Non-Blocking Communication

- Non-Blocking communications
- Non-Blocking send
- Non-Blocking receive
- Handles
- Non-Blocking synchronous send
- Non-Blocking receive
- Blocking and non-blocking

- Routine names
- Completion tests
- Wait/Test routines
- Multiple communications
- Testing multiple non-blocking communications
- Class Exercise: Calculating Ring
Non-Blocking communications

Separate communication into three phases:

1. Initiate non-blocking communication ("post" a send or receive)

2. Do some other work not involving the data in transfer
   - Overlap calculation and communication
   - Latency hiding

3. Wait for non-blocking communication to complete
Non-Blocking send
Non-Blocking receive
Handles used for non-blocking communication

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Same as for blocking (MPI_Datatype or INTEGER)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communicator</td>
<td>Same as for blocking (MPI_Comm or INTEGER)</td>
</tr>
<tr>
<td>Request</td>
<td>MPI_Request or INTEGER</td>
</tr>
</tbody>
</table>

- A request handle is allocated when a non-blocking communication is initiated
- The request handle is used for testing if a specific communication has completed
Non-Blocking synchronous send

C:

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

Fortran:

```fortran
CALL
    MPI_ISSSEND(BUF,COUNT,DATATYPE,DEST,TAG,COMM,REQUEST,IERROR)
```

```
<type>  BUF(*)
INTEGER COUNT,DATATYPE,DEST,TAG,COMM
INTEGER REQUEST,IERROR
```

“_I” for “Immediate” because they return more or less immediately
Non-Blocking receive

C:

```c
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype,
               int source, int tag, MPI_Comm comm, MPI_Request *request)
```

Fortran:

```fortran
CALL MPI_IRECV(BUF,COUNT,DATATYPE,SOURCE,TAG,COMM,REQUEST,IERROR)
```

Note: no STATUS argument

system-defined, not directly accessible by programmer
Blocking and non-blocking

• Send and receive can be blocking or non-blocking

• A blocking send can be used with a non-blocking receive, and vice-versa, e.g.,

  MPI_Isend  \rightarrow  MPI_Recv

• Non-blocking sends can use any mode - synchronous, buffered, standard or ready

Note: there is no advantage for buffered or ready modes
## Routine names

<table>
<thead>
<tr>
<th>Non-Blocking Operation</th>
<th>MPI Call</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard send</td>
<td>MPI_ISEND</td>
</tr>
<tr>
<td>Synchronous send</td>
<td>MPI_ISSEND</td>
</tr>
<tr>
<td>Buffered send</td>
<td>MPI_IBSEND</td>
</tr>
<tr>
<td>Ready send</td>
<td>MPI_IRSEND</td>
</tr>
<tr>
<td>Receive</td>
<td>MPI_IRECV</td>
</tr>
</tbody>
</table>

“\( I \)”: Immediate
Completion tests

• Waiting vs. Testing

• **Wait:** routine does not return until completion finished

• **Test:** routine returns a TRUE or FALSE (0) value depending on whether the communication has completed
Wait/Test routines

C:

```c
int MPI_Wait(MPI_Request *request, MPI_Status *status)
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
```

Fortran:

```fortran
CALL MPI_WAIT(REQUEST, STATUS, IERR)
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERR

CALL MPI_TEST(REQUEST, FLAG, STATUS, IERR)
LOGICAL FLAG
INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERR
```

Here is where `STATUS` appears. It is a three-element structure in C and an integer array of size `MPI_STATUS_SIZE` (defined in `mpif.h`) in Fortran.
Multiple communications

- Test or wait for completion of one (and only one) message:
  - int MPI_Waitany(…)
  - int MPI_Testany(…)

- Test or wait for completion of all messages:
  - int MPI_Waitall(…)
  - int MPI_Testall(…)

- Test or wait for completion of as many messages as possible:
  - int MPI_Waitsome(…)
  - int MPI_Testsome(…)
Class Exercise: Calculating Ring

• Repeat the Processor Ring exercise, this time using non-blocking communication routines

• In addition, each processor should calculate the sum of all the ranks as it receives them