FORTRAN and MPI

Message Passing Interface (MPI)

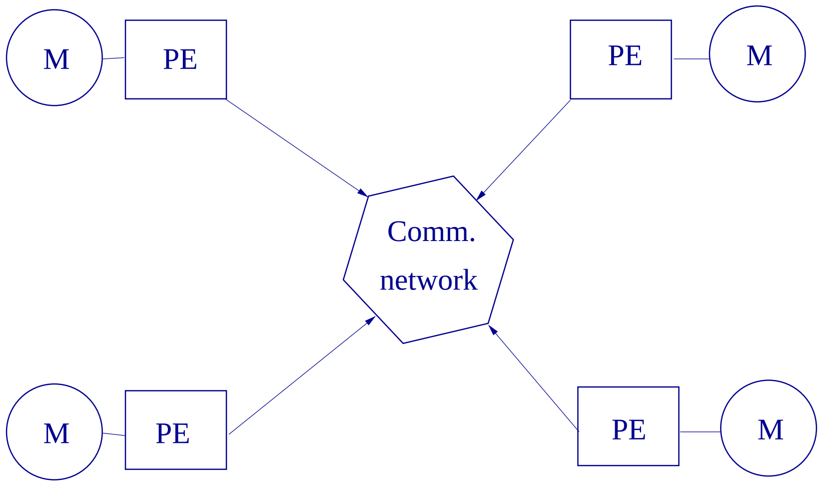
Day 1

DANISH CENTER FOR APPLIED MATHEMATICS AND MECHANICS DCAMM

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

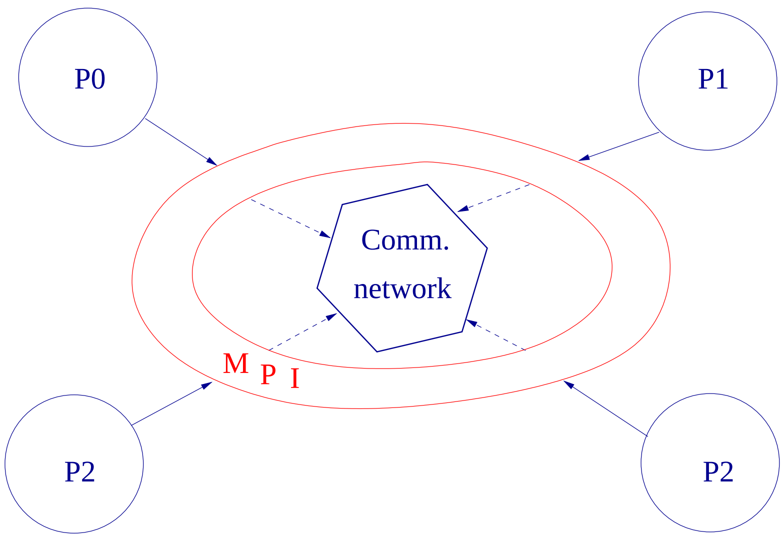


PE - processing elements

M - memories

DANISH CENTER FOR APPLIED MATHEMATICS AND MECHANICS DCAMM

Message passing programming paradigm



P - processes

- 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 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 mail box, phone line or a fax machine.

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

- The sender must have addresses to sent 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.

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

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

MPI Programs

Header files

Should appear everywhere you call MPI procedures.

```
C:           #include <mpi.h>

Fortran:     include 'mpif.h'
```

Fortran datatypes:

MPI datatype	Fortran datatype
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)
MPI_BYTE	
MPI_PACKED	

C datatypes

MPI datatype	C datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_CHAR	unsigned char
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int

Initialising MPI

```
C:

int MPI_Init (int *argc, char ***argv)
--------------------------------------
Fortran:

subroutine MPI_Init(ierror)
integer ierror
```

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)

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(ierror)
integer ierror
```


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 MPI standard contains many functions ($\geq$ 125).

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

MPI_INIT	initialize MPI
MPI_COMM_SIZE	determine how many processes there are
MPI_COMM_RANK	find out which is my process number
MPI_SEND	send a message
MPI_RECV	receive a message
MPI_FINALIZE	terminate MPI


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

       CALL MPI_INIT(ierror)
       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(ierror)

       STOP
       END
```

```
/* A simple "hello world" program for MPI/F         */
#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;
}
```