPI_Cart_shift`). There are many other, advanced functions.

MPI: File Handling

- A file can be opened simultaneously on all the processes. This functionality is not available on all implementations yet.

```
MPI_File_open(communicator, file name, access
mode, info, file handle)
```

```
MPI_File_read(file handle, buffer, count, data
type, status)
```

```
MPI_File_write(file handle, buffer, count, data
type, status)
```

```
MPI_File_close(file handle)
```

- Several other functions are also available.