MPI – Point-to-point communication

- Communication between process 0 and process 1: hello_1to0.c

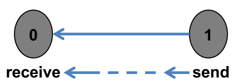
```
#include <stdio.h>
#include <string.h>
#include <sched.h>  //with #define _GNU_SOURCE before
#include "mpi.h "

int main(int argc, char **argv)
{
   int        myRank, nbProcs, cpuId=-1, tag=50, nameLength;
   char       message[512], procName[MPI_MAX_PROCESSOR_NAME];
   MPI_Status status;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
   MPI_Comm_size( MPI_COMM_WORLD, &nbProcs);

   MPI_Get_processor_name(procName, &nameLength);
   cpuId = sched_getcpu();
```

24

MPI – Point-to-point communication

- Communication between process 0 and others: hello_1to0.c

```
   if ( myRank == 1 ) { // processes of rank 1
      sprintf( message, "Hello from %d on node %s-cpu%d!",
               myRank, procName, cpuId );
      MPI_Send( message, strlen(message)+1, MPI_CHAR,
                0, tag, MPI_COMM_WORLD ); }
   else if ( myRank == 0 ) // process of rank 0
      MPI_Recv( message, 512, MPI_CHAR, 1,
                tag, MPI_COMM_WORLD, &status );
      printf( "Proc %d: %s\n", myRank, message );
   }

   MPI_Finalize();
   return 0;
}
```



25

MPI – Datatypes

- Similar to C, examples:

MPI datatype	C datatype
MPI_CHAR	char
MPI_INT	int
MPI_UNSIGNED	unsigned int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_BYTE	

- Complex data types
  - `MPI_PACKED` performance of data communication
  - Derived types: structure, column of matrix …

26

MPI – Point-to-point communication

- Communication between process 0 and others: hello_master.c

variables for communication

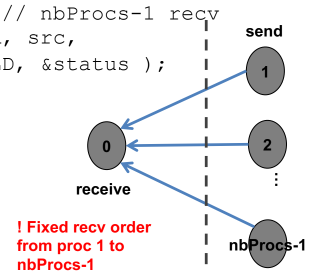
```
#include <stdio.h>
#include <string.h>
#include <sched.h>
#include "mpi.h "
int main(int argc, char **argv)
{
   int        myRank, nbProcs, cpuId=-1, src, tag=50, nameLength;
   char       message[100], procName[MPI_MAX_PROCESSOR_NAME];
   MPI_Status status;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
   MPI_Comm_size( MPI_COMM_WORLD, &nbProcs);

   MPI_Get_processor_name(procName, &nameLength);
   cpuId = sched_getcpu();
```

27

MPI – Point-to-point communication

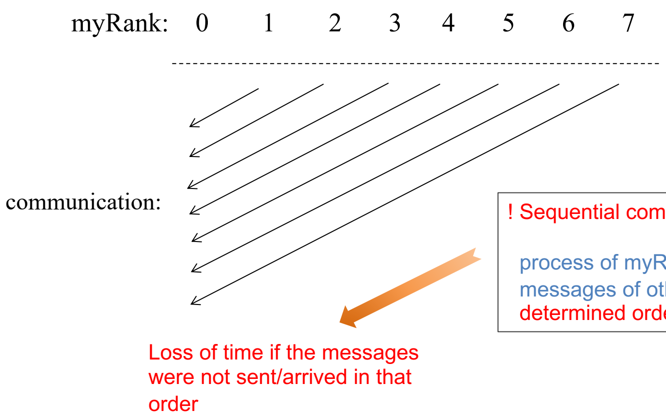
- Communication between process 0 and others: hello_master.c

```
   if ( myRank != 0 ) { // processes of rank ≠ 0
      sprintf( message, "Hello from %d on node %s-cpu%d!",
               myRank, procName, cpuId );
      MPI_Send( message, strlen(message)+1, MPI_CHAR,
                0, tag+myRank, MPI_COMM_WORLD ); }
   else // process of rank 0
      for ( src=1; src<nbProcs; src++ ) { // nbProcs-1 recv
         MPI_Recv( message, 100, MPI_CHAR, src,
                   tag+src, MPI_COMM_WORLD, &status );
         printf( "%s\n", message );
      }

   MPI_Finalize();
   return 0;
}
```



MPI – Point-to-point communication

- Running of hello_master.c with 8 processes



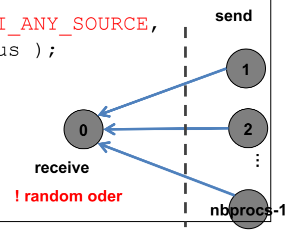
! Sequential communication !

process of myRank=0 receives the messages of others processes in a determined order.

MPI – Point-to-point communication

- Modify the reception order of messages

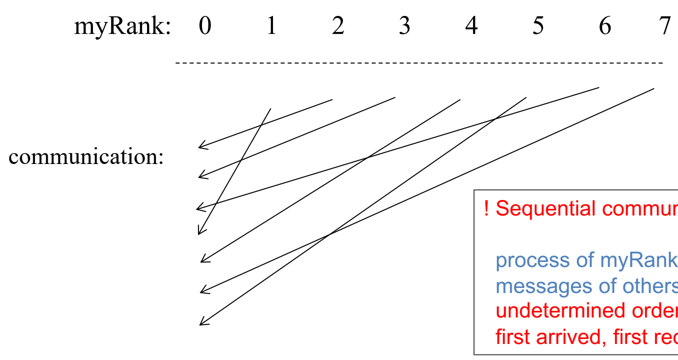
```
   if ( myRank != 0 ) {
      sprintf( message, "Hello from %d on %s-cpu%d!",
               myrank, procName, cpuId );
      MPI_Send( message, strlen(message)+1, MPI_CHAR,
                0, tag, MPI_COMM_WORLD ); }
   else
      for ( src=1; src<nbProcs; src++ ) {
          MPI_Recv( message, 100, MPI_CHAR, MPI_ANY_SOURCE,
                    tag, MPI_COMM_WORLD, &status );
          printf( "%s\n", message );
      }

   MPI_Finalize();
   return 0;
}
```



30

MPI – Point-to-point communication

- Run the program with 8 processes



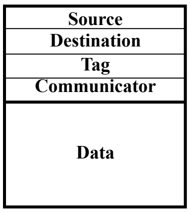
! Sequential communication !

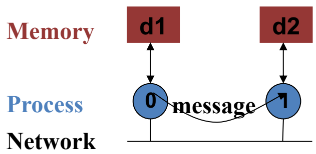
process of myRank=0 receives the messages of others processes in a undetermined order.
first arrived, first received

31

Point-to-Point Communication

- Communication between two processes
  - Source and Destination
- `Message = header + data`
- Data conversion if necessary
- Transmission mechanism





32

Point-to-Point Communication

- Blocking communication
  - Blocking send and receive: `MPI_Send, MPI_Recv`
- Parameters of `MPI_Send` and `MPI_Recv`
  - Data address
  - Elements number of data
  - Type of data elements: `MPI_Datatype`
  - Source or Destination of the message (`MPI_ANY_SOURCE`)
  - Tag of message (`MPI_ANY_TAG`), may be used to indicate different type of message
  - Communicator (`MPI_COMM_WORLD`): `MPI_Comm`
  - Status: `MPI_Status` (`MPI_SOURCE`, `MPI_TAG`, `MPI_ERROR`)

33

MPI – Point-to-point communication

- Two side operation
  - A Send must be matched by a Recv
- Safe program in blocking communication

```
MPI_Comm_rank (MPI_COMM_WORLD, myRank);
if (myRank == 0) {
   MPI_Send( sendBuf, count, MPI_INT, 1, tag1, MPI_COMM_WORLD );
   MPI_Recv( recvBuf, count, MPI_INT, 1,
             tag2, MPI_COMM_WORLD,  &status );
}
else if (myRank == 1) {
   MPI_Recv( recvBuf, count, MPI_INT, 0,
             tag1, MPI_COMM_WORLD, &status );
   MPI_Send( sendBuf, count, MPI_INT, 0, tag2, MPI_COMM_WORLD );
}
```

34

MPI – Point-to-point communication

- Two side operation
  - A Send must be matched by a Recv
- Safe program in blocking communication

```
MPI_Comm_rank (MPI_COMM_WORLD, myRank);
if (myRank == 0) {
   MPI_Send( sendBuf, count, MPI_INT, 1, tag1, MPI_COMM_WORLD );
   MPI_Recv( recvBuf, count, MPI_INT, 1,
             tag2, MPI_COMM_WORLD,  &status );
}
else if (myRank == 1) {
   MPI_Send( sendBuf, count, MPI_INT, 0, tag2, MPI_COMM_WORLD );
   MPI_Recv( recvBuf, count, MPI_INT, 0,
             tag1, MPI_COMM_WORLD, &status );
}
```

possible dead lock if we exchange the order of Recv and Send ! (depending on the implementation of MPI)

35

MPI – Blocking communication

- Features
  - Completion of `MPI_Send` means send variable can be reused
  - Completion of `MPI_Recv` mean receive variable can be read
  - Cause synchronization -> Increase communication time
  - Affect the performance of parallel program
- Solution (if there are many communication between two processes)
  - Non-blocking communication

36

MPI – Non-blocking communication

- Operation in 2 steps
  - Request: `MPI_Isend`, `MPI_Irecv`

```
MPI_Isend(buf, count, datatype, dest,
          tag, comm, &request);
MPI_Irecv(buf, count, datatype, src,
          tag, comm, &request);
```

  - Completion: `MPI_Wait(&request, &status);`
  - Test of completion:

```
MPI_Test(&request, &flag, &status);
```

  - Avoid dead lock
  - Allow communication / computation overlapping
  - Persistent request can be used if many communication

37

MPI – Non-blocking communication

- Example

```
#include <stdio.h>
#include <string.h>
#include <mpi.h>

int main( int argc, char **argv ) {
    int myRank=-1, nbProcs=0;
    char sendMsg[128], recvMsg[128];

    MPI_Request requestS, requestR;
    MPI_Status status;
                                          use with nbProcs = 2!

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
    MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );

    sprintf(sendMsg, "Hello from proc %d/%d!", myRank, nbProcs);
```

38

MPI – Non-blocking communication

- Example

```
    if (myRank == 0) {
       MPI_Isend(sendMsg, strlen(sendMsg)+1, MPI_CHAR, 1,
                 100, MPI_COMM_WORLD, &requestS);
       MPI_IRecv(recvMsg, 128, MPI_INT, 1,
                 200, MPI_COMM_WORLD,  &requestR);
    }
    if (myRank == 1) {
       MPI_Isend(sendMsg, strlen(sendMsg)+1, MPI_CHAR, 0,
                 200, MPI_COMM_WORLD, &requestS);
       MPI_IRecv(recvMsg, 128, MPI_CHAR, 0,           nbProcs = 2!
                 100, MPI_COMM_WORLD, &requestR);
    }
    MPI_Wait(&requestR, &status);
    printf("Proc %d: received message: %s\n", myRank, recvMsg);
    MPI_Finalize();
    return 0;
}
```

39

MPI – Non-blocking communication

- Example

```
if (myRank == 0) {
    MPI_Isend(sendMsg, strlen(sendMsg)+1, MPI_CHAR, 1,
              100, MPI_COMM_WORLD, &requestS);
    MPI_Recv(recvMsg, 128, MPI_INT, 1,
             200, MPI_COMM_WORLD,  &status);
}

if (myRank == 1) {
    MPI_Isend(sendMsg, strlen(sendMsg)+1, MPI_CHAR, 0,
              200, MPI_COMM_WORLD, &request1);
    MPI_Recv(recvMsg, 128, MPI_CHAR, 0,
             100, MPI_COMM_WORLD, &status);
}
```

mixte MPI_Isend and MPI_Recv is possible.

40

41

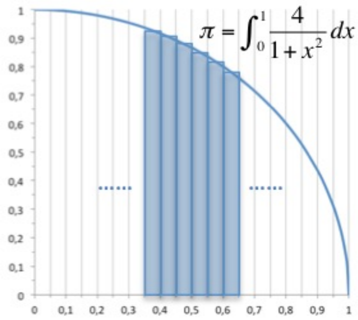
MPI – Point-to-Point Communication

- Exercise: Computing of PI – Numerical integration principle
  - Mathematical formula

    $$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

  - Rectangle rule
    - ☆Let n be the number of rectangles
    - ☆Let h be the width of rectangles
    - ☆We have: $h = \frac{1}{n}$
  - An approximation of $\boldsymbol{\pi}$:

$$\sum_{i=0}^{n-1} h \frac{4}{1 + (ih + 0{,}5h)^2}$$



MPI – Point-to-Point Communication

- Exercise: Sequential computing of PI

```
#include <stdio.h>

int  main(int argc, char **argv)
{
   int    i, nbRects=1000000; // read from keyboard in program
   double x, h, sum=0.0, pi=0.0;

   printf("Please input the number of rectangles of [0-1]: ");
   scanf("%d", &nbRects);   // if (argc>1) nbRects = atoi(argv[1]);

   h = 1.0 / nbRects ;

   for (i=0; i<nbRects; i++) {
      x = (i+0.5)*h;    sum += 4.0 / (1.0 + x*x);
   }

   pi = h * sum ;
   printf("Pi is approximatly: %.16f\n", pi) ;
   return 0;
}
```

```
double PI25DT = 3.141592653589793238462643;
```

MPI – Point-to-Point Communication

- Exercise: Parallel computing of PI – Send/Recv

```
#include <stdio.h>
#include <mpi.h>

int  main(int argc, char **argv)
{
   int    i, nbRects= 1000000, rectsPerProc=0;
   int    myDeb=0, myEnd=0;
   double x, h, mySum=0.0, pi=0.0;

   int         myRank, nbProcs, tag=50;
   MPI_Status status;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
   MPI_Comm_size( MPI_COMM_WORLD, &nbProcs);
```

44

MPI – Point-to-Point Communication

- Exercise: Parallel computing of PI – Send/Recv

```
if (myRank==0) { // process of rank 0 does inputs
    scanf("%d%*c", &nbRects);

    rectsPerProc = nbRects / nbProcs;

    // rank 0 sends rects_per_proc to the other
    for (i=1; i<nbProcs; i++)
        MPI_Send(&rectsPerProc,1,MPI_INT,i,tag,MPI_COMM_WORLD);
}
else // process of rank≠0 receives rects_per_proc from rank 0
    MPI_Recv(&rectsPerProc, 1, MPI_INT, 0, tag,
             MPI_COMM_WORLD, &status);

h = 1.0 / (rectsPerProc*nbProcs);
```

45

MPI – Point-to-Point Communication

- Exercise: Parallel computing of PI – Send/Recv

```
myDeb=myRank*rectsPerProc;  myEnd=myDeb+rectsPerProc;
for (i=myDeb; i<myEnd; i++) {
   x = (i+0.5)*h;    mySum += 4.0 / (1.0 + x*x);
}
if (myRank != 0)
   MPI_Send(&mySum, 1, MPI_DOUBLE, 0, tag, MPI_COMM_WORLD);
else {  // rank 0
   for (i=1; i<nbProcs; i++) {
      MPI_Recv(&pi, 1, MPI_DOUBLE, MPI_ANY_SOURCE,
               tag, MPI_COMM_WORLD, &status);
      mySum = mySum + pi;
   }
   pi = mySum * h; printf("Pi is approximatly %0.16f\n", pi);
}
MPI_Finalize();   return 0;
}
```

46

Exercise: Calculation of PI – Send/Recv

- Resume
  - The computation of the `sum` is well distributed.
  - Sequential communication involves all processes at the beginning and the end of program.

    → May be improved by using collective communication
- To show more collective communication functions, we assume that only process of `rank 0` has the value of `nbRects` at the beginning of the program

47

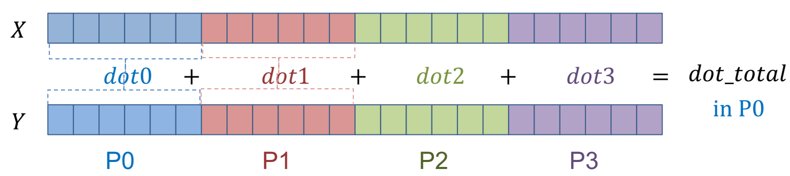
48

MPI – Point-to-Point Communication

- Exercise - Dot product
  - Compute $\mathrm{d} = X \cdot Y,\ X, Y \in \mathbb{R}^n,\ n$ big number

$$d = \sum_{i=0}^{n-1} X_i Y_i, i = 0, \dots, n-1;\ X_i, Y_i \in \mathbb{R}$$

  - Parallel algorithm - Example with 4 processes



49

50

50

51

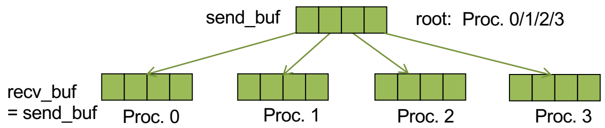
Collective Communication

- What is it?
  - Communication involving all processes of a group
- Objective
  - Increase the performance of parallel program
- How?
  - By reduce of idle processes ⟹ decrease the communication time
- Use cases
  - When I/O
  - Parallel algorithms need collective communication

51

MPI – Collective communication

- Broadcast (diffusion)
  - A process (`root`) has a message to send to others



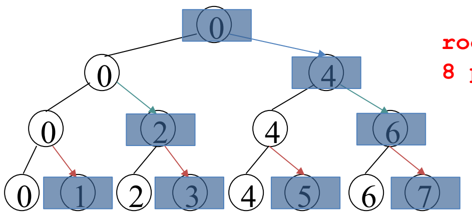

```
int MPI_Bcast(void *msg, int count,
     MPI_Datatype datatype, int root, MPI_Comm comm);
```

```
int send_buf[4];
//root initialize the send_buf
MPI_Bcast(send_buf, 4, MPI_INT, 0, MPI_COMM_WORLD);
```

52

MPI – Collective communication

- Broadcast: Possible implementation



root=0
8 processes

53

MPI – Collective communication

- Broadcast - Broadcast of the dimension of a image

```
int myRank, nbProcs, dims[2], i, tag=30;
…
MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );
if (myRank == 0){
/* Fill dims */
    for ( i=1; i<nbProcs; i++)
        MPI_Send( dims, 2, MPI_INT, i, tag, MPI_COMM_WORLD);
}
else
    MPI_Recv(dims, 2, MPI_INT, 0, tag, MPI_COMM_WORLD);
```

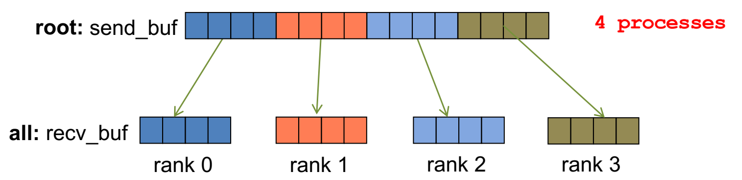
Point-to-point communication:
Number of steps - `O(nbProcs)`

```
int myRank, nbProcs, dims[2], i, tag=30;
…
MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
MPI_Comm_size( MPI_COMM_WORLD, &nbProcs );
MPI_Bcast(dims, 2, MPI_INT, 0, MPI_COMM_WORLD);
```

Collective communication:
Number of steps – $O(\log_2(nbProcs))$ with binary tree

MPI – Collective communication

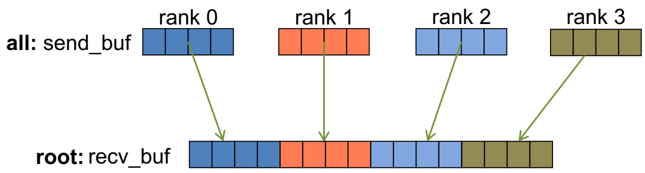
- Scatter – Data distribution




```
int MPI_Scatter( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                 void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                 int root, MPI_Comm comm );
```

MPI – Collective communication

- Gather – Data fusion (fusion)

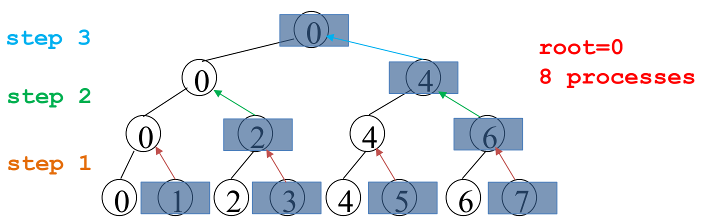



```
int MPI_Gather( void *sendbuf, int sendcnt, MPI_Datatype sendtype,
                void *recvbuf, int recvcnt, MPI_Datatype recvtype,
                int root, MPI_Comm comm );
```

56

MPI – Collective communication

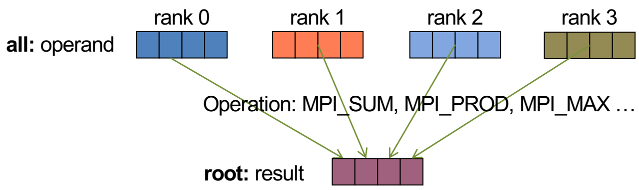
- Gather: Possible implementation



57

MPI – Collective communication

- Reduce – Gather + operation on data (réduction)

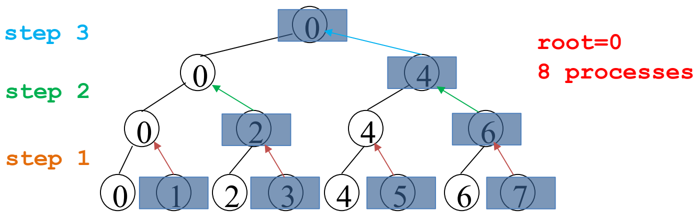



```
int MPI_Reduce( void *operand, void *result,
       int count, MPI_Datatype datatype, MPI_Op op,
       int root, MPI_Comm comm );
```

58

MPI – Collective communication

- Reduce: Possible implementation



59

Collective Communication

- Reduce operations

Operation	Meaning	Operation	Meaning
`MPI_MAX`	Maximum	`MPI_LOR`	logical OR
`MPI_MIN`	Minimum	`MPI_BOR`	bitwise OR
`MPI_SUM`	Sum	`MPI_LXOR`	XOR
`MPI_PROD`	Product	`MPI_BXOR`	
`MPI_LAND`	Logical AND	`MPI_MAXLOC`	max or min +
`MPI_BAND`	bitwise AND	`MPI_MINLOC`	index

60

MPI – Collective communication

- Exercise: Parallel computing of PI

```
#include <stdio.h>
#include <mpi.h>

int  main(int argc, char **argv)
{
   int    i, nbRects= 1000000, rectsPerProc=0;
   int    myDeb=0, myEnd=0;
   double x, h, mySum=0.0, pi=0.0;

   int        myRank, nbProcs, tag=50;
   MPI_Status status;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &myRank );
   MPI_Comm_size( MPI_COMM_WORLD, &nbProcs);
```

61

MPI – Collective communication

- Exercise: Parallel computing of PI

```
if (myRank==0) { // process of rank 0 does inputs
   printf("Please input the number of rectangles for [0, 1]: ");
   scanf("%d%*c", &nbRects);

   rectsPerProc = nbRects / nbProcs;
}

MPI_Bcast(&rectsPerProc, 1, MPI_INT, 0, MPI_COMM_WORLD);

h = 1.0 / (rectsPerProc*nbProcs);
```

62

MPI – Collective communication

- Exercise: Parallel computing of PI

```
   myDeb = myRank * rectsPerProc;
   myEnd = my_deb + rectsPerProc;

   for (i=myDeb; i<myEnd; i++) {
      x = (i+0.5) * h;
      mySum += 4.0 / (1.0 + x*x);
   }
   mySum = h * mySum ;

   MPI_Reduce(&mySum,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

   if (myRank==0)
      printf("Pi is approximatly %0.16f\n", pi);

   MPI_Finalize();
   return 0;
}
```

63

MPI – Collective communication

- Barrier synchronization
  - Make a appointment for all processes
- Use case: time measurement
  - Time measurement for each process

```
double start, exectime;
MPI_Barrier(MPI_COMM_WORLD);
start = MPI_Wtime();
……
exectime= MPI_Wtime()- start;
```

In main function

  - Execution time of program: the one of the slowest process

FIN CM2

64

65