Programming with MPI

Advanced Completion Issues

Nick Maclaren

nmm1@cam.ac.uk

May 2008

More on Completion

More on Point–to–Point made simplifying assumptions
This describes when those are not so
Three more advanced features complicate things

- Waiting for a subset of requests (described shortly)

- Cancellation of requests (described shortly)
- Persistent requests (described in a later lecture)

I don't recommend using any of these
But this is a description of the issues

Empty Statuses

MPI has the concept of an empty status

An empty status looks like the following:

- The tag is MPI_ANY_TAG
- The source is MPI_ANY_SOURCE
  [$\Rightarrow$ or possibly MPI_PROC_NULL]
- MPI_Get_count returns zero

And, for properties we haven't covered yet:

- The error code is MPI_SUCCESS
- MPI_Test_cancelled returns False

Completion of Subsets (1)

MPI_Testsome and MPI_Waitsome

These check for or complete some of the requests
    and return a count of how many
    plus arrays of indices and statuses

For wait and when test's flag is True:

The index array lists the completed requests
First count elements of the status array are set
The other statuses are not defined

Completion of Subsets (2)

If not enough of the requests are ready

- The tests set their flag to False
- The waits hang until something happens

If enough of the requests are ready

- Any completes just one request
- Some/all complete all ready requests
- The tests set their flag to True

All completed requests are released exactly
as for the individual request forms

Error Codes (1)

What if not using MPI_ERRORS_ARE_FATAL?
Multiple errors from the all and some forms

One of the many reasons the default is easiest

The error code may be MPI_ERR_IN_STATUS

The individual error codes are in the statuses
Including the empty statuses of the all forms

Error Codes (2)

<status array> (MPI_ERROR , <index>) (Fortran)
<status array> [<index>] . MPI_ERROR (C)

The MPI_ERROR fields are set if and only if:

- You call one of the all or some forms
- Its error code is MPI_ERR_IN_STATUS

That field is never set for the any forms
I.e. exactly like the individual request forms
They will never return MPI_ERR_IN_STATUS

Fortran Multiple Errors

```
INTEGER :: i , error , requests ( 100 ) ,        &
        statuses ( MPI_STATUS_SIZE , 100 )

CALL MPI_Waitall ( 100 , requests , statuses , error )
IF ( error == MPI_ERR_IN_STATUS ) THEN
        DO i = 1 , 100
                IF ( statuses ( MPI_ERROR , i ) /=        &
                                MPI_SUCCESS ) THEN
                        CALL fail ( statuses ( MPI_ERROR , i ) )
                END IF
        END DO
ELSE IF ( error /= MPI_SUCCESS ) THEN
        CALL fail ( error )
END IF
```

C Multiple Errors

```
int i , error , requests [ 100 ] ;
MPI_Status statuses [ 100 ] ;

error = MPI_Waitall ( 100 , requests , statuses ) ;
if ( error == MPI_ERR_IN_STATUS ) {
    for ( i = 1 ; i < 100 ; ++i ) {
        if ( statuses[i] . MPI_ERROR  !=
                MPI_SUCCESS )
            fail ( statuses[i] . MPI_ERROR )
    }
else if ( error != MPI_SUCCESS )
    fail ( error ) ;
```

Completion Oddities (1)

There are actually some exceptions to the above

- You can avoid them by not causing them

The facilities are described (briefly) later

- Persistent requests are not released
  you have to release them yourself

This course doesn't describe these in detail

- Cancellation is different from completion
  the request merely becomes inactive

You still have to complete or release it

Completion Oddities (2)

Requests become inactive in only three ways:

1. Setting MPI_REQUEST_NULL explicitly
2. Passing an already completed request
3. Using cancellation (see later)

Multiple completion unavoidably causes 2

- Either remove them from the request array
- Or you can learn more about the functions

It isn’t hard, but each group is different

Inactive Requests (1)

We first consider the individual request forms

Wait and test work on inactive requests

- they return immediately and successfully
- the status is set to empty

Inactive Requests (2)

We next consider the any forms

If none of the requests are active
Including the case of a zero length request array

- they return successfully and immediately
- the index is set to MPI_UNDEFINED
- the status is set to empty

Otherwise, they consider just the active requests

I.e. very like the individual request forms

Inactive Requests (3)

We now consider the all forms

If none of the requests are active
Including the case of a zero length request array

- they return successfully and immediately

Otherwise, they consider just the active requests

- In both cases, all statuses corresponding to inactive requests are set to empty

Inactive Requests (4)

We last consider the some forms

If none of the requests are active
Including the case of a zero length request array

- they return successfully and immediately
- the index count is set to MPI_UNDEFINED

Otherwise, they consider just the active requests

- The index array is only completed requests
  i.e. ones completed by this call
- Only completed requests have statuses

Inactive Requests (5)

The above all looks like unnecessary complexity

- But it isn’t – MPI has got it right

It means that you can write clean, obvious code
And everything will all work as it should

Cancellation (1)

This is just an overview of the facility

You may need to abandon active requests
⇒ Try to avoid ever getting into that hole

- Cancellation is for exceptional circumstances

It may be both unreliable and inefficient

MPI_Cancel will start the cancellation

- It will not release the request

Cancellation (2)

You must still call MPI_Wait or MPI_Test
Or one of the request array versions of those

- MPI_Test_cancelled checks the status

Returns a flag saying if the cancellation succeeded

- If you use cancellation, test that first

All other status fields are undefined if cancelled

Cancellation (3)

You can also simply release the request
By calling MPI_Request_free

You can also call this on active requests
They will be disconnected, but will complete

- DON'T do that – not even for sends

You have no way of telling when they complete
And what happened when they finally do