FORTRAN and MPI

Message Passing Interface (MPI)

Day 1

Course plan:

- MPI - General concepts
- Communications in MPI
  - Point-to-point communications
  - Collective communications
- Parallel debugging
- Advanced MPI: user-defined data types, functions
  - Linear Algebra operations
- Advanced MPI: communicators, virtual topologies
  - Parallel sort algorithms
- Parallel performance. Summary. Tendencies
Message passing computer architecture paradigm

PE - processing elements
M - memories

Message passing programming paradigm

P - processes
Message passing programming paradigm (cont.)

- A single program is run on each processor.
- All variables are private.
- Processes communicate via special subroutine calls - MPI is just a library.
- There is no 'magic' parallelism.
- The program is written in a conventional sequential language, i.e. C or Fortran.

Messages

- Messages are packets of data moving between processes.
- The message passing system has to be told the following information:
  - Sending process
  - Source location
  - Data type
  - Data length
  - Receiving process(es)
  - Destination location
  - Size of receive buffer(s)
A message passing system is similar to a mailbox, phone line or a fax machine.

A process needs to be connected to a message passing interface. Thus,

- The sender must have addresses to send the message to
- Receiving process must:
  - participate (cf. have a mailbox it checks, a phone it answers, ...)
  - have capacity to receive (have a big enough mailbox etc)

Point-to-Point Communication

- Simplest form of message passing.
- One process sends a message to another
- Both ends must actively participate
- Sending a point-to-point message requires specifying all the details of the message
Point-to-Point Communication types

- Synchronous vs asynchronous
- Blocking vs non-blocking
- Buffer space, reliability, ...

Leads to a myriad of different types of point-to-point communication calls.

Collective communications

- Collective communication routines – higher level routines involving several processes at a time (often all).
- Can be built out of point-to-point communications.
MPI: a standard Message Passing Interface

- Defined by MPI Forum – 40 vendor and academic/user organizations
- Provides source code portability across all systems
- Allows efficient implementation.
- Provides high level functionality.
- Supports heterogeneous parallel architectures.
- An addition to MPI-1 – MPI-2.

The main features of MPI:

(i) a set of routines that support point–to–point communication between pairs of processors in blocking and nonblocking versions;

(ii) a communicator abstraction that provides support for the design of modular parallel software libraries;

(iii) application topologies specifying the logical layout of the processes;

(iv) a rich set of collective communication routines performing coordinated communications among a set of processes.
Basic MPI concepts and routines

MPI semantic terms

- An MPI program consists of autonomous processes executing their own C or Fortran code, in a MIMD (SPMD) style.

- The code executed by each process need not be identical.

- The processes communicate via call to MPI communication primitives.

Types of MPI calls

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local</td>
<td>if the completion of a procedure depends only on the local executing process (no communication required)</td>
</tr>
<tr>
<td>non-local</td>
<td>if the completion of the procedure may require the execution of some MPI procedure on another process</td>
</tr>
<tr>
<td>blocking</td>
<td>if return from the procedure indicates that the user is allowed to reuse the resources specified in the call</td>
</tr>
<tr>
<td>nonblocking</td>
<td>if the procedure may return before the initiated operation completes and before the user is allowed to reuse the resources (buffers)</td>
</tr>
<tr>
<td>collective</td>
<td>if all processes (in a group) need to invoke the procedure</td>
</tr>
</tbody>
</table>
MPI Programs

Header files

Should appear everywhere you call MPI procedures.

C: include <mpi.h>

Fortran: include ‘mpif.h’
### Fortran datatypes:

<table>
<thead>
<tr>
<th>MPI datatype</th>
<th>Fortran datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>MPI_REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>MPI_DOUBLE PRECISION</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>MPI_COMPLEX</td>
<td>COMPLEX</td>
</tr>
<tr>
<td>MPI_LOGICAL</td>
<td>LOGICAL</td>
</tr>
<tr>
<td>MPI_CHARACTER</td>
<td>CHARACTER(1)</td>
</tr>
<tr>
<td>MPI_BYTE</td>
<td></td>
</tr>
<tr>
<td>MPI_PACKED</td>
<td></td>
</tr>
</tbody>
</table>

### C datatypes

<table>
<thead>
<tr>
<th>MPI datatype</th>
<th>C datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_CHAR</td>
<td>signed char</td>
</tr>
<tr>
<td>MPI_SHORT</td>
<td>signed short int</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>signed int</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>signed long int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_SHORT</td>
<td>unsigned short int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_CHAR</td>
<td>unsigned char</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>MPI_LONG_DOUBLE</td>
<td>long double</td>
</tr>
<tr>
<td>MPI_UNSIGNED</td>
<td>unsigned int</td>
</tr>
<tr>
<td>MPI_UNSIGNED_LONG</td>
<td>unsigned long int</td>
</tr>
</tbody>
</table>
Initialising MPI

C:

int MPI_Init (int * argc, char ***argv)

--------------------------------------

Fortran:

subroutine MPI_Init(ierr)
integer ierr

Must be the first MPI procedure called.

Handles

- MPI controls its own internal data structures
- MPI exposes 'handles' to allow programmers to refer to these
- C handles are of defined typedefs
- Fortran handles are integers.
Communicators

- orthogonal message passing universes

- human analogy: the mail system is one communicator and the phone system another

- every message travels in a communicator (every message passing call has a communicator argument)

- more than just groups of processes – context

- very useful for libraries (library messages don’t interfere with library users messages)

---

MPI_COMM_WORLD communicator

- MPI_COMM_WORLD is the default communicator setup by MPI_Init()

- contains all processes

- for today, just use it wherever a communicator is required!

- MPI_COMM_WORLD is a handle (look in header file)
MPI Programs

RANK

How do we identify different processes?

C:
int MPI_Comm_rank(MPI_Comm comm, int *rank);

Fortran:
subroutine MPI_Comm_rank(comm, rank, ierror)
integer comm, rank, ierror

SIZE

How many processes are contained within a communicator?

C:
int MPI_Comm_size(MPI_Comm comm, int *size);

Fortran:
subroutine MPI_Comm_size(comm, size, ierror)
integer comm, size, ierror
Exiting MPI

Must be the last MPI procedure called by each process.

C:
   int MPI_Finalize();

Fortran:
   subroutine MPI_Finalize(ierr)
   integer ierr

To abort all processes of an MPI job:

C:
   int MPI_Abort(MPI_Comm comm, int errcode);

Fortran:
   subroutine MPI_Abort(comm, errcode, ierr)
   integer comm, errcode, ierr
The essence of MPI

The MPI standard contains many functions (≥ 125).

The number of **basic building blocks** in MPI is small.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_INIT</td>
<td>initialize MPI</td>
</tr>
<tr>
<td>MPI_COMM_SIZE</td>
<td>determine how many processes there are</td>
</tr>
<tr>
<td>MPI_COMM_RANK</td>
<td>find out which is my process number</td>
</tr>
<tr>
<td>MPI_SEND</td>
<td>send a message</td>
</tr>
<tr>
<td>MPI_RECV</td>
<td>receive a message</td>
</tr>
<tr>
<td>MPI_FINALIZE</td>
<td>terminate MPI</td>
</tr>
</tbody>
</table>

Let's start programming MPI:

```c
PROGRAM hello
C ---> A simple "hello world" program for MPI/C
IMPLICIT NONE
INCLUDE "mpif.h"
INTEGER ierror, rank, size

CALL MPI_INIT(ierr)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
IF (rank .EQ. 0) WRITE(*,*) 'Hello world!'
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
WRITE(*,*) 'I am ', rank, ' out of ', size
CALL MPI_FINALIZE(ierr)

STOP
END
```
Let's start programming MPI:

```c
/* A simple "hello world" program for MPI */
#include <mpi.h>
#include <stdio.h>

int main(int argc, char *argv[]) {
    int size, rank;

    MPI_Init(&argc, &argv); /* Initialize MPI */
    MPI_Comm_size(MPI_COMM_WORLD, &size); /* Get the number */
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); /* Get my number */
    printf("Hello World!\n"); /* Print a message */
    MPI_Finalize(); /* Shut down and clean */

    return 0;
}
```

Maya Neytcheva, IT, Uppsala University maya@it.uu.se