Intermediate MPI (Message-Passing Interface)

What happens when a process sends a message?

Suppose process 0 wants to send a message to process 1. Three possibilities:

- Process 0 can stop and wait until Process 1 is ready to receive the message.
- Process 0 can copy the message into a buffer (internal to the library or user-specified) and return from the `MPI_Send` call.
- It can report failure.

An MPI implementation is allowed to use the first or second interpretation but is not required to use the second one.

Dealing with buffering in MPI

How do we ensure that the parallel program works correctly without depending upon the amount of buffering, if any, provided by the message passing system?

- **Ordered send and receive**. For example even processes send first while odd processes receive first.
- **Combined send and receive**. MPI provides a combined function `MPI_Sendrecv` that allows us to send and receive data without worrying about deadlock from a lack of buffering.
- **Use buffered sends**. We provide the buffering.
- **Use nonblocking communication**. This can often give the best performance, especially if we use it to overlap communication and computation.
- **Use synchronous sends**. MPI provides `MPI_Ssend`. Send doesn't return until the destination process starts receiving the message. However, this can have performance and scalability issues.

Send communication modes

- Standard Mode - Not assumed that corresponding receive routine has started. Amount of buffering not defined by MPI. If buffering provided, send could complete before receive reached.
- Buffered Mode - Send may start and return before a matching receive. Necessary to specify buffer space via routine MPI_Buffer_attach().
- Synchronous Mode - Send and receive can start before each other but can only complete together.
- Ready Mode - Send can only start if matching receive already reached, otherwise error. Use with care.

More on Send communication modes

- Each of the four modes can be applied to both blocking and nonblocking send routines.
- Only the standard mode is available for the blocking and nonblocking receive routines.
- Any type of send routine can be used with any type of receive routine.

Buffered Send

- Prototypes.

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

int MPI_Buffer_attach(void *buffer, int size)
int MPI_Buffer_detach(void *buffer, int *size)
```

- MPI_Bsend allows the user to send messages without worrying about where they are buffered (because the user must have provided buffer space with MPI_Buffer_attach).
- The buffer size given should be the sum of the sizes of all outstanding Bsends that you intend to have, plus MPI_BSEND_OVERHEAD for each Bsend that will be done.
- MPI_Buffer_detach returns the buffer address and size so that nested libraries can replace and restore the buffer.

MPI Nonblocking routines

- Nonblocking send - MPI_Isend(...) - will return “immediately” even before source location is safe to be altered.
- Nonblocking receive - MPI_Irecv(...) - will return even there is no message to accept.

Nonblocking routine formats

MPI_Isend(buf,count,datatype,dest,tag,comm,request)

MPI_Irecv(buf,count,datatype,source,tag,comm, request)

Completion detected by MPI_Wait() and MPI_Test().

MPI_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Waitall(int count, MPI_Request array_of_requests[], MPI_Status array_of_statuses[])

int MPI_Waitany(int count, MPI_Request array_of_requests[], int *index, MPI_Status *status)

MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

MPI_Isend example

```
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* find rank */
if (myrank == 0) {
    int x;
    MPI_Isend(&x,1,MPI_INT, 1, msgtag, MPI_COMM_WORLD, req1);
    compute();
    MPI_Wait(req1, status);
} else if (myrank == 1) {
    int x;
    MPI_Recv(&x,1,MPI_INT,0,msgtag, MPI_COMM_WORLD, status);
}
```

Sending/Receiving structures (Part 1)

- We can send a structure by packing it as an array of bytes:

```
struct test {
    int n;
    double x[3], y[3];
};

if (pid == source) {
    struct test test1;
    MPI_Send(&test1, sizeof(struct test), MPI_BYTE, destination,
             tag, MPI_COMM_WORLD);
} else (pid == destination) {
    struct test test2;
    MPI_Recv(&test2, sizeof(struct test), MPI_BYTE, source, tag,
             MPI_COMM_WORLD, status);
}
```

- However, this relies on the layout of the structure being the same on all nodes. It also obfuscates the code and introduces platform dependency so it is not a recommended practice for MPI programs.

Sending/Receiving structures (Part 2)

- We can send a structure by creating a custom MPI data type for the structure

```
struct test {
       int n;
       double x[3], y[3];
};
const int nitems = 3;
int blocklengths[3] = {1, 3, 3}; //lengths of i, x and y as #items
MPI_Datatype types[3] = {MPI_INT, MPI_DOUBLE, MPI_DOUBLE};
MPI_Aint offsets[3];
MPI_Datatype mpi_test_type;
offsets[0] = offsetof(struct test, n);
offsets[1] = offsetof(struct test, x);
offsets[2] = offsetof(struct test, y);
MPI_Type_create_struct(nitems, blocklengths, offsets, types,
                       &mpi_test_type);
MPI_Type_commit(&mpi_test_type);
```

- See example lab/MPI/send-struct/ for a working example. This is the recommended way of sending a structure in MPI.
- Note that there is no way to send a structure that has variable length (because of pointers stored in it) in one message. We have to use two messages.