Scientific Computing on Parallel Machines

Cálculo científico en ordenadores paralelos

Markus Uhlmann

CIEMAT

www.ciemat.es/sweb/comfos/personal/uhlmann

UCM – Abril 2008

Schedule

Part I	Introduction to parallel programming	lecture 1
Part II	Introduction to MPI	
	General introduction	
	& MPI program structure	lecture 2
	Point-to-point communication	lecture 3,4
	Collective communication	lecture 5
	2D Poisson example	lecture 6,7
	Non-contiguous data & Mixed datatypes	lecture 7
	Virtual topologies & Communication subsets	lecture 8
	Use of linear algebra libraries	lecture 9
	Extensions	lecture 10

Summary of last lecture

Motivation & history of parallel computing

Parallel computer architectures

- Flynn's taxonomy $\longrightarrow$ MIMD
- shared vs. distributed memory, hybrid

Parallel programming models

- threads (OpenMP) vs. message passing (MPI)

Summary of last lecture (continued)

Design of parallel programs

- identify potential concurrency (Amdahl's law!)
- partitioning: domain vs. functional decomposition
- communication patterns
- data dependency
- load balancing
- granularity

Part II Introduction to MPI

Message Passing Model

Characteristics

- each process works on LOCAL data
  & DIFFERENT instructions (MIMD)
  BTW: process = processor = task
- EXPLICIT interchange of data by function calls
- exchange is MUTUAL process (changed in MPI-2)

$\Rightarrow$ flexibility, performance

BUT: time-consuming implementation

Comparison of parallel programming paradigms

Message Passing (MPI)

- adapted to distributed memory architectures
- efficiency
- portability
- flexibility

Multithreading (OpenMP)

- adapted to shared memory architectures
- simplicity

What is MPI?

Message Passing Interface

- library of communication subroutines/functions
- implementation defined in standards MPI-1, MPI-2
- well adapted to machines with distributed memory
- wide availability (also: shared memory and hybrid)
- interfaces for `C` and `FORTRAN`

MPI standards

MPI-1 (version 1.2)

- defined in 1994-7 by Message Passing Interface Forum
- defines naming conventions and behavior of functions
  $\longrightarrow$ portability
- implementation is left to vendors (optimization)

MPI-2

- adds additional functionality
  (parallel I/O, Fortran 90, C++, one-sided communication)
- starting to become widely available

Where to get more information?

References for the MPI standard

- online reference book for the MPI standard 1.2
- short summary of MPI library syntax (MPICH)
- NCSA MPI web course
- definition of the MPI standard 2.0

MPI Basic Syntax

General syntactic conventions

- definitions are completely analogous in C and Fortran
- naming convention: routines/functions, constants

  `MPI_XXXXX`
- return values: integer

```
ierr=MPI_Finalize();
call MPI_FINALIZE(ierr)
```

  one exception:

```
double t1=MPI_WTIME()
double precision t1=MPI_WTIME()
```

- return values can be compared with constant `MPI_SUCCESS`

MPI Basic Syntax – predefined data types

- data types: defined in transparent way for the user

C	Fortran
MPI_LONG, MPI_SHORT,...	MPI_INTEGER
MPI_FLOAT, MPI_DOUBLE	MPI_REAL, MPI_DOUBLE_PRECISION
MPI_CHAR, MPI_UNSIGNED_CHAR	MPI_CHARACTER
	MPI_COMPLEX, MPI_DOUBLE_COMPLEX
	MPI_LOGICAL
MPI_BYTE	MPI_BYTE
MPI_PACKED	MPI_PACKED

- assures portability

MPI data types & language bindings

C

- `MPI_Comm` – a communicator
- `MPI_Status` – a structure containing status information
- `MPI_Datatype` – a type declaration

  example:
  ```
  MPI_Status my_status;
  ```
- arrays are indexed starting with 0
- logicals are integers with 0 (false), non-zero (true)

MPI data types & language bindings

Fortran

- all integer values

  example:
  `integer my_status(MPI_STATUS_SIZE)`
- array arguments are indexed from 1
- binary values are of type `LOGICAL`

MPI Program Structure

Components

1. include MPI header file (internal declarations)
2. MPI-related variable declarations
3. initialize the MPI environment
4. ... computation and MPI communication calls ...
5. close MPI communications

Example "Hello World" (FORTRAN)

```
      PROGRAM HELLO
      INCLUDE 'mpif.h'
      INTEGER myrank, size, ierr
C     Initialize MPI:
      call MPI_INIT(ierr)
c     write welcome messages
      WRITE(*,*)"Hello World!"
C     Terminate MPI
      call MPI_FINALIZE(ierr)
      END
```

(source)

Example "Hello World" (C)

```c
#include <stdio.h>
#include <mpi.h>
void main (int argc, char *argv[]) {
    int myrank, size;
/* Initialize MPI */
    MPI_Init(&argc, &argv);
/* write a welcome message */
    printf("Hello World!\n");
/* Terminate MPI */
    MPI_Finalize();
}
```

(source)

Compiling MPI programs

Compilation

- supply the name of the library explicitly
  `f77 <objects> -lmpi`
- use script with pre-defined variables
  `mpif77`, `(mpif90)`, `mpicc`

$\longrightarrow$ generates a single executable file

Executing MPI programs

Execution

- use start-up script of MPI implementation, e.g.
  `mpirun -np <# procs> <executable>`
- run on single processor: `--all-local`

$\longrightarrow$ launches `np` processes executing the same executable

Obtaining the course problem code

http://www.ciemat.es/sweb/comfos/personal/uhlmann/mpi4_code.tar

Before running the example code:

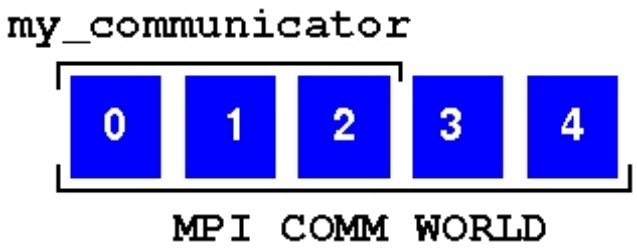
- provide a mechanism for `mpirun` to authenticate
- `ssh-keygen -t rsa`
  (use empty passphrase)
- `cp ~/.ssh/id_rsa.pub ~/.ssh/authorized_keys`
- test it with: `/usr/lib/mpich/sbin/tstmachines`

MPI Communicators

- 'handles' representing a group of processes
- intracommunication: among members of same communicator
- each process has a unique rank within each communicator

$$0 \leq \texttt{rank} \leq \texttt{no. of processes - 1}$$

- a process can belong to different communicators
- default global communicator: `MPI_COMM_WORLD`



Obtaining information about the communicator

Size of the communicator

- `call MPI_COMM_SIZE(comm,size,ierr)`
  `int ierr=MPI_Comm_size(MPI_Comm comm,int *size)`
- `size` is the number of members of the communicator

Procesor rank within the communicator

- `call MPI_COMM_RANK(comm,myrank,ierr)`
  `int ierr=MPI_Comm_rank(MPI_Comm comm,int *myrank)`
- `myrank` is the rank of the calling processor
  $0 \leq$ `myrank` $\leq$ `size-1`

Exercise: obtaining processor identification

(A)

- modify initial "hello world" program
- obtain information about processor id and communicator
- write information to standard output

(B)

- define an array
- perform some local operations on data
- e.g. each processor computes `A(1:n)=myrank*n`
- output to screen

Example "Hello World 2" (FORTRAN)

```
      PROGRAM HELLO
      INCLUDE 'mpif.h'
      INTEGER myrank, size, ierr
C     Initialize MPI:
      call MPI_INIT(ierr)
C     Get my rank:
      call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
C     Get the total number of processors:
      call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierr)
      WRITE(*,*)"Processor",myrank,"of ",size,
     $     ": Hello World!"
C     Terminate MPI
      call MPI_FINALIZE(ierr)
      END
```

(source)

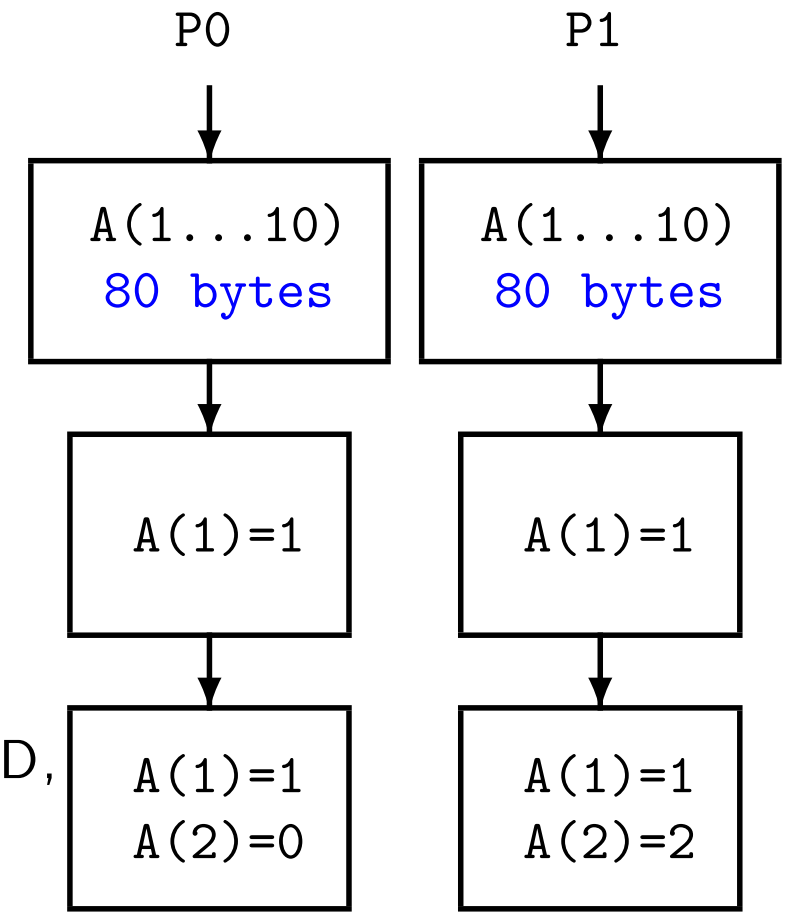
Example "Hello World 2" (C)

```c
#include <stdio.h>
#include <mpi.h>
void main (int argc, char *argv[]) {
    int myrank, size;
/* Initialize MPI */
    MPI_Init(&argc, &argv);
/* Get my rank */
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
/* Get the total number of processors */
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Processor %d of %d: Hello World!\n",
           myrank, size);
/* Terminate MPI */
    MPI_Finalize();
}
```

(source)

Memory Allocation & Access Is LOCAL

```
c allocates 10 x real, locally
c -> global_size=nprocs*10*8byte
      real*8 A(10)
         ...
         ...
c process-independent assignment
      A(1)=1.0
         ...
         ...
c assignment with local value
      call MPI_COMM_RANK(MPI_COMM_WORLD,
     &                   myrank,ierr)
      A(2)=dfloat(myrank)*2.0
```



Types of MPI calls

Locality of calls

- local (e.g. `MPI_COMM_RANK`)
- non-local (all communication calls)
- collective

Execution flow/completion of operation after call

- blocking (e.g. `MPI_SEND`)
- non-blocking (e.g. `MPI_ISEND`)