**MPI_Gatherv**

Gathers (unevenly distributed) data into specified locations from all processes in a group

```c
int MPI_Gatherv ( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                  void *recvbuf, int *recvcts, int *displs,
                  MPI_Datatype recvtype,
                  int root, MPI_Comm comm )
```

**Input Parameters**
- `sendbuf`: starting address of send buffer (choice)
- `sendcount`: number of elements in send buffer (integer)
- `sendtype`: data type of send buffer elements (handle)
- `recvcounts`: integer array (of length group size) containing the number of elements that are received from each process (significant only at root)
- `displs`: integer array (of length group size). Entry `i` specifies the displacement relative to `recvbuf` at which to place the incoming data from process `i` (significant only at root)
- `recvtype`: data type of recv buffer elements (significant only at root) (handle)
- `root`: rank of receiving process (integer)
- `comm`: communicator (handle)

**Example for a 3-process problem (NP=3, rank is the rank of this process)**

```c
int cnts[3], offsets[3], i, *all_data, my_data;

cnts[0] = 100; cnts[1] = 302; cnts[2] = 467; // data distribution

my_data = malloc(sizeof(int)*cnts[rank]);
fill_my_data(my_data,cnts[rank]); // call a fcn that computes our data

if (rank == ROOT) {
    all_data = malloc(sizeof(int)*cnts[NP-1]+displs[NP-1]);
    offsets[0] = 0;
    for (i = 1; i < 3; i++)
        offset[i] = offset[i-1]+cnts[i-1];
}

MPI_Gatherv(my_data, cnts[rank], MPI_INT, all_data, cnts, offsets,
            MPI_INT, ROOT_RANK, MPI_COMM_WORLD);
```
**MPI_Gather**
Gathers together (evenly distributed) values from a group of processes

```c
int MPI_Gather ( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                 void *recvbuf, int recvcount, MPI_Datatype recvtype,
                 int root, MPI_Comm comm )
```

**Input Parameters**
- `sendbuf` starting address of send buffer (choice)
- `sendcount` number of elements in send buffer (integer)
- `sendtype` data type of send buffer elements (handle)
- `recvcount` number of elements for any single receive (integer, significant only at root)
- `recvtype` data type of recv buffer elements (significant only at root) (handle)
- `root` rank of receiving process (integer)
- `comm` communicator (handle)

**Output Parameter**
- `recvbuf` address of receive buffer (choice, significant only at root)

---

**MPI_Scatter**
Sends data from one task to all other tasks in a group

```c
int MPI_Scatter ( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                  void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                  int root, MPI_Comm comm )
```

**Input Parameters**
- `sendbuf` address of send buffer (choice, significant only at root)
- `sendcount` number of elements sent to each process (integer, significant only at root)
- `sendtype` data type of send buffer elements (significant only at root) (handle)
- `recvcount` number of elements in receive buffer (integer)
- `recvtype` data type of receive buffer elements (handle)
- `root` rank of sending process (integer)
- `comm` communicator (handle)

**Output Parameter**
- `recvbuf` address of receive buffer (choice)
**MPI_Reduce**
Reduces values on all processes to a single value

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

**Input Parameters**
- `sendbuf`: address of send buffer (choice)
- `count`: number of elements in send buffer (integer)
- `datatype`: data type of elements of send buffer (handle)
- `op`: reduce operation (handle)
- `root`: rank of root process (integer)
- `comm`: communicator (handle)

**Output Parameter**
- `recvbuf`: address of receive buffer (choice, significant only at root)

**MPI_Scan**
Computes the scan (partial reductions) of data on a collection of processes

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

**Input Parameters**
- `sendbuf`: starting address of send buffer (choice)
- `count`: number of elements in input buffer (integer)
- `datatype`: data type of elements of input buffer (handle)
- `op`: operation (handle)
- `comm`: communicator (handle)

**Output Parameter**
- `recvbuf`: starting address of receive buffer (choice)
**MPI_Send**
Performs a basic send (and may block until the message is received)

```c
int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,
        int tag, MPI_Comm comm )
```

**Input Parameters**
- **buf**: initial address of send buffer (choice)
- **count**: number of elements in send buffer (nonnegative integer)
- **datatype**: datatype of each send buffer element (handle)
- **dest**: rank of destination (integer)
- **tag**: message tag (integer)
- **comm**: communicator (handle)

**MPI_Recv**
Basic receive

```c
int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,
        int tag, MPI_Comm comm, MPI_Status *status )
```

**Output Parameters**
- **buf**: initial address of receive buffer (choice)
- **status**: status object (Status)

**Input Parameters**
- **count**: maximum number of elements in receive buffer (integer)
- **datatype**: datatype of each receive buffer element (handle)
- **source**: rank of source (integer)
- **tag**: message tag (integer)
- **comm**: communicator (handle)

**Notes**
The count argument indicates the maximum length of a message; the actual number can be determined with MPI_Get_count.