Message Passing Interface

Basic functions

Outline

- Introduction to the Message Passing Interface
  - Parallel programming model of MPI
  - "MPI for Dummies": the 6 basic instructions.
    - » How to run a MPI program.
  - More avanced MPI:
    - » Collective communication
    - » Non-blocking communication

Message Passing Interface (MPI)

- MPI is a library for Parallel Programming
  - Extends C or Fortran (bindings for C++),
  - Provides abstract datatypes and functions for communication.
- MPI is the de-facto Message Passing Standard
- More than 180 functionalities
  - Point-to-point and colective comm, non-blocking com, abstract datatypes, DMA, logical organization of the processes, dynamicity, etc…
  - "MPI is as simple as using 6 functions and as complicated as a user wishes to make it."
- Two main open source distributions:
  - MPICH www-unix.mcs.anl.gov/mpi/mpich
  - OpenMPI (LAM-MPI): www.open-mpi.org

Development of MPI

- November 1992 First draft of MPI 1
- November 1993 Second draft of MPI 1.0
- June 1994 MPI 1.0
- June 1995 MPI 1.1
- November 1997 MPI 1.2
- November 1997 MPI 2.0
- Late 1998 Partial implementation of MPI 2.0
- 2000 Most of MPI-2 available
- 2005 All major MPI distributions include MPI 2.0

A few references on MPI

- http://www.mpi-forum.org, for the norm MPI.
- International conference: EuroPVM-MPI (LNCS, Springer)
- Books:
  - Gropp, William *et al.*, Using MPI, MIT Press.
  - Gropp, William *et al.*, Using MPI-2, MIT Press.
  - Snir, M. *et al.*, Dongarra, J., MPI: The Complete Reference.

The MPI paradigm

- Each one of the p processes run the same binary program
  - Single program, Multiple Data paradigm.
  - In "basic" MPI you launch the p processes at the start of the program, and all the p processes must run until the end.
- Each process is identified by a unique rank (a number between 0 and p-1).
- Based on the rank, each process can:
  - Execute tests (if... then) to run those parts of the program that are relevant;
    - » (Advanced use): nothing prevents the processes to launch threads...
  - Send/receive messages to/from any other given process.
    - » There are many types of possible messages.

MPI in 6 words

1) MPI_Init(&argc, &argv)   // No comment.
2) MPI_Comm_rank(&r, communicator)
   // returns the rank in the var. int ´r´
3) MPI_Comm_size(&p, communicator)
   // returns the number of processes in the var. int ´p´
4) MPI_Send(/* a bunch of args */)
5) MPI_Recv(/* almost the same bunch of args */)
6) MPI_Finalize()   // No comment

The basic communicator is MPI_COMM_WORLD.

What is a MPI message?

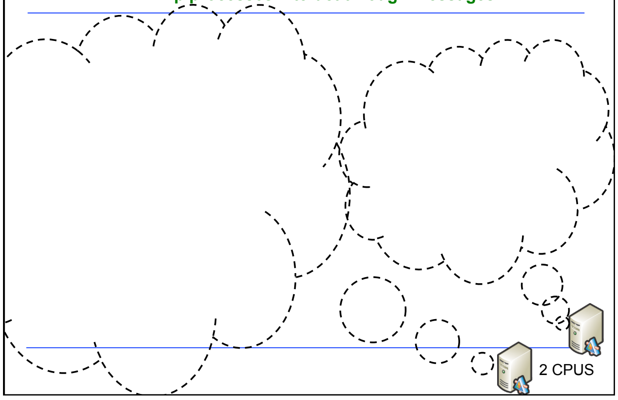
- Look at the MPI_send function:
  int MPI_Send(void*, int, MPI_Datatype, int, int, MPI_Comm).
- Typical call:
  MPI_Send(&work, 1, MPI_INT, dest, WORKTAG, MPI_COMM_WORLD);
- It sends the content of a buffer from the current process to the receiver ´dest´ process.
- The buffer is defined by the 3 first arguments:
  - Work (void*): pointer to the memory area where the data are found.
  - 1, MPI_INT: number and basic type of the data (almost sizeof()!)
- A message is identified by a tag (see WORKTAG).
  - The tag must be the same in the Recv and Send.
  - The type is irrelevant to the matching algorithm between sender and receiver.

MPI_Recv

- Profile of the call:
  int MPI_Recv(void*, int, MPI_Datatype, int, int, MPI_Comm, MPI_Status*)
- Typical use:
  MPI_Status* s;
  int d, TAG = 103;
  MPI_Recv(&d, 1, MPI_INT, source, TAG, MPI_COMM_WORLD, s);
- ‘source’ is the rank of the sender proc., TAG is the tag of the message.
- This call is blocking.
  - When the process executes the next instruction, d contains the data that was expected.

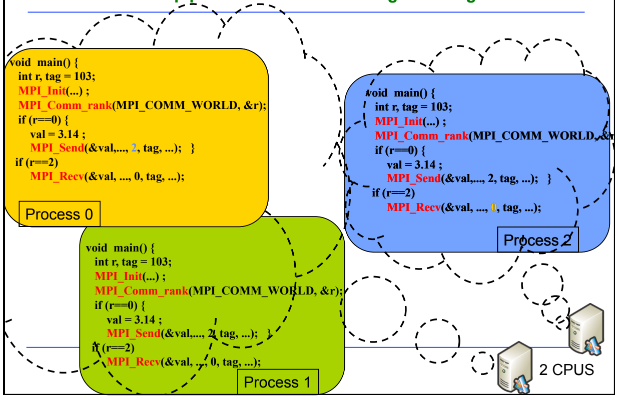
Programming with MPI

p processes interact through messages.



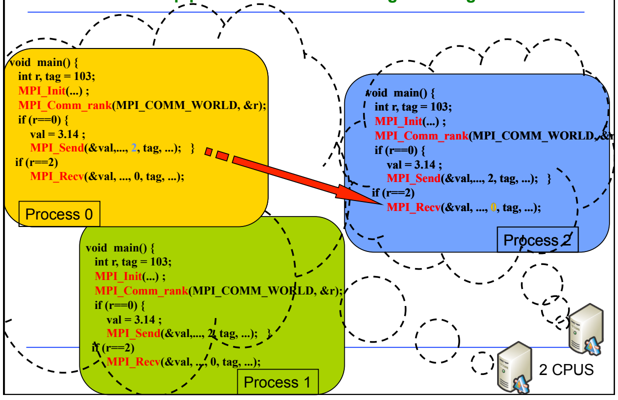
Programming with MPI

p processes interact through messages.



Programming with MPI

p processes interact through messages.



Whole example

```
void  main() {
  int p, r, tag = 103;
  MPI_Status stat;
  double val ;
  MPI_Init(&argc, &argv) ;
  MPI_Comm_rank(MPI_COMM_WORLD, &r);
  if (r==0) {
     printf("Processor 0 sends a message to 1\n");
     val = 3.14 ;
     MPI_Send(&val, 1, MPI_DOUBLE, 1, tag, MPI_COMM_WORLD);   }
  else {
     printf("Processor 1 receives a message from 0\n");
     MPI_Recv(&val, 1, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD, &stat);
     printf("I received the value: %.2lf\n", valor);   }
}
```

A few observations

- MPI is much more powerfull than trivial Master/Slave programming.
  - A frequent template: par ranks processos perform something, impar ranks processes run something else.
- The tags must be predefined by the programmer.
  - The set (source, tag, dest) identifies the message, so be carefull with casts.
- The buffer is a contiguous memory area
  - Non-contiguous data must be serialized before being sent.
- A message has a fixed size, which must be known before issuing a MPI_recv
  - It is frequent to have to send 2 messages:
    - » First the size (1 int),
    - » Then the "real" message (´size´ elements).

Blocking vs. Non-blocking communication

- MPI_Recv(&x,...) is blocking.
- MPI_Send(&x,...) is "non-blocking"
  - But x is copied into na internal buffer.
  - Send is non-blocking... Until the internal buffer gets full!
- Non-blocking variants: MPI_Irecv() and MPI_Isend()
  - Same args as Recv/Send, with one extra of type MPI_Request.
  - The MPI_Request enables the testing of the completion of the non-blocking communication.
- Explicitly bufferized versions of Send/recv: MPI_Bsend, MPI_Brecv().

Test & Wait non-blocking comm.

- MPI_Test(MPI_Request* req, int* flag, MPI_Status* stat)
  - Sets 'flag' to 0 or 1, depending of 'req'
  - You have to test 'flag' afterwards (if (flag)...)
- MPI_Wait(MPI_Request* req, MPI_Status* stat)
  - Waits until the completion of the non-blocking comm.
- Key for High-Performance: computation/ communication overlap.
  - » Launch a non-blocking communication (e.g recv),
  - » (in a loop) run all you can run, without having received the data, and test regularly for the reception.
  - » If the loop ends up, then block with a Wait.

That's all folk!

- See you next week!