MPI send/ receive

MPI Point to Point Communication

MPI point-to-point operations typically involve message passing between two, and only two, different MPI tasks. One task is performing a send operation and the other task is performing a matching receive operation. Different types of send and receive routines:

- Synchronous send
- Blocking send / blocking receive
- Non-blocking send / non-blocking receive
- Buffered send
- Combined send/receive
- “Ready” send

Any type of send routine can be paired with any type of receive routine.

MPI Send / Receive

MPI point-to-point communication routines generally have an argument list that takes one of the following formats:

```
MPI_Send (&buf,count,datatype,dest,tag,comm)
MPI_SEND (buf,count,datatype,dest,tag,comm,ierr)
```

Buffer: Program address space that references the data that is to be sent or received. In most cases, this is simply the variable name that is be sent/received. For **C programs**, this argument is passed by reference and usually must be prepended with an ampersand: `&var1`.

Data Count: Indicates the number of data elements of a particular type to be sent.

Data Type: For reasons of portability, MPI predefines its elementary data types.

- `MPI_CHAR` – signed char
- `MPI_INT` – signed int
- `MPI_FLOAT` – float
- `MPI_DOUBLE` – double

You can also create your own derived data types.

Destination: An argument to send routines that indicates the process where a message should be delivered. Specified as the rank of the receiving process.

Source: An argument to receive routines that indicates the originating process of the message. Specified as the rank of the sending process. This may be set to the wild card `MPI_ANY_SOURCE` to receive a message from any task.

Tag: Arbitrary non-negative integer assigned by the programmer to uniquely identify a message. Send and receive operations should match message tags. For a receive operation, the wild card `MPI_ANY_TAG` can be used to receive any message regardless of its tag. The MPI standard guarantees that `int` can be used as **tags**, but most implementations allow a much larger range than this.

Communicator: Indicates the communication context, or set of processes for which the source or destination fields are valid. Unless the programmer is explicitly creating new communicators, the predefined communicator `MPI_COMM_WORLD` is usually used.

Status: For a receive operation, indicates the source of the message and the tag of the message. In `C`, this argument is a pointer to a predefined structure `MPI_Status` (ex. `stat.MPI_SOURCE` `stat.MPI_TAG`). Additionally, the actual number of bytes received are obtainable from Status via the `MPI_Get_count` routine.

Request: Used by non-blocking send and receive operations. Since non-blocking operations may return before the requested system buffer space is obtained, the system issues a unique **"request number"**. The programmer uses this system assigned **"handle"** later (in a `WAIT` type routine) to determine completion of the non-blocking operation. In `C`, this argument is a pointer to a predefined structure `MPI_Request`.

Blocking sends: `MPI_Send(buffer,count,type,dest,tag,comm)` Non-blocking sends: `MPI_Isend(buffer,count,type,dest,tag,comm,request)` Blocking receive: `MPI_Recv(buffer,count,type,source,tag,comm,status)` Non-blocking receive: `MPI_Irecv(buffer,count,type,source,tag,comm,request)`

MPI_Send:Basic blocking send operation. Routine returns only after the application buffer in the sending task is free for reuse.

```
MPI_Send (&buf,count,datatype,dest,tag,comm)
MPI_SEND (buf,count,datatype,dest,tag,comm,ierr)
MPI_Recv (&buf,count,datatype,source,tag,comm,&status)
MPI_RECV (buf,count,datatype,source,tag,comm,status,ierr)
```

Synchronous blocking send:Send a message and block until the application buffer in the sending task is free for reuse and the destination process has started to receive the message.

```
MPI_Ssend (&buf,count,datatype,dest,tag,comm)
MPI_SSEND (buf,count,datatype,dest,tag,comm,ierr)
```

Buffered blocking send*:permits the programmer to allocate the required amount of buffer space into which data can be copied until it is delivered. Insulates against the problems associated with insufficient system buffer space.

```
MPI_Bsend (&buf,count,datatype,dest,tag,comm)
MPI_BSEND (buf,count,datatype,dest,tag,comm,ierr)
```