Non-blocking point-to-point messages

- MPI_Isend and MPI_Irecv
- Two parts:
  - post the operation
  - Wait for results: need to call MPI_Wait or MPI_Test
- Can help with overlapping computation with communication
MPI_Isend

int MPI_Isend( const void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request )

buf: address of send buffer

count: number of elements in send buffer

datatype: datatype of each send buffer element

dest: rank of destination process

tag: message tag

comm: communicator

request: communication request handle
MPI_Irecv

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

buf: address of receive buffer

count: maximum number of elements in receive buffer

datatype: datatype of each receive buffer element

source: rank of source process

tag: message tag

comm: communicator

request: communication request handle
MPI_Wait

int MPI_Wait( MPI_Request *request, MPI_Status *status )

request: communication request

status: status object

• Status object can provide information about:
  • Source process for a message: status.source
  • Message tag: status.tag
  • Number of elements: MPI_Get_count( MPI_Status *status, MPI_Datatype datatype, int count)

• If you don’t want to inspect it, you can use MPI_STATUS_IGNORE
Using non-blocking send/recv

```c
int main(int argc, char *argv) {
    ...
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    ...
    if (rank % 2 == 0) {
        data = rank;
        MPI_Isend(&data, 1, MPI_INT, rank+1, 0, ...);
    } else {
        data = rank * 2;
        MPI_Irecv(&data, 1, MPI_INT, rank-1, 0, ...);
    }
    ...
    MPI_Wait(&req, &stat);
    printf("Process %d received data %d\n", data);
}
...}
```
Announcements

• Assignment 0 is posted, and due on Thursday
  • Not graded, but you have to submit through gradescope

• Assignment 1 will be posted on Monday
  • You will have 2 weeks to complete it

• Quiz 0 is done
  • Questions?

• Quiz 1 next week
  • You will have a 24 hour period in which to do it, and the date will be announced in ELMS, but will not be before next class (likely released next Wed. morning)
int main(int argc, char *argv) {
    ...

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Irecv(&data1, 16, MPI_DOUBLE, (rank-1)%4, 0, ...);
    MPI_Irecv(&data2, 16, MPI_DOUBLE, (rank+1)%4, 0, ...);

    MPI_Isend(&data3, 16, MPI_DOUBLE, (rank-1)%4, 0, ...);
    MPI_Isend(&data4, 16, MPI_DOUBLE, (rank+1)%4, 0, ...);

    MPI_Waitall(...);

    compute();

    ...
}
Other MPI calls

- int MPI_Test( MPI_Request *request, int *flag, MPI_Status *status )
  
  Returns flag=true (non-zero) if the operation has completed

  Use MPI_STATUS_IGNORE if you don’t need to look at the status field

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

  MPI_Waitany

  MPI_Waitsome

  Equivalent calls for test – MPI_Testall, MPI_Testany, MPI_Testsome
Collective operations

• All processes in the communicator participate in the operation

• And all processes in the communicator must make collective calls in the same order
  • Otherwise no guarantee the operations will complete correctly

• Not all collective operations require synchronization/barrier
  • A process can return from a collective operation when it completes whatever work it needs to do for the operation
Collective operations

• int MPI_Barrier( MPI_Comm comm)
  • Blocks until all processes in the communicator have reached this routine

• int MPI_Bcast( void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm )
  • Send data from root to all processes

• int MPI_Reduce( const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm )
  • Reduce data from all processes to the root
Collective operations

- int MPI_Scatter( const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  - Send data from root to all processes

- int MPI_Gather( const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

  - Gather data from all processes to the root

- MPI_Scan

https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/
Other MPI calls

- **MPI_Wtime**
  - Returns elapsed time

```c
{
  double starttime, endtime;
  starttime = MPI_Wtime();
  ....  code region to be timed  ...
  endtime = MPI_Wtime();
  printf("Time %f seconds\n", endtime-starttime);
}
```
Calculate the value of \[ \pi = \int_0^1 \frac{4}{1 + x^2} \]

```c
int main(int argc, char *argv[]) {
    ...

    n = 10000;

    h   = 1.0 / (double) n;
    sum = 0.0;

    for (i = 1; i <= n; i += 1) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    ...
}
```
Calculate the value of \[ \pi = \int_0^1 \frac{4}{1 + x^2} \]

```c
int main(int argc, char *argv[]) {
    ...

    n = 10000;
    MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

    h   = 1.0 / (double) n;
    sum = 0.0;

    for (i = myrank + 1; i <= n; i += numranks) {
        x = h * ((double)i - 0.5);
        sum += (4.0 / (1.0 + x * x));
    }
    pi = h * sum;

    MPI_Reduce(&pi, &globalpi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    ...
}
```
Protocols for sending message

- **Eager**
  - Message sent assuming destination can store

- **Rendezvous**
  - Message only sent after handshake (receiving ack) with destination

- **Short**
  - Data sent with the message envelope
Other MPI send modes

- Basic mode:
  - MPI_Send

- Buffered mode:
  - MPI_Bsend
  - Use MPI_Buffer_attach to provide space for buffering

- Synchronous mode:
  - MPI_Ssend

- Ready mode:
  - MPI_Rsend

https://www.mcs.anl.gov/research/projects/mpi/sendmode.html