

# Introduction to MPI

MECA 2300

February 16, 2010

# Distributed Memory

- each process access only to its own memory
- messages are send between processes
- cheaper than shared memory for high number of CPU
- explicit communications : implementation requires more work
- distributed and shared memory infrastructures can be mixed
- *MPI Message Passing Interface* is a standard allowing code portability accross various architectures, *openmpi* is a commonly used open source implementation of this standard.

# Communications

- Point to point
  - `MPI_Send` family
  - `MPI_Receive` family
- Collective inside one `MPI_Communicator`
  - all to one `MPI_Gather`
  - one to all
    - same data to all processes `MPI_Bcast`
    - different data to each process `MPI_Scatter`
  - all to all
    - same data to all processes `MPI_AllScatter`
    - different data to each process `MPI_AllToAll`
  - barrier `MPI_Barrier`

# MPI\_Communicator

- = one group of processes communicating together
  - number of processes composing the group `MPI_Comm_size`
  - id of this process inside the group `MPI_Comm_rank`
  - `MPI_COMM_WORLD` is the group of all the processes launched with `mpirun`

# Hello World

```
#include <stdio.h>
#include <mpi.h>
int main (argc, argv) int argc; char *argv[];
{
    int rank, size,
    MPI_Init (&argc, &argv);
    MPI_Comm_rank (MPI_COMM_WORLD, &rank);
    MPI_Comm_size (MPI_COMM_WORLD, &size);
    printf( "Hello world from process %d of %d\n",
        rank, size );
    MPI_Finalize();
    return 0;
}
```

**MPI\_Init** before the first MPI call, **MPI\_Finalize** after the last one

# Compile and Run

- Commands
  - compile `mpicxx test1.cc -o test`
  - run `mpirun -np 4 ./test test_args`
- Remarks
  - depending of your installation, command names may vary e.g. `mpirun.openmpi`, `openmpirun`, ...
  - the number of processes does not have to equals the number of processors (especially for debugging purpose)
  - on real clusters, those commands are generally inside submission scripts launched with command like `qsub`

# Deadlocks

Process 0 | Process 1

---

## Deadlock

Recv(1) | Recv(0)  
Send(1) | Send(0)

Both processes are waiting for data before sending starts

## Unsafe

Send(1) | Send(0)  
Recv(1) | Recv(0)

Depends on the implementation  
Send may not return (e.g. buffer full)

## Reorder

Send(1) | Recv(0)  
Recv(1) | Send(0)

Always works  
Can be tricky in practice

## Non-blocking

ISend(1) | ISend(0)  
IRecv(1) | IRecv(0)  
Wait | Wait

Always works  
Sent data cannot be modified before the communication completes

## Round of communications example

Process  $i$  send data to process  $i+1$

```
MPI_Status status;
MPI_Request request;
int tag = 1;
if(rank != nthreads-1)
    MPI_Isend(sendptr, n, MPI_INT, rank+1, tag,
              MPI_COMM_WORLD, &request);
if(rank != 0)
    MPI_Recv(data, n, MPI_INT, rank-1, tag,
              MPI_COMM_WORLD, &status);
MPI_Wait(&request, &status);
```

# Matrix with ghosted rows