Implementing Master/Slave Algorithms

- Many algorithms have one or more master processes that send tasks and receive results from slave processes
- Because there is one (or a few) controlling processes, the master can become a bottleneck

Skeleton Master Process

```
do while(.not. Done)
     ! Get results from anyone
     call MPI_Recv( a,…, status, ierr )
     ! If this is the last data item,
     ! set done  to .true.
     ! Send more work to them
     call MPI_Send( b, …, status(MPI_SOURCE), &
                   … , ierr )
enddo
```

- Not included:
  - » Sending initial work to all processes
  - » Deciding when to set done

Skeleton Slave Process

- ```
  Do while (.not. Done)
       ! Receive work from master
       call MPI_Recv( a, …, status, ierr )
       … compute for task
       ! Return result to master
       call MPI_Send( b, …, ierr )
  enddo
  ```
- Not included:
  - » Detection of termination (probably message from master)

3

Problems With Master/Slave

- Slave processes have nothing to do while waiting for the next task
- Many slaves may try to send data to the master at the same time
  - » Could be a problem if data size is very large, such as 20-100 MB
- Master may fall behind in servicing requests for work if many processes ask in a very short interval
- Presented with many requests, master may not evenly respond

4
  ```

Spreading out communication

- Use double buffering to overlap request for more work with work

```
Do while (.not. Done)
    ! Receive work from master
    call MPI_Wait( request, status, ierr )
    ! Request MORE work
    call MPI_Send( …, send_work, …, ierr )
    call MPI_IRecv( a2, …, request, ierr )
    … compute for task
    ! Return result to master (could also be nb)
    call MPI_Send( b, …, ierr )
enddo
```

- MPI_Cancel
  - » Last Irecv may never match; remove it with MPI_Cancel
  - » MPI_Test_cancelled required on IBM (!), then MPI_Request_free

5

Limiting Memory Demands on Master

- Using MPI_Ssend and MPI_Issend to encourage limits on memory demands
  - » MPI_Ssend and MPI_Issend do *not* specify that the data itself doesn't move until the matching receive is issued, but that is the easiest way to implement the synchronous send operations
  - » Replace MPI_Send in slave with
    - – MPI_Ssend for blocking
    - – MPI_Issend for nonblocking (even less synchronization)

6

Distributing work further

- Use multiple masters, slaves select a master to request work from at random
- Keep more work locally
- Use threads to implement work stealing (threads discussed later)

7

Fairness in Message Passing

- What happens in this code:

```
if (rank .eq. 0) then
  do i=1,1000*(size-1)
    call MPI_Recv( a, n, MPI_INTEGER,&
        MPI_ANY_SOURCE, MPI_ANY_TAG, comm,&
        status, ierr )
    print *,' Received from',status(MPI_SOURCE)
  enddo
else
  do i=1,1000
    call MPI_Send( a, n, MPI_INTEGER, 0, i, &
                   comm, ierr )
  enddo
endif
```

- In what order are messages received?

8

Fairness

- MPI makes no guarantee, other than that all messages will be received.
- The program could
  - » Receive all from process 1, then all from process 2, etc.
  - » That order would starve processes 2 and higher of work in a master/slave method
- How can we encourage or enforce fairness?

9

MPI Multiple Completion

- Provide one Irecv for each process:

```
do i=1,size-1
    call MPI_Irecv(…, req(i), ierr)
enddo
```

- Process *all* completed receives (wait guarantees at least one):

```
call MPI_Waitsome( size-1, req, count,&
   array_of_indices, array_of_statuses, ierr )
do j=1,count
   ! Source of completed message is
    … array_of_statuses(MPI_SOURCE,j)
   ! Repost request
   call MPI_Irecv( …, req(array_of_indices(j)),)
enddo
```

10

Exercise: Providing Fairness

- Write a program with one master; have each slave process receive 10 work requests and send 10 responses. Make the slaves' computation trivial (e.g., just return a counter) Use the simple algorithm (MPI_Send/MPI_Recv)
  - » Is the MPI implementation fair?
- Write a new version using MPI_Irecv/MPI_Waitall
  - » Be careful of uncancelled Irecv requests
  - » Be careful of the meaning of array_of_indices (zero versus one origin)

11