Writing Message-Passing Parallel Programs with MPI

A two-day course

Trainers Notes

Elspeth Minty

Edinburgh Parallel Computing Centre

University of Edinburgh

Introduction

These trainers notes are intended to be used in conjunction with the *Writing Message-Passing Parallel Programs with MPI* course notes and slides. Their aim is to provide an overview of how the course is taught within EPCC, and suggestions on how it could be adapted in different circumstances.

It is not intended that the notes be used as speakers notes during the course sessions. Rather they are aimed at providing a list of key points which can be used in conjunction with the slides to construct speakers notes. We have tried to ensure that the notes on the sessions are as complete as possible. One side effect of this is that many trainers may find them rather too complete.

While these notes contain pointers to relevant passages of the MPI standard, they are not intended as an overview of MPI and it is expected that those using the trainers notes will have a thorough understanding of MPI. Trainers should also ensure that they have completed and understood the course exercises and case study, both the C and Fortran versions. It may also be worthwhile spending some time going through the extensions to the exercises. Example solutions for the exercises and the case studies are included in the course package. It should be emphasised that these are *example* solutions and there may be a number of other equally valid solutions.

Student Prerequisites

The course itself assumes that students have a good working knowledge of UNIX and C/Fortran. It does *not* assume that the students have any prior knowledge of the concepts of message passing.

Course Structure

The course as its presented here is a two day course. Possible adaptions for the course are discussed below. The basic format of the sessions is a short talk and then a practical using the material covered in the lectures. The emphasis is on giving students as much “hands on” experience of using MPI as possible. In addition the practicals are designed so that each builds on the previous as much as possible.

The only main session which doesn’t follow this format is the *Getting Started* session which aims to give students an introduction to the general principles behind message passing, and to discuss the reasons for the development of MPI and the principles behind it.

The following is an example time table for the course:

Day 1	Talk	Practical
Course Introduction	09:00 - 09:15	
Getting Started	09:15 - 09:45	

Day 1	Talk	Practical
MPI Programs	09:45 - 10:15	10:15 - 10:45
Coffee	10:45 - 11:15	
What' s in a Message	11:15 - 11:30	
Point-to-Point Communication	11:30 - 12:00	12:00 - 13:00
Lunch	13:00 - 14:00	
Non-Blocking Communication	14:00 - 14:30	14:30 - 15:30
Coffee	15:30 - 16:00	
Derived Datatypes	16:00 - 16:20	16:20 - 17:00

Day 2	Talk	Practical
Virtual T opologies	09:00 - 09:30	09:30 - 11:00
Coffee	11:00 - 11:30	
Collective Communications	11:30 - 12:00	12:00 - 13:00
Lunch	13:00 - 14:00	
Case Study	14:00 - 14:30	14:30 - 17:00

The timetable shown above is one which is typically used for classes of mixed experience. Suggestions for other ways of timetabling the course are discussed in "Variations of the course" on page 8.

Relative to the talks, the practical sessions are fairly long. This is to give students who have no message passing background sufficient time to finish the practicals. As it is likely that students who have used message passing previously will finish the practicals in less time than is scheduled, extra exercises have been added which are intended to give experienced users a deeper insight into that area of MPI.

Variations of the course

The timetable given in "Course Structure" on page 6 is for a two day course for a mixed experience group. Depending on the students needs and abilities it may be useful to vary from the timetable shown. The later modules of the course (those following the session on Non-Blocking Communication) have been written in such a way as to give course organisers the option of reordering the modules to suit their needs without affecting the sense of the course.

One possible variation on the basic course is a three day course which gives more of a general introduction to parallel computing, and the machines/tools used in the course. For students who have no background in parallel computing, this course structure would give them more time to absorb the concepts and familiarise themselves with the machines and tools which they will be using for the remainder of the course. An example timetable for the initial day of such a course is shown below. It is based on a similar course run within EPCC.

Day 0	Talk	Practical
Course Introduction	09:00 - 09:15	
Introduction to Parallel Computers	09:15 - 10:00	
Decomposition techniques	10:00 - 11:00	
Coffee	11:00 - 11:30	
Parallel Computing Concepts	11:30 - 13:00	
Lunch	13:00 - 14:00	
Machine Overview	14:00 - 15:00	

Day 0	Talk	Practical
Coffee	15:00 - 15:30	
Tools	15:30 - 17:00	15:30 - 17:0.0

Suggestions for the contents of the above talks include:

Course Introduction: This session gives the trainers an opportunity to find out more about the course participants. It is described in more detail in "Course Introduction" on page 15.

Introduction to Parallel Computing: The main idea of this talk is to give the students some idea of the development of parallel computing, Amdahl's Law, and of the types of architectures which are available.

Decomposing the Potentially Parallel: This is envisaged as a general discussion of the different methods of parallelising code such as task farms, domain decomposition, etc., and their advantages/disadvantages.

Parallel Computing Paradigms: The aim of this session is to give an overview of message passing and data parallel programming. While the course sets out to teach message passing with MPI, a discussion of data parallel computing would give the students a wider knowledge of parallel computing in general. The session described in "Getting Started" on page 17 could easily be adapted to an overview of message passing concepts.

Machine Overview: This talk should familiarise the student with the machine which they will be using for the course. If the course is being run on a network of workstations rather than a parallel machine, this would provide a valuable opportunity to show the students that workstations can be regarded as parallel resources.

Tools: This session should cover how to compile and run code on the machine used for the course, and the use of any debuggers and profilers available. It should include a practical which could involve finding and fixing a simple bug in a single process code, and profiling the resulting code. One example of such an exercise which is used within EPCC is that of an uninitialised variable which leads to a divide by zero. While specific to one platform, many debuggers and profilers work in a similar way and it would be hoped that students could readily adapt what they have learned in this session to other platforms.

The remainder of the course described in “Course Structure” on page 6 could be adapted to fill the remaining two days as follows

Day 1	Talk	Practical
MPI Programs	09:00 - 09:30	09:30 - 10:00
What’ s in a Message	10:00 - 10:15	
Coffee	10:15 - 10:45	
Point-to-Point Communication	10:45 - 11:15	11:15 - 12:30
Lunch	12:30 - 13:30	
Non-Blocking Communication	13:30 - 14:00	14:00 - 15:15

Day 1	Talk	Practical
Coffee	15:15 - 15:45	
Derived Datatypes	15:45 - 16:05	16:05 - 17:00

Day 2	Talk	Practical
Virtual T opologies	09:00 - 09:30	09:30 - 11:00
Coffee	11:00 - 11:30	
Collective Communications	11:30 - 12:00	12:00 - 13:00
Lunch	13:00 - 14:00	
Case Study	14:00 - 14:30	14:30 - 17:00

The time allocated to the practical sessions on Day 1 has been increased. This is to ensure that students, who have little or no background in parallel computing, have sufficient time to absorb the basic ideas before moving onto the advanced topics of the second day.

The pace of this course is deliberately slower to allow the students time to fully absorb the concepts of parallel computing, and to explore the exercises in greater detail than would be possible in the timetable described in “Course Structure” on page 6.

Another useful variation of the timetable, is to reduce it to a one day course for students who have a strong background in message passing and simply want to learn about MPI. Here a timetable could be as follows:

	Talk	Practical
Course Introduction	09:00 - 09:15	

	Talk	Practical
MPI Programs	09:15 - 09:40	09:40 - 10:00
What' s in a Message	10:00 - 10:10	
Point-to-Point Communication	10:10 - 10:30	10:30 - 11:00
Coffee	11:00 - 11:30	
Non-Blocking Communication	11:30 - 11:50	11:50 - 12:30
Lunch	12:30 - 13:30	
Derived Datatypes	13:30 - 13:45	13:45 - 14:15
Virtual T opologies	14:15 - 14:45	14:45 - 15:30
Coffee	15:30 - 16:00	
Collective Communications	16:00 - 16:30	16:30 - 17:00

In addition, the case study could be given out as a stand-alone exercise. The sessions on the above course are fairly short and only students with a strong message passing background would manage to complete all the practical sessions. If it is felt that the practicals are too short, then either the Derived Datatypes or Virtual Topologies session could be given out as a handout.

Before running the course

Before running the course for the first time, you should check that your MPI implementation of MPI is properly installed and that the examples, exercises and case study all run correctly on the course machine in both C and Fortran.

The *tar* files for the case study should be placed in the directory given in the notes, or the notes altered accordingly.

If you are not using the EPCC implementation of MPI, you should ensure that Appendix A of the course notes contains the correct information on how to compile and run MPI code with your implementation.

Finally, Appendix B of the course notes contains information on how to use the machine we run the course on in EPCC. This is intended as a guide to the type of information which students find useful, and should be adapted to the platform which you will be running the course on.

Student Documentation

Each student on the course is given the following information:

1. *Course notes*. These are a set of notes which accompany the sessions in the course. They are intended to be complementary to the talks rather than following them directly.

2. *Slides*. The slides are given out so that the students can make additional notes during the talks if they wish. In EPCC, we give them out with two slides on one side on A4 paper.

3. *MPI Standard*. The course notes are not intended as a complete guide to MPI, and the students are encouraged during the course to make use of the MPI standard.

4. *MPI FAQ*. A copy of the MPI Frequently Asked Questions is also given out.

Structure of trainers Notes

The remainder of the trainers notes consists of descriptions of the individual sessions which make up the course. These give the aims of the sessions and exercises, and the key points to be brought out in the talks. The key points are not intended as an exhaustive list of what should be said in each session, but a guide to what we feel is important. Where appropriate references to the relevant sections of the course notes and MPI standard are included.

Course Introduction

Aims of Session

This session is to enable those teaching the course to introduce themselves to the students, go over the structure of the course, explain where to get coffee and lunch, etc.

It's also useful to get an idea of what the backgrounds and interests of the participants are. This should help identify inexperienced students who may have problems with the course, as well as those who are likely to get through the material with little difficulty.

Hints on Teaching

After introducing yourself and any other trainers, go round the class and ask everyone to say a few words about who they are and what they are looking to get out of the course. In addition to providing potentially useful information to trainers, this also helps break the ice between the students.

Getting Started

Aims of Session

This session is really an introduction to message passing concepts with some information on the background to MPI included. It is aimed at students who have little or no background in parallel computing.

Key Points

1. The serial programming paradigm is based around a single process with some memory attached to them. The message passing paradigm is simply an extension to this model where we have a number of processes each with some memory attached to them.

2. The processes all work independently on their own subprogram. In order to get data which is stored in another processes memory, a process must send data which the receiving process can understand as a request for information. The receiving process can then send back the data which the other process wants.

3. Similarity between message passing system and everyday items such as mail box, phone line, and fax machine. Try to tie in message passing with familiar everyday examples.

4. Synchronous/asynchronous relates to the completion of the communication, while blocking/non-blocking relates to the communication call itself.

5. It's also worth noting that a number of message passing routines do not behave like subroutine calls. Non-blocking communications is an example of this as there is still work going on in the background after the non-blocking call returns.

6. As well as simplifying code, collective communications can be identified by the compiler and, if possible, implemented in a more efficient manner.

7. MPI aims to be efficient on different platforms as well as portable. Also supports heterogeneous systems.

Hints on Teaching

1. An example of a barrier is waiting to start teaching until the last student arrives. Everyone gets to the same point (seated in the classroom) but the teaching does not begin until the last person has arrived and is seated.

2. Similarly, a broadcast can be compared to you, as trainer, talking to the class. You are broadcasting information, and the students are receiving it.

References

This material is covered in "Getting Started" on page 7 of the course notes. There is a discussion of the development of MPI in Chapter 1 of the specification.

NB: All references to the MPI specification refer to the May 5, 1994 version.

MPI Programs

Aims of Session

This session describes how to use MPI in terms of the header files required and how to initialise/exit MPI. The concept of a communication is introduced, and a number of simple routines to query MPI_COMM_WORLD are described.

As this session includes the first exercise, this is an appropriate point to cover the details of how to use the machine and MPI implementation used in the course.

Key Points

1. When describing the format of MPI routines it should be emphasised that the course notes only contain the MPI interface for the routines, and not the C and Fortran versions. The reason for this is to encourage students to become familiar with using the MPI specification rather than relying on the notes which only include a relatively small number of the MPI routines.

2. The handles which MPI uses to control operations can be queried to find out information about either the status or result of the operation.

3. When MPI_INIT is called the initial communication world, MPI_COMM_WORLD, is set up, and each process is given a rank in the communication world. The ranks range from zero to one minus the number of processes. It is not possible for more processes to join MPI_COMM_WORLD later.

4. MPI cannot be re-initialised after MPI_FINALIZE has been called.

5. Explain the machine and MPI implementation being used on the course.

Hints on Teaching

This is the first time the students login to the machine used in the course, so its worth taking them through it one step at a time.

References

The topics discussed in this session are discussed in "MPI Programs" on page 17 of the course notes. The relevant sections of MPI specification are 2.1 - 2.8, 7.5, and 5.4.1

Exercises

This exercise is aimed at ensuring that the students fully understand that they have multiple version of versions of a program running on different processes. The main points are as follows:

1. When the "hello world" code is run on several processes, each prints out the message.

2. Each process has a unique rank in the range 0 to one less than the total number of processes.

3. Using this information it is possible to limit the number of processes which print the message out.

What's in a Message ?

Aims of Session

This is a short session which describes the MPI datatypes. Its aim is to get across the ideas of what an MPI datatype is and why they are used.

Key Points

1. Messages are consist of MPI datatypes.

2. Allows messages to be sent between heterogeneous platforms without any conversion problems

3. Each basic C and Fortran type has an equivalent MPI datatype.

4. Can build more complex datatypes based on the basic datatypes.

5. Send and receive messages of the same datatype.

Hints on Teaching

Its worth mentioning MPI_BYTE and MPI_PACKED but they are only included for completness.

References

This material is covered in "What's in a Message?" on page 23 of the course notes, and Sections 3.2.1 and 3.13 of the MPI standard.

Point-to-Point Communication

Aims of Session

This is one of the key sessions in the course. It provides information on how to perform point-to-point communication in MPI. The session recaps on the relevant parts of the Getting Started session but with specific reference to MPI. All the blocking MPI communication modes are covered.

The practical at the end of the session gives students an opportunity to write a simple MPI program to carry out point-to-point communications.

Key Points

1. Point to point communication is when one process sends a message to one other process.

2. Both processes taking part in the communication must be in the same communicator.

3. There are a variety of modes in which a message can be sent but only one receive mode.

4. The example send given in the slides (Slide 43: Sending a message) is a synchronous send. This is the safest way to send a message.

5. The information given to the send routine can be divided into two: a description of the message (where it is, how long, what type) and details on where to send the message.

6. The receiving process can use wildcards for the source and tag. The actual source and tag are returned in the status parameter.

7. All the information about the communication is contained in the *communication envelope*. This can be queried by the program to find information such as the size and source of a message.

8. If a process sends two messages to another process, then they are guaranteed to arrive in the order they left, *i.e.*, messages do not overtake each other. However, you cannot say anything about the ordering of messages from different processes.

Hints on Teaching

1. The analogy between the information sent by the communication network and a message in an envelope is a useful one. The people delivering the message don't care what is in the envelope, the envelope is delivered to the person who it is addressed to and can have a return address on it. The person receiving the envelope can use information on its contents to decide whether or not to open it.

2. The necessity to ensure that the receivers buffer is large enough can be illustrated using two different sized manuals and their boxes. Each manual fits exactly into its own box, and the smaller manual will fit into the larger box. However the larger manual will not fit into the smaller box.

References

This session is covered in "Point-to-Point Communication" on page 25 of the course notes. The relevant sections of the MPI standard are 3.1-3.5, and 7.4

Exercises

The aim of this exercise is to get students to write an MPI program which passes a number from one process to another. They have to ensure that one process is sending the message and another is receiving it, or deadlock will occur. It is intended to illustrate the following points.

1. While the two processes are executing exactly the same program, they can be made to do different things (in this case, sending and receiving) depending on their rank.

2. If one process executes a blocking send, another processes must post a receive or the send will never complete.

This is a good time to introduce the MPI timing routines. This will enable more advanced students to time how long the communications take for different message sizes and so make the exercise more interesting to them. For the less experienced students it is yet another example that the processes work independently of each other, and that the timers on two different processes are unlikely to be synchronised.

Non-Blocking Communication

Aims of Session

This session starts from the problem of processes deadlocking and introduces non-blocking communications. The practical used in this session is the basis for the remaining practicals in the course.

Key Points

1. Deadlock is a common problem. It can be avoided by, for example, having odd numbered processes send and even numbered processes receive, but a much neater solution is to use non-blocking communications.

2. The program must not update the variable(s) involved in the communication until it has tested that the non-blocking communication has completed.

3. MPI non-blocking routines are similar to their blocking equivalents. The difference is the MPI handle which is returned by the call. It is this handle which is queried by the program to find out if the communication has completed.

4. The program *must* test the handle to ensure that the communication has completed. The communication system may allocate resources relating to the communication which will not be freed until the communication has both finished and been tested.

5. It is normally not necessary to have both a non-blocking send and a non-blocking receive.

6. A program can either test for completion or wait until the program has completed. The test routine will return immediately with the flag set to either TRUE or FALSE depending on whether or not the communication has completed, while the wait will block until the communication has completed.

7. The choice between testing for completion or waiting depends on whether or not you have work to do which does not require the data involved in the communication.

8. A non-blocking call followed immediately by a wait is identical to the blocking equivalent of the call.

Hints on Teaching

1. A good way to start off this session is with a demonstration of deadlock. If you get, say, three students to come to the front of the class and stand them in a circle as shown in Figure 1.

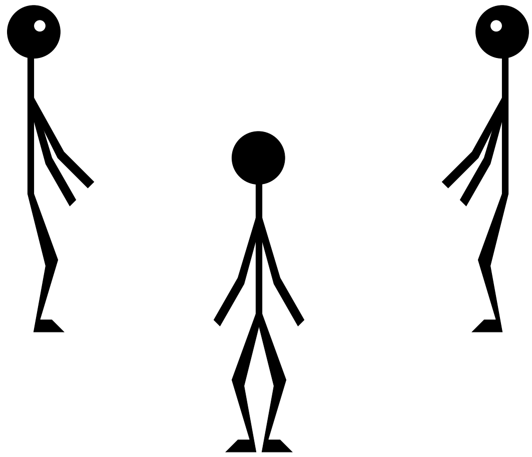

Figure 1: Students stand in a circle

The three students are then numbered 0, 1, and 2, given an envelope with their right neighbours number on it, and told to put their other left hand behind their back. They then have to simulate sending a message to their neighbour by passing the envelope as in Figure 2.

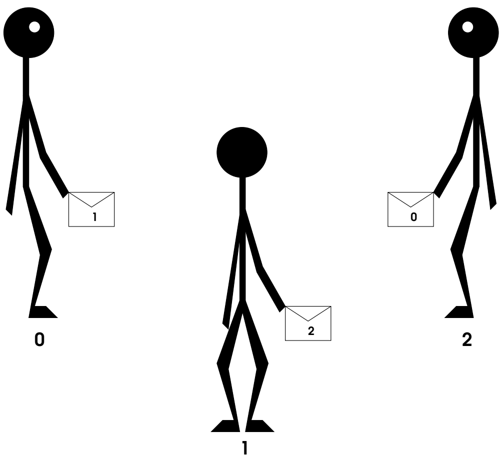

Figure 2: Trying to pass the envelope to your neighbour

Everyone is trying to pass their envelope to their right neighbour and so cannot receive an envelope from their left neighbour. This is a deadlock situation. The solution is shown in Figure 3

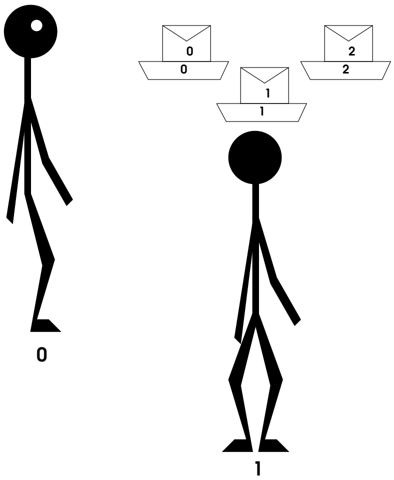

Figure 3: Use of buffers to resolve deadlock

Each student is assigned a buffer. The envelopes are sent by putting the envelope into the buffer with the correct number on it. The students receive the messages by picking the envelopes out of the buffers and then check that the envelope they sent has been received.

We have found this demonstration of deadlock extremely useful in the EPCC courses, as it provides an obvious, and understandable, example of both the problem and the solution.

References

This session is covered in “Non-Blocking Communication” on page 35 of the course notes, and in sections 3.7 of the MPI standard

Exercises

This exercise forms the basis of the rest of the practical sessions. Its basically a global sum carried out by all the processes passing ranks round a ring until they get their own rank back. They then print out the result which should be the sum of all the processes ranks. For example, for three processes the stages would be as shown in Figure 4.

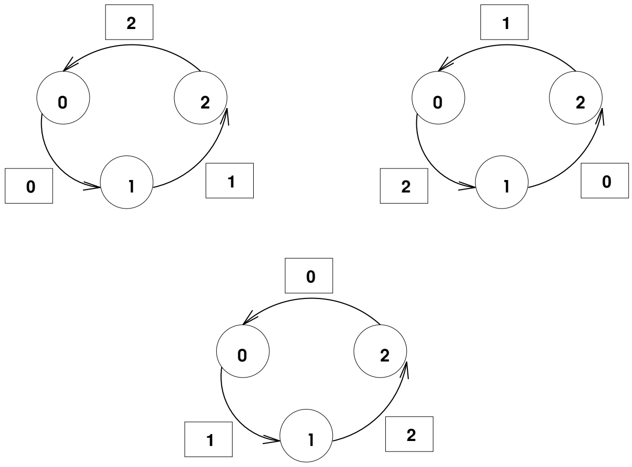

Figure 4: Rotating information around a ring.

Each process initially sets its message to be its rank. They then pass this onto the process on their right, and receive from the process on their left. They add the number they received to their local sum and pass the number onto the neighbour on the right. This continues until they receive their rank back again.

This exercise illustrates the use of non-blocking communications where this type of regular communication pattern exists.

Introduction to Derived Datatypes

Aims of Session

So far all the communications have involved sending messages of the basic MPI datatype. This session shows how to build up more complicated datatypes.

Key Points

1. Derived datatypes are datatypes which are built from the basic MPI datatypes.

2. Derived datatypes provide a template of the data which is to be sent. All the data in the datatype is identified by its offset from a base address. The base address is the address which is passed to the MPI routine using the derived datatype. This allows the same MPI datatype to be used for any number of variables of the same form.

3. MPI provides a number of different routines for creating derived datatypes, each aimed at certain types of data, *i.e.,* contiguous data, non-contiguous data, and non-contiguous mixed data.

4. Every derived datatype must be committed before it can be used.

5. The MPI routine MPI_TYPE_EXTENT is useful for calculating displacements as it takes into account any alignment issues.

6. Derived datatypes are an efficient method of sending data as the data is moved from its location in one processes memory to a location in a different processes memory without any intermediate buffering. However, as mentioned in the previous session, this means the program cannot update the data until the communication has completed.

7. The use of MPI_BOTTOM in Fortran programs is still an unresolved issue in MPI. See p6 of the Errata Sheet for MPI Specification (July 12, 1994) for more details.

Hints on Teaching

Constructing derived datatypes is mainly a case of going through each of the arguments to the routine you want to use and putting in the appropriate values. From our experiences within EPCC, most students go into this session thinking derived datatypes are very complicated but find the practical session fairly straightforward.

References

This session is covered in “Introduction to Derived Datatypes” on page 43 of the course notes, and in section 3.12 of the student notes.

Exercises

This exercise expands on the previous sessions exercise. Instead of passing the rank round as one integer, it should be passed as a floating point number as well. Both the integer and floating point sum should be printed.

The aim of this exercise is simply to illustrate the use of derived datatypes.

Convenient Process Naming: Virtual Topologies

Aims of Session

This session covers the basics of how to define process topologies within MPI. It concentrates on cartesian topologies but mentions that it is possible to create more general process topologies. As well as looking at virtual topologies this is the only session of the course which discusses creating new communicators.

Key Points

1. As well as providing a convenient way to name processes, virtual topologies potentially allow MPI to rearrange process ranks to fit in better with the required topology and so to optimise communications.

2. While the topology naming scheme provides a logical way of naming processes, messages must still be sent to process ranks. MPI provides a number of routines which map between the topology naming scheme and process ranks.

3. When creating a virtual topology, all processes in a communicator must call the MPI_CART_CREATE routine at the same time.

4. When creating a virtual topology, allowing MPI to reorder the process ranks may have performance benefits.

5. The MPI_CART_SHIFT function returns the rank of the neighbouring processes in the direction and at the displacement specified. It does not move data about. This is different from Data Parallel shifts which do move data about.

6. The cartesian partitioning routine returns one communicator to each process *not* an array of communicators.

7. Fortran programmers may find the numbering of the dimensions of the cartesian topology confusing. The array holding the dimensions would typically start from 1 but the dimensions are numbered from 0. A note is included on this in "Creating a cartesian virtual topology" on page 56 of the student notes.

Hints on Teaching

The idea of virtual topologies will be new to most students, even those with previous experience of message passing. A useful way to approach it is to relate it back to ring exercise and discuss it in terms of the problems in wrapping round the process ranks in the communications.

References

This session is covered in "Convenient Process Naming: Virtual Topologies" on page 55 of the course notes and chapter 6 of the MPI standard.

Exercises

This exercise is again based on passing messages round a ring of processes. It shows how virtual topologies can be used to simplify communications. The exercise also provides students with the opportunity to create a communicator other than MPI_COMM_WORLD. One thing to watch out for in the exercise is that students will continue to query the old communicator, MPI_COMM_WORLD, to find out information such as process ranks, rather than the communicator associated with a virtual topology.

Students should be encouraged to use cartesian partitioning in the second part of the exercise to illustrate the idea of communicators as separate communications worlds. Processes in different dimensions may be in different communicators but the all see a communication world with processes ranked from zero to one less than the number of processes in the communicator.

Collective Communications

Aims of Session

MPI provides a wide range of collective communications which it is impossible to cover in detail in the time available. This session aims at providing an overview of the different types of operations which are possible rather than the different versions of individual operations. The basic procedure for creating a user defined reduction operation is included but is aimed to be a pointer to how to do it rather than providing detailed information.

Key Points

1. In MPI, all processes in a communicator take part in collective communications.

2. Their use can have performance benefits as many platforms have special hardware features for doing operations such as barriers, broadcasts, etc.

3. In the scatter operation, all of the processes need to specify the buffer from which the data is being scattered, but its contents are only meaningful on the root process.

4. Global reduction operators combine data from across the processes into one result.

5. With MPI_REDUCE the result is only meaningful on the root process. MPI_ALLREDUCE will put the result on all the processes.

6. The contents of the result buffer on processes other than the root may be altered but its contents are not defined.

7. As well as the pre-defined reduction operators, users can define their own.

8. Scan operations are likely to be familiar to students with a Data Parallel background.

Hints on Teaching

There are a number of simple analogies which can be made between collective communications and more familiar everyday events.

1. A barrier is similar to a situation where an event cannot happen until everyone in a group has arrived. An example of this could be this talk itself - it did not start until the whole class had assembled in the classroom.

2. A broadcast is a one-to-all communication. This can be compared to the talk itself where you, as trainer, are broadcasting to the students who are receiving the information.

3. Again to use the course as an example, passing out the notes is a scattering operation, and if you were to take them back in again this would be a gather operation.

References

This session is covered in "Collective Communication" on page 61of the course notes, and in chapter 4 of the MPI standard.

Exercises

This is the final exercise in the series based on performing a global sum on processes ranks by passing them round a ring of processes. The first part of this exercise shows how much simpler it is to use collective communications than to use point-to-point communications to do the same task. The second part illustrates the use of the scan operation while the final section uses barriers to ensure that the processes print their results in order. It may be that the processes print their results in order without barriers but students should be encouraged to use them to *ensure* the order is correct.

MPI Case Study

Aims

The case study integrates the major MPI concepts learned during the course into one realistic application; it is designed to reinforce the student new knowledge of MPI. The case study also provides an overview of the issues faced when writing a realistic message passing program using MPI.

The case study models populations of foxes and rabbits as they evolve and interact over the years. Formulating the problem in terms of foxes and rabbits minimises the time spent explaining the underlying mathematical model. The number of foxes and rabbits evolves over time as they both breed, the foxes eat the rabbits or starve due to overfeeding, and die naturally. The model used by the case study is described in “MPI Case Study” on page 71 of the course notes.

As the aim of the case study is to provide a realistic example of parallelising a program, the students are provided with serial code as a starting point. Two sequential codes are provided: a plain sequential code and one augmented with comments and data-structure definitions. We encourage the students to start from the augmented code which lets them concentrate on MPI rather than Fortran or C programming.

Key Points

The case study is an opportunity to review the following MPI features:

1. Cartesian Topologies (1-D and 2-D)

2. Derived Data Types (Strided Vectors)

3. Point-to-Point Communications (Data Shifts)

4. Collective Communications (Global Sums)

Students are also asked work out incrementally a two-dimensional geometric decomposition of the problem.

Hints

Some students may have some difficulties seeing how a geometric data decomposition can be programmed. The concept of halo data and the difference between local and global array indices need careful explanation.

The augmented sequential code strongly suggests a mapping of the cartesian topology of processes to the dimensions of the arrays Fox and Rabbit that store population counts. This mapping underpins to the geometric data decomposition written by the student. Different mappings will seem more natural to different people; it is probably best if demonstrators have a chance to look at the sample solution before the actual practical.

References

This material is covered in “MPI Case Study” on page 71.