MPI – Time measurement

- MPI program
  - Execution time
    - ☆Time for computation
    - ☆Time for communication
    - ☆Time for processes synchronization & management
  - Execution time measurement: **Wall clock time**
    - ☆Time elapsed between the end and the beginning of a program

66

MPI – Time measurement

- Exercise: Parallel computing of PI

```
#include <stdio.h>
#include <mpi.h>
int  main(int argc, char **argv)
{
   int    i, nb_rectangles=1000000, rects_per_proc=0;
   int    my_deb=0, my_end=0;
   double x, h, my_sum=0.0, pi=0.0;
   int    myRank, nbProcs;

   double start, execTime;

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

67

MPI – Time measurement

- 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", &nb_rectangles);

    rects_per_proc = nb_rectangles / nbProcs;
 }

MPI_Barrier(MPI_COMM_WORLD);
start = MPI_Wtime();

MPI_Bcast(&rects_per_proc, 1, MPI_INT, 0, MPI_COMM_WORLD);
…
MPI_Reduce(&my_sum,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);

execTime= MPI_Wtime()- start;
printf("Proc %d: execTime=%.12f\n", myRank, execTime);
```

and take the largest one

68

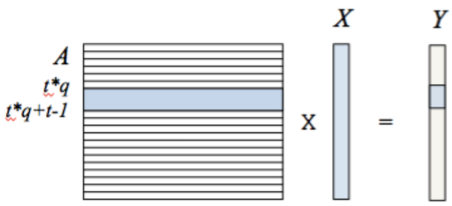
69

MPI – Exercise 2

- Matrix–Vector multiplication
  - Compute $Y = AX$, A: matrix nxn, and $X, Y \in \mathbb{R}^n$, n: a large number

$$Y_i = \sum_{j=0}^{n-1} a_{ij} X_j, i = 0, \ldots, n-1; \;\; a_{ij}, X_j, Y_i \in \mathbb{R}$$

  - Parallel algorithm: line-version



$t = \frac{n}{nbProcs}$

$q: myRank$

full results of several Y_i are in process q

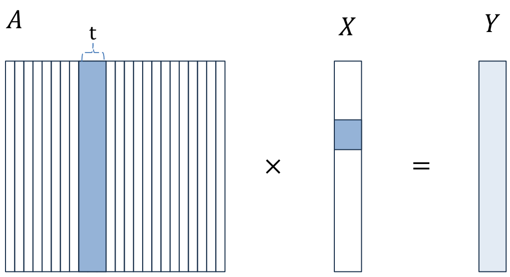
$$Y_i = \sum_{j=0}^{n-1} a_{ij} X_j, i = t * q, \ldots, t * q + t - 1$$

gather of all Y_i in a process, ex. process 0

70

MPI – Exercise 2

- Matrix–Vector multiplication
  - Parallel algorithm: column-version



partial results of all Y_i are in process q

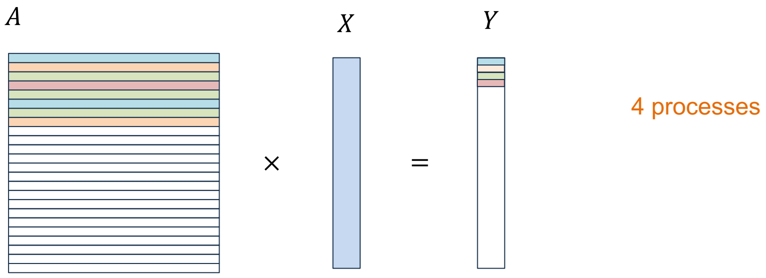
$$Y_i = \sum_{j=t*q}^{t*q+t-1} a_{ij} X_j, i = 0, \ldots, n-1$$

reduction of all partial results of Y_i in a process: ex. process 0

71

MPI – Exercise 2

- Matrix–Vector multiplication
  - Parallel algorithm: 1 line to each process et the beginning; then 1 line to the process which sent the current result if the computation is not complete.



72

MPI – Collective communication

- Exercise – Matrix–Vector multiplication
  - Parallel algorithm: line-version
    - Aim: each process compute several components of Y
    - Algorithm
      1. Distribution of the lines of A and broadcast of X to each process
      2. Every process computes in parallel its Y_i
      3. Gathering of the elements of Y
      4. Possible broadcasting of Y

73

MPI – Collective communication

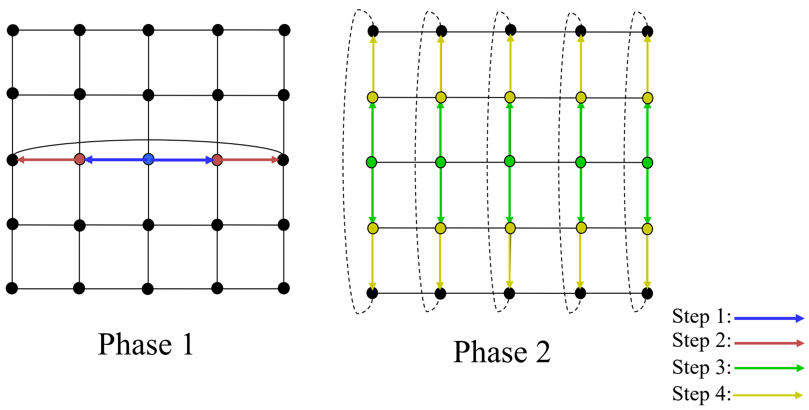
- Exercise – Matrix–Vector multiplication - line-version
  - Process of rank *0*
    1. Initialisation of matrix A and vector X
    2. Broadcast vector X *to all processes* (*MPI_Bcast*)
    3. Distribute the lines of matrix A *to all processes* (*MPI_Scatter*)
    4. Compute its Y components
    5. Gathering of Y components (*MPI_Gather*)
    6. Display Y
  - Other processes
    1. Broadcasting of vector X (*MPI_Bcast*)
    2. Distribution of matrix A's lines (*MPI_Scatter*)
    3. Compute its Y components
    4. Gathering of Y components (*MPI_Gather*)

!!! contiguous memory

74

Matrix-Vector Multiplication

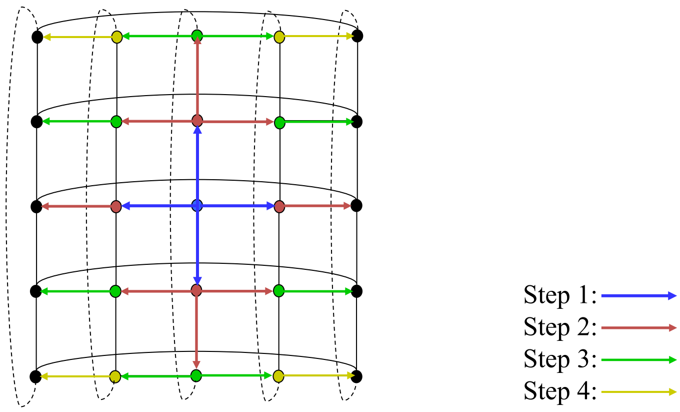
- Distribution in a torus topology – algorithm 1:



75

Matrix-Vector Multiplication

❑Distribution in a torus topology – algorithm 2:

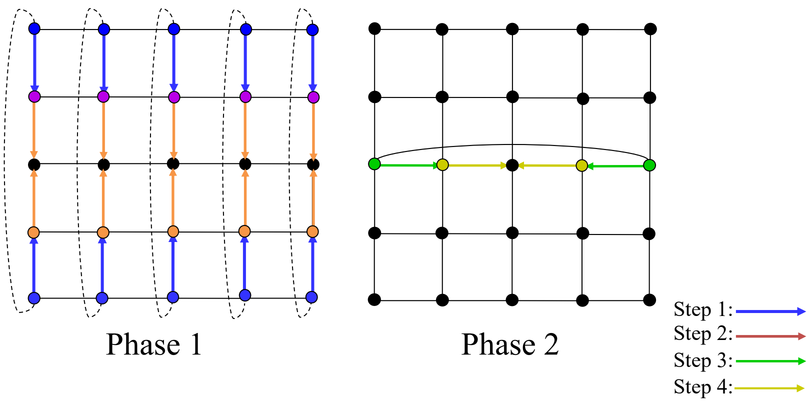


Design of Parallel Programs 76

76

Matrix-Vector Multiplication

❑Gathering / reduce in a torus topology – algorithm 1:

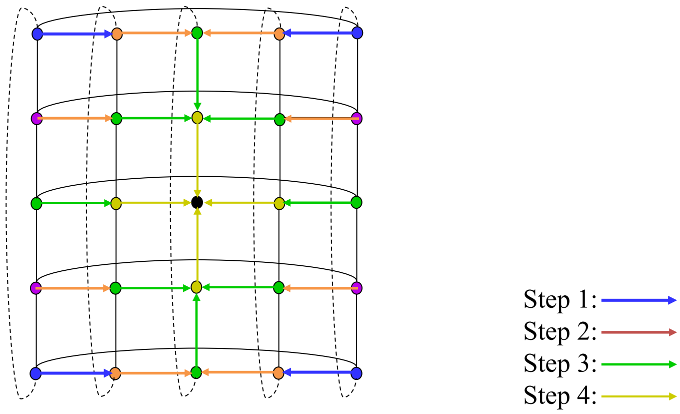


Design of Parallel Programs 77

77

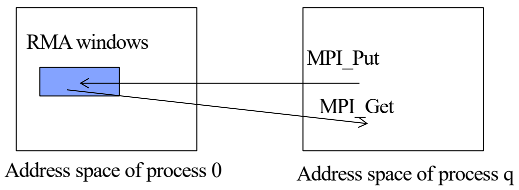
Matrix-Vector Multiplication

❑Gathering / reduce in a torus topology – algorithm 2:



MPI – RMA (Remote Memory Access)

- Direct access to parts of another process' memory
- Use cases
  - Dynamic change of the communication partner
  - Global data
- Software implementation
  - Process x creates a RAM window
  - Process y gets/puts data from/into process x's data window



!!! ending of data transfert by synchronization

MPI – RMA (Remote Memory Access)

- Procedure
  - RMA window creation: example with `MPI_Win_Create`
  - Process synchronization: example `MPI_Fence` (active target mode)
  - RMA operations: `MPI_Get`, `MPI_Put`, `MPI_Accumulate`
  - RMA window free: `MPI_Win_free`
- Characteristics
  - Public memory creation
  - Collective operation
  - One process create the window, accessible by all processes

MPI – RMA

- Exercise – Calculation of PI

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

int  main(int argc, char **argv)
{
   long    i, nb_ rectangles=0, rects_par_proc=0;
   long    my_deb=0, my_end=0;
   double  x, h, sum=0.0, pi=0.0;
    int     myrank, nbprocs;
   MPI_Win win_pi, win_nbRects;

   MPI_Init( &argc, &argv );
   MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
   MPI_Comm_size( MPI_COMM_WORLD, &nbprocs);

   /* Lecture of n from keyboard by process of rank 0 */
```

82

MPI – RMA

- Exercise – Calculation of PI

```
if (myrank==0) {
    MPI_Win_create(&nb_rectangles, sizeof(long), sizeof(long),
        MPI_INFO_NULL, MPI_COMM_WORLD, &win_nbRects);
    MPI_Win_create(&pi, sizeof(double), sizeof(double),
        MPI_INFO_NULL, MPI_COMM_WORLD, &win_pi);
}
else {
    MPI_Win_create(MPI_BOTTOM, 0, sizeof(long),
        MPI_INFO_NULL, MPI_COMM_WORLD, &win_nbRects);
    MPI_Win_create(MPI_BOTTOM, 0, sizeof(double),
        MPI_INFO_NULL, MPI_COMM_WORLD, &win_pi);
}

MPI_Win_fence(0, win_nbRects);
if (myrank != 0)
   MPI_Get(&nb_rectangles,1,MPI_LONG,0,0,1,MPI_LONG,win_nbRects);
MPI_Win_fence(0, win_nbRects);
MPI_Win_free (win_nbRects);
```

83

MPI – RMA

- Exercise – Calculation of PI

```
h = 1.0/nb_rectangles;  rects_par_proc = nb_rectangles/nbprocs;
my_deb = myrank*rects_par_proc;  my_end = my_deb+rects_par_proc;
for (i=my_deb; i<my_end; i++) {
   x = (i+0.5)*h;    sum += 4.0 / (1.0 + x*x);
}
pi = h * sum ;
MPI_Win_fence(0, win_pi);
if (myrank)
   MPI_Accumulate(&pi, 1, MPI_DOUBLE, 0, 0, 1,
                  MPI_DOUBLE, MPI_SUM, win_pi);
MPI_Win_fence(0, win_pi);
if (myrank==0) printf("Pi is approximatly %0.16f\n", pi);
                              MPI_Win_free (win_pi);
MPI_Finalize();  return 0;
}
```

84

85

Data grouping

- Need to send heterogeneous data
- Methods
  - `MPI_Pack` / `MPI_Unpack`
    - Use of a buffer of bytes to send arbitrary data
  - Derived types
    - Corresponding MPI datatype for structure
- Choice of methods
  - `MPI_Pack` / `MPI_Unpack`
    - Heterogeneous data to send a few times
    - Communication of data of variable length
  - Derived types
    - Heterogeneous data to send repeatedly
    - Non-contiguous data of the same type ⇒ `MPI_Type_vector, MPI_Type_indexed`

86

Pack / Unpack

- Objective
  - Put the heterogeneous non-contiguous data together to be sent at one time
- Example

```
int getdata ( int rank, int root, float *a, int *n)
{
   char buffer[100];
   int  position;

   if ( rank == root ) {
      position = 0;
      MPI_Pack( a, 1, MPI_FLOAT, buffer, 100,
                &position, MPI_COMM_WORLD );
      MPI_Pack( n, 1, MPI_INT, buffer, 100,
                &position, MPI_COMM_WORLD );
   }
```

87

Pack / Unpack

- Example (continue)

```
  MPI_Bcast( buffer, 100, MPI_PACKED,
             root, MPI_COMM_WORLD );

  if (rank != root) {
     position = 0;
     MPI_Unpack( a, 1, MPI_FLOAT, buffer, 100,
                 &position, MPI_COMM_WORLD );
     MPI_Unpack( n, 1, MPI_INT, buffer, 100,
                 &position, MPI_COMM_WORLD );
  }
}
```

88

Derived type

- Objective
  - Define heterogeneous data access
- General method
  - Build
  - Validate
  - Destroy

```
typedef struct {
    float a, b;
    int   n;
} indata_type;

void Build_derived_type(indata_type *indata,
       MPI_Datatype *msg_type)
{
    int           blok_lengths[3];
    MPI_Aint      displacements[3], addresses[4];
    MPI_Datatype typelist[3];
```

89

Derived type

- General method (continue)

```
typelist[0] = typelist[1] = MPI_FLOAT; typelist[2] = MPI_INT;
block_lengths[0]= block_lengths[1]= block_lengths[2]=1;
MPI_Address(indata, &addresses[0]);
MPI_Address(&(indata->a), &addresses[1]);
MPI_Address(&(indata->b), &addresses[2]);
MPI_Address(&(indata->n), &addresses[3]);
displacements[0] = addresses[1] - addresses[0];
displacements[1] = addresses[2] - addresses[0];
displacements[2] = addresses[3] - addresses[0];

MPI_Type_struct( 3, block_lengths, displacements,
                 typelist, msg_type );
MPI_Type_commit( msg_type );
}                 MPI_Type_free( msg_type );
```

FIN CM3

90