Programming with MPI

Other Features Not Covered

Nick Maclaren

Computing Service

nmm1@cam.ac.uk, ext. 34761

May 2008

The Beginning of the End

This mentions things you may need to know about

- Some are very esoteric and few people use them

But you may be one of the very few who needs to

- Others should be avoided like the plague

But may be recommended in books and on the Web

Just note them, and come back if you need to

Accumulating Reduction

This is where process N receives
the reduction from processes 0…N

I have no idea why MPI calls it prefix reduction
Or why the function is called MPI_Scan

You use it exactly like MPI_Reduce
Except that it may be quite a lot slower

MPI–2 adds an exclusive scan (MPI_Exscan)
[MPI_Scan is inclusive]
Some things you can't do with inclusive scans

User-Defined Reduce Operations

Can define your own global reduction operations
Few people want to, but can sometimes be needed
Probably useful only for derived types

ScaLAPACK does for complex reductions in C
Don't ask me why – I could do its job more simply
Please ask me for help with complex in C/C++

Functions are MPI_Op_create and MPI_Op_free
And a C opaque type and C++ class MPI_Op

User-Defined Attributes (1)

You can define your own attributes
Associate them with communicators

Can ensure they are copied and freed correctly
whenever a communicator is copied or freed

- A bit cleaner than using global variables

All people writing MPI libraries should use them

Peter Pacheco likes them – see that reference
I have omitted them only for simplicity

User-Defined Attributes (2)

This can all be done in MPI–1, as well
But I shall give the new (recommended) names

Relevant MPI–2 function names:

- MPI_Comm_create_keyval
- MPI_Comm_delete_attr
- MPI_Comm_free_keyval
- MPI_Comm_get_attr
- MPI_Comm_put_attr

User-Defined Attributes (3)

Associated definitions:

MPI_Comm_copy_attr_function
MPI_Comm_delete_attr_function
COMM_COPY_ATTR_FUNCTION
MPI_COMM_DUP_FN
MPI_COMM_NULL_COPY_FN
MPI_COMM_NULL_DELETE_FN
COMM_DELETE_ATTR_FUNCTION

User-Defined Attributes (4)

You can set callback functions using attributes
Very useful for cleaning up in library code

That sort of thing is way beyond this course!

Please ask if you want to know about it

Ready Mode

There is a ready mode, for dubious reasons
Send works only if the receive is ready
Theoretically, it might be more efficient

- I don't recommend using this feature, ever

A late receive is undefined behaviour
Unlikely to get an error – just chaos

Functions are MPI_Irsend and MPI_Rsend

Don't use MPI_Rsend_init, either (next slide)

Persistent Communications

You can define persistent point–to–point
Just might be faster on some implementions

You initialise some requests, once only
and then use them multiple times

Relevant functions:

MPI_Bsend_init MPI_Send_init MPI_Startall
MPI_Recv_init MPI_Ssend_init
MPI_Rsend_init MPI_Start

Reduce-Scatter

A bizarre function called MPI_Reduce_scatter

Equivalent to MPI_Reduce + MPI_Scatterv

It is provided in case it can be optimised better

I have scratched my head and can't see how or why

Consider it, if it is exactly what you want

Otherwise I suggest ignoring it completely

MPI Derived Types (1)

These have also been renamed by MPI–2
Relevant new (recommended) function names:

- MPI_Get_address
- MPI_Type_create_hindexed
- MPI_Type_create_hvector
- MPI_Type_create_resized
- MPI_Type_create_struct
- MPI_Get_elements
- MPI_Pack
- MPI_Pack_size
- MPI_Type_contiguous
- MPI_Type_get_extent
- MPI_Type_indexed
- MPI_Type_size
- MPI_Type_vector

MPI Derived Types (2)

The C/C++ opaque type is MPI_Datatype

Associated definitions:

MPI_BOTTOM
MPI_PACKED
MPI_DATATYPE_NULL

More Communicators (1)

So far, we have described only intra–communicators
Communication within a group of processes

You can also define inter–communicators
Communication between two groups of processes
Almost nobody seems to want/need to do this

Relevant functions:

MPI_Comm_remote_group
MPI_Comm_remote_size
MPI_Comm_test_inter
MPI_Intercomm_create
MPI_Intercomm_merge

More Communicators (2)

It's dubious which of the following is trickier:
Inter–communicators or overlapping communicators

MPI supports both of them, especially MPI–2
And even the combination, for masochists!

Almost everything is clearly and precisely defined

- Thinking about using either makes my head hurt

If you really must use either facility
study the MPI standard, carefully
And you are on your own trying to tune it!

Topologies (1)

Communicators may have virtual topologies
Used to map program's structure to cluster's

- You describe the program's structure

Library may optimise CPU allocation for it
Used to be important, now is very esoteric

- Almost totally useless on switched networks

Most others now use high–connectivity topologies
See Parallel Programming: Options and Design

Or can use explicit CPU allocation (outside MPI)

Topologies (2)

Relevant functions and constants:

MPI_Cart_coords	MPI_Dims_create
MPI_Cart_create	MPI_Graph_create
MPI_Cart_get	MPI_Graph_get
MPI_Cart_map	MPI_Graph_map
MPI_Cart_rank	MPI_Graph_neighbors
MPI_Cart_shift	MPI_Graph_neighbors_count
MPI_Cart_sub	MPI_Graphdims_get
MPI_Cartdim_get	MPI_Topo_test
MPI_GRAPH	MPI_CART

Datatype Conversion

MPI will convert data from one type to another
Essentially, when it can always be got ‘‘right’’

- I strongly advise not using this facility

Do the conversion yourself, checking for errors
Can do it either beforehand or afterwards

If you want MPI to do it, read the standard
You need to know the precise restrictions

Heterogeneous Clusters

This is where not all systems are similar
As mentioned, MPI has facilities to support them

- They are an absolute nightmare to use

Don't be taken in by the availability of facilities

- The problem is primarily semantic differences

Most systems use the same hardware formats
See ''How Computers Handle Numbers'' for more

Data packing resolves most compiler differences
Assuming a common interchange format, of course

MPI-2 Facilities

We have already used some of them in the course
I haven't looked at most of them in any detail

And how many implementations support them?
The simpler ones are probably available, and work

Investigation would be needed for the complex ones
I know that some implementations ''support'' them

- Please ask for help if you need the features

I can enquire from higher–level experts if needed

Miscellany (1)

Quite a few minor extensions and similar
Will mention ones most likely to be useful

Some already described (e.g. MPI_Finalized)
Won't repeat the ones that have been

Features for supporting other parts of MPI–2
No point in describing them separately

Miscellany (2)

Can call an error handler from user code
Enables cleaner error handling in some programs

Can set callbacks for MPI_Finalize
Useful for cleaning up in library code

Can pass null arguments to MPI_Init
Probably useful only for library code

Name Changes (1)

Changes to some names, deprecating the old ones
The old ones still work, except for C++
Some error handling, almost all attribute caching,
and most derived datatype ones

Error handling name changes:

MPI_Errhandler_create $\Rightarrow$ MPI_Comm_create_errhandler
MPI_Errhandler_get $\Rightarrow$ MPI_Comm_get_errhandler
MPI_Errhandler_set $\Rightarrow$ MPI_Comm_set_errhandler
MPI_Handler_function $\Rightarrow$ MPI_Comm_errhandler_fn

Name Changes (3)

Some attribute caching name changes:

MPI_Attr_delete	$\Rightarrow$	MPI_Comm_delete_attr
MPI_Attr_get	$\Rightarrow$	MPI_Comm_get_attr
MPI_Attr_put	$\Rightarrow$	MPI_Comm_set_attr
MPI_Copy_function	$\Rightarrow$	MPI_Comm_copy_attr_function
MPI_Delete_function	$\Rightarrow$	MPI_Comm_delete_attr_functio
MPI_Dup_fn	$\Rightarrow$	MPI_Comm_dup_fn

Name Changes (4)

More attribute caching name changes:

MPI_Keyval_create	$\Rightarrow$	MPI_Comm_create_keyval
MPI_Keyval_free	$\Rightarrow$	MPI_Comm_free_keyval
MPI_Null_copy_fn	$\Rightarrow$	MPI_Comm_null_copy_fn
MPI_Null_delete_fn	$\Rightarrow$	MPI_Comm_null_delete_fn
COPY_FUNCTION	$\Rightarrow$	COMM_COPY_ATTR_FN
DELETE_FUNCTION	$\Rightarrow$	COMM_DELETE_ATTR_FN

Name Changes (5)

Derived type name changes:

MPI_Address	$\Rightarrow$	MPI_Get_address
MPI_Type_hindexed	$\Rightarrow$	MPI_Type_create_hindexed
MPI_Type_hvector	$\Rightarrow$	MPI_Type_create_hvector
MPI_Type_struct	$\Rightarrow$	MPI_Type_create_struct
MPI_Type_extent	$\Rightarrow$	MPI_Type_get_extent
MPI_Type_lb	$\Rightarrow$	MPI_Type_get_extent
MPI_Type_ub	$\Rightarrow$	MPI_Type_get_extent
MPI_LB	$\Rightarrow$	MPI_Type_create_resized
MPI_UB	$\Rightarrow$	MPI_Type_create_resized

MPI_Status Enhancements

Can ask for status not to be returned

- Do that only when it is definitely irrelevant

Generally, use it only for wait after send

Pseudo–pointer MPI_STATUS_IGNORE

or array version MPI_STATUSES_IGNORE

In C++, just omit the status argument

- Can inspect status without freeing request

Very important when writing MPI libraries

Use the function MPI_Request_get_status

Memory Allocation

Can provide callbacks for memory allocation
Primarily provided for RDMA support
But could well have other uses

- It is intrinsically implementation–dependent

Implementations may call it in different ways

Procedures MPI_Alloc_mem and MPI_Free_mem

Language Bindings etc.

MPI–2 includes direct C++ support
It includes a mpi module for Fortran 90

We have already used both of those in this course

Some features for language interoperability
E.g. C++ and Fortran, both calling MPI
Important for anyone writing MPI libraries

- Otherwise, I recommend not going there

Call MPI from only one language – it's easier

External Interfaces (1)

What the MPI–2 standard calls the section!

Mechanisms to improve MPI's diagnostics
Potentially useful, but not a major improvement

- Worth looking at, as quite simple to use

Portable thread support within MPI

- My recommendation is don't go there

Have already given recommendations on what to do

External Interfaces (2)

Extended attribute caching facilities
Potentially useful, especially for library writers

Includes ways for an application to extend MPI
Potentially very useful, but definitely advanced

Recommendation:

> If you need functionality MPI–1 doesn't have
> Check extensions before writing your own

Extended Collectives

A generalised all–to–all (MPI_Alltoallw)

Put an icepack on your head before using it

It could have its uses, but is very complicated

These can be used on inter–communicators

Important for the support of process creation

But I recommend not even thinking of doing that!

I/O (1)

Genuinely parallel I/O to a single file

Much better than most ‘‘parallel I/O’’ interfaces
But definitely and unavoidably complicated

- Don’t go there unless you really need to

That applies to all forms of parallel I/O

But, if you need to, you really need to

- Ask for help if you think you may

I/O (2)

When might you need to?

- Your application is severely limited by I/O
- Serious tuning of serial I/O has failed
- Spreading I/O across multiple files has, too

Then you need to change to using parallel I/O
MPI–2 parallel I/O is well worth considering

I don't think that any Cambridge users need it
But please tell me if I am wrong!

Canonical Data Representation

For the support of heterogeneous clusters
I.e. ones with different data representations

Enhancements to MPI_Pack and MPI_Unpack
a new data representation format ‘‘external32’’

- I recommend not going there unless you have to

Process Creation etc.

You can add groups of processes dynamically
MPI–2 is probably the best way to do this

- My recommendation is don't even think of it

This was a nightmare area in PVM
The potential system problems are unbelievable

And that is even if you are your own administrator
If you aren't, you may get strangled for using this

MPI 3.0

Currently being planned and developed
Not even a draft specification available yet
Probable extensions include:

- Non–blocking collectives
- Improved Fortran 90 support

Watch this space, but don’t hold your breath
Interested people should talk to me offline

Finished!

And that’s mentioned every major feature in MPI