Programming with MPI

Error Handling

Nick Maclaren

nmm1@cam.ac.uk

May 2008

Error Handling (1)

Most standards get this hopelessly wrong
MPI gets it at least half right, as follows:

- All invalid uses are defined to be erroneous

Implementations are encouraged to detect them

- Most undefined results are detectable

E.g. they are set to a special, invalid value

- Most nonsense is defined to be erroneous

E.g. you cannot legally create a deadlock

Error Handling (2)

- No concept of conforming but undefined

I.e. it's a valid program with no known meaning

C is infested with it, Fortran/C++ use it
MPI has it, but only as bugs in the standard

- The default error handling is to stop

Not handling errors is fairly fail–safe in MPI

Error Handling (3)

However, all is not sweetness and light:

- Implementations not required to detect errors

Some errors are usually detected, others rarely are

- MPI has not specified a debugging mode

Error detection is at the whim of your implementor

Ones that can be detected locally usually are

e.g. providing an out–of–range process number

Inconsistencies across collectives may be

Error Handling (4)

- Some errors are almost undetectable

They include most language/MPI interface ones
E.g. incorrect datatype for the buffer type

- Non–MPI ones are obviously not handled

And they may cause MPI to fail horribly

MPI can't fix up other standards' defects!

Changing Error Handling

- There are programmable error handling facilities

But they don't allow actual recovery
Ask me if you want to know why this is unavoidable

You can use them only for cleaning–up
Including writing your own diagnostics

- But do look at the implementation documents

MPI encourages documented enhancements

Simple Error Handling (1)

There are several predefined error handlers

MPI_ERRORS_ARE_FATAL is the default

MPI_ERRORS_RETURN returns an error code
Fortran last argument, and C function result

Simple Error Handling (2)

- You attach the setting to a communicator

I recommend setting it early, and setting it once
And setting it consistently across processes

The call to do that is one that has changed name:

MPI_Comm_set_errhandler (new name)

MPI_Errhandler_set (old name)

If an MPI function returns an error code

i.e. anything that isn't MPI_SUCCESS

Call your code to diagnose, clean–up and stop

Simple Error Handling (3)

Error codes are implementation dependent

- A function to map them into an error string

You should use this for reasonable diagnostics

You do this by calling MPI_Error_string

Maps the error code to a textual message

Maximum length MPI_MAX_ERROR_STRING

- This is a block of text and not a C string

The length is returned via a separate argument

Warning – Edge of Cliff

- MPI_ERRORS_RETURN is dangerous

You must test for errors in ALL calls
One undetected error will cause chaos later

- But this facility can be very useful

You can write your own, helpful diagnostics
You can flush all your output to files
You can tidy up external state and not just crash

- Also can be used temporarily for debugging

Best to set the mode just around the failing call

Fortran Error Handling (1)

```
INTEGER :: error

CALL MPI_Comm_set_errhandler (       &
     MPI_COMM_WORLD ,       &
     MPI_ERRORS_RETURN , error )
```

Old versions of gfortran do not support this
There was a bug in generic resolution handling
There is a truly mind–boggling bypass if you need it

Fortran Error Handling (1.5)

This code works – heaven alone knows why!
It's harmless (but unhelpful) on other systems

```
INTEGER :: error, junk

junk = MPI_COMM_WORLD
CALL MPI_Comm_set_errhandler (       &
     junk ,       &
     MPI_ERRORS_RETURN , error )
```

Fortran Error Handling (2)

```
INTEGER :: error , length , temp
CHARACTER        &
    ( LEN = MPI_MAX_ERROR_STRING) :: message

< call some MPI function >
IF ( error /= MPI_SUCCESS ) THEN
       CALL MPI_Error_string ( error ,       &
              message , length , temp )
       PRINT * , message(1:length)
      CALL MPI_Abort ( MPI_COMM_WORLD ,       &
              1 , temp )
END IF
```

C Error Handling (1)

```
int error ;

error = MPI_Comm_set_errhandler (
        MPI_COMM_WORLD ,
        MPI_ERRORS_RETURN ) ;
```

C Error Handling (2)

```
int error , length ;
char message [ MPI_MAX_ERROR_STRING + 1 ] ;

< call some MPI function >
if ( error != MPI_SUCCESS ) {
        MPI_Error_string ( error , message ,
                & length ) ;
        procname [ length ] = '\0' ;
        printf ("%s\n" , message) ;
        MPI_Abort ( MPI_COMM_WORLD , 1 ) ;
 }
```

Note the way that the length is returned

Advanced Error Handling (1)

Can also map error codes into error classes
with the function MPI_Error_class

A documented set of 60 distinct values
with names MPI_ERR_...
Use these to distinguish various types of error

You may want to do this, for advanced handling
I can't offhand imagine why, but it's there

Advanced Error Handling (2)

Can define your own error handlers
Just functions to call when there is an error

- Safer than using MPI_ERRORS_RETURN

Provided that you code the function carefully

Just the same logic as for MPI_ERRORS_RETURN
in the examples that were given above

This course doesn't cover it, for simplicity
Experienced programmers will have no trouble

Advanced Error Handling (3)

Functions are:

- MPI_Comm_create_errhandler (new name)
- MPI_Errhandler_create (old name)
- MPI_Errhandler_free

C type name MPI_Handle_function

And a MPI constant MPI_ERRHANDLER_NULL

Very Advanced Error Handling

- I recommend not doing any of this

Even most experts will never need or want to

Error handling is, in fact, purely local
Every process can have a different handler
Actually, every communicator in every process ...
Also, MPI 2 extended it to some other classes

You can also change it whenever you want
For saving and restoring the old one, you need:

MPI_Comm_Get_errhandler (new name)

MPI_Errhandler_get (old name)

Epilogue

That's more–or–less all there is to MPI errors

The only exercise is to try the simple case out