Lập trình MPI-2 (MPICH) trên Linux PC Cluster

Đỗ Thanh Nghị
dtnghi@cit.ctu.edu.vn

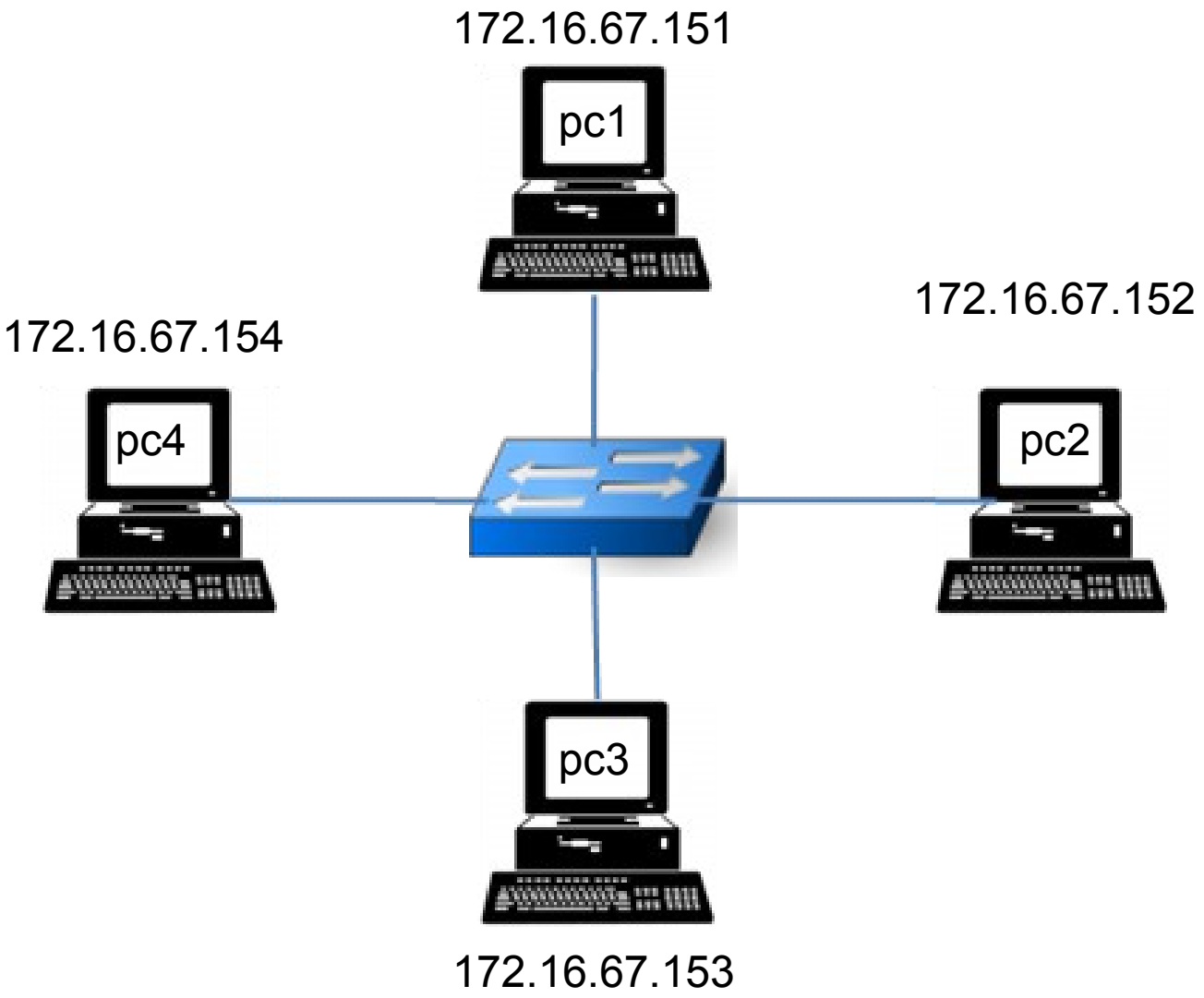
Nội dung

- Cài đặt Linux PC cluster
- Biên dịch và thực thi
- Các ví dụ minh họa

Nội dung

- **Cài đặt Linux PC cluster**
- Biên dịch và thực thi
- Các ví dụ minh họa

Cài đặt Linux PC cluster



Cài đặt Linux PC cluster

- Cài đặt Fedora Core 15 Linux cho các PC
  - Download Fedora từ **http://fedoraproject.org/**
  - Nhớ chọn bản thích hợp cho cấu hình 32, 64 bits
  - Ghi phân phối vào đĩa DVD
  - Cài đặt Fedora từ from DVD
  - Tạo phân vùng chính (15 GB) cho root (/), phân vùng swap (kích thước bằng 2 lần dung lượng RAM) và phân vùng cho **/home** kích thước còn lại
  - Nhớ chọn option (**Software development**) để có đủ các công cụ lập trình (gcc, g++, g77, etc)
  - Cài hết cho tất cả các máy PC

Cài đặt Linux PC cluster

- Cấu hình mạng cho các PC
  - Cấu hình địa chỉ mạng như mô tả của các máy tính
  - Máy pc1.cluster.cit có địa chỉ 172.16.67.151 (255.255.255.0)
  - Máy pc1.cluster.cit được dùng làm NFS server
  - Các máy pc1, pc2, pc3, pc4 nối kết nhau tạo thành cluster
  - Máy pc2.cluster.cit có địa chỉ 172.16.67.152 (255.255.255.0)
  - Máy pc3.cluster.cit có địa chỉ 172.16.67.153 (255.255.255.0)
  - Máy pc4.cluster.cit có địa chỉ 172.16.67.154 (255.255.255.0)

Cài đặt Linux PC cluster

- Cấu hình mạng cho các PC
  - Khởi động lần đầu (chạy cấu hình)
  - Tạo người dùng (ví dụ tên **user**)
  - Mở console kiểm tra liên kết giữa các PC (dùng lệnh **ping**)
  - Vào người dùng **root** để cấu hình hệ thống
  - Tắt firewall bằng các lệnh sau:

```
sed -i.bak "s/SELINUX=enforcing/SELINUX=permissive/g" /etc/selinux/config
/sbin/chkconfig --del iptables
service iptables stop
```

Cài đặt Linux PC cluster

- Cấu hình mạng cho các PC
  - Từ người dùng **root**
  - Soạn thảo tập tin **/etc/hosts** thêm các dòng sau:

```
172.16.67.151    pc1.cluster.cit    pc1
172.16.67.152    pc2.cluster.cit    pc2
172.16.67.153    pc3.cluster.cit    pc3
172.16.67.154    pc4.cluster.cit    pc4
```

  - Kiểm tra tên bằng cách dùng lệnh **ping** từ **pc1** sang bất kỳ pc nào chỉ dùng tên, không dùng địa chỉ
  - Soạn thảo tập tin **/etc/sysconfig/network**, bỏ tất cả domainname (**cluster.cit**) từ **HOSTNAME**, sau đó thực hiện:

```
source /etc/sysconfig/network
hostname $HOSTNAME
```

Cài đặt Linux PC cluster

- Cấu hình **SSH** cho các PC
  - Từ người dùng **user** trên **pc1**
  - Sinh khóa

```
ssh-keygen -t dsa -f ~/.ssh/id_dsa -N ""
cp .ssh/id_dsa.pub .ssh/authorized_keys
```

  - Chép đến tất cả các máy còn lại

```
scp -r .ssh pc2:
scp -r .ssh pc3:
scp -r .ssh pc4:
```

  - Kiểm tra bằng cách ssh vào người dùng user trên các PC (lần đầu tiên nhập passwd, từ lần thứ 2 trở đi không hỏi passwd)

```
ssh user@pc1
ssh user@pc2
ssh user@pc3
ssh user@pc4
```

Cài đặt Linux PC cluster

- Cấu hình NFS server
  - Cấu hình cho **pc1** là NFS server
  - Từ người dùng **root** trên **pc1**, tạo thư mục **/pub**

    mkdir /pub

    chmod 777 /pub
  - Soạn thảo tập tin **/etc/exports** thêm dòng sau:

    /pub 172.16.67.0/255.255.255.0(rw,no_root_squash)
  - Khởi động dịch vụ NFS

    chkconfig --level 345 nfs on

    service nfs restart

Cài đặt Linux PC cluster

- Cấu hình NFS client
  - Mount đến NFS server, từ người dùng **root**

```
mkdir  /home/user/cluster
chown  user.user  /home/user/cluster
mount –t nfs pc1:/pub      /home/user/cluster
```

  - Soạn thảo tập tin **/etc/fstab** thêm dòng

```
pc1:/pub          /home/user/cluster      nfs      defaults      0 0
```

  - Thực hiện mount

```
mount –a
```

  - Các **user** trên các PC sử dụng thư mục **pc1:/pub** thông qua **/home/user/cluster**
  - Các chương trình, mpich, đều được lưu vào **/home/user/cluster**

Nội dung

- Cài đặt Linux PC cluster
- **Biên dịch và thực thi**
- Các ví dụ minh họa

Cài đặt MPICH

- Cài đặt thư viện **mpich-2**
  - Download mpich-2 từ địa chỉ **http://www.mcs.anl.gov/research/projects/mpich2**
  - Chọn bản 1.3.2 (**mpich2-1.3.2.tar.gz**)
  - Giải nén vào thư mục **/home/user/cluster/mpich**
  - Cấu hình, biên dịch

    cd /home/user/cluster/mpich

    ./configure

    make
  - Cấu trúc thư mục của **mpich**

    mpich/bin (tập tin thực thi: mpicc, mpicxx, mpif77, mpirun, mpiexec, etc)

    mpich/include (tập tin header của mpich)

    mpich/lib (tập tin thư viện của mpich)

Sử dụng MPICH

- Sử dụng thư viện **mpich-2**
  - Biên dịch chương trình C

    mpicc -o prog.exec srcfile.c
  - Biên dịch chương trình C++

    mpicxx -o prog.exec srcfile.cc
  - Biên dịch chương trình Fortran

    mpif77 -o prog.exec srcfile.f

Sử dụng MPICH

- Thực thi chương trình
  - Tạo **machinefile** đặt tên là **hosts.txt** như sau

```
#tên-máy:số-lượng-core
pc1:2
pc2:1
pc3:1
pc4:1
```

  - Thực thi chương trình **prog.exec** sử dụng **10** processes

```
mpirun  -machinefile hosts.txt  -np 10   prog.exec
```

Nội dung

- Cài đặt Linux PC cluster
- Biên dịch và thực thi
- **Các ví dụ minh họa**

Các ví dụ minh họa

- Hello world từ các procs
- Tính toán số PI
- Giải thuật bubble sort
- Giải thuật merge sort
- Nhân ma trận
- Etc.

Cấu trúc chương trình MPI

```
#include <mpi.h>

int main(int argc, char ** argv) {

    MPI_Init(&argc, &argv);



    MPI_Finalize();
    return 0;
}
```

Các hàm của MPI (1)

- 6 hàm cơ bản
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_rank
  - MPI_Comm_size
  - MPI_Send
  - MPI_Recv
- 125 hàm khác
  - MPI_Bcast, MPI_Reduce, MPI_Barrier, etc

Các hàm của MPI (2)

- int MPI_Init(int* argc, char*** argv);
  - Tất cả tiến trình đều gọi MPI_Init
  - Khởi động thực thi song song
  - Truyền các tham số dòng lệnh
  - Thành công: MPI_SUCCESS

- int MPI_Finalize(void);
  - Tất cả tiến trình đều gọi MPI_Finalize
  - Kết thúc thực thi song song
  - Thành công: MPI_SUCCESS

Các hàm của MPI (3)

- int MPI_Comm_size(MPI_Comm comm, int* size);
  - Trả về tổng số tiến trình
- int MPI_Comm_rank(MPI_Comm comm, int* rank);
  - Trả về số hiệu tiến trình hiện hành

Các hàm của MPI (4)
Gửi nhận dữ liệu

- int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm);
  - Gửi **count** phần tử dữ liệu trong **buf** có kiểu **datatype** đến tiến trình **dest**

- int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int src, int tag, MPI_Comm comm, MPI_Status *status);
  - Nhận **count** phần tử dữ liệu có kiểu **datatype** từ tiến trình **src** đặt vào trong **buf**

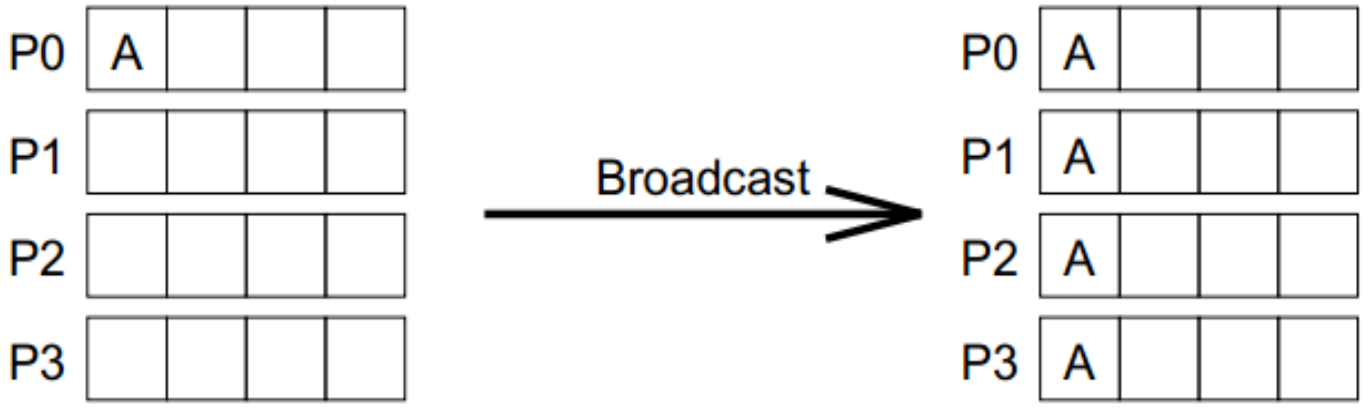
Các hàm của MPI (5)
Gửi nhận dữ liệu

MPI Datatype	C Datatype
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Các hàm của MPI (6)
Gửi nhận dữ liệu (collective)

- int MPI_Barrier(MPI_Comm comm);
  - Đồng bộ các tiến trình

- int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int src, MPI_Comm comm);
  - Tiến trình **src** gửi **count** phần tử dữ liệu trong **buf** có kiểu **datatype** đến tất cả các tiến trình



Các hàm của MPI (7)
Gửi nhận dữ liệu (collective)

- int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int target, MPI_Comm comm);
  - Thực hiện phép toán tổng hợp **op** của **count** phần tử dữ liệu có kiểu **datatype**, gửi từ các tiến trình đến **target**

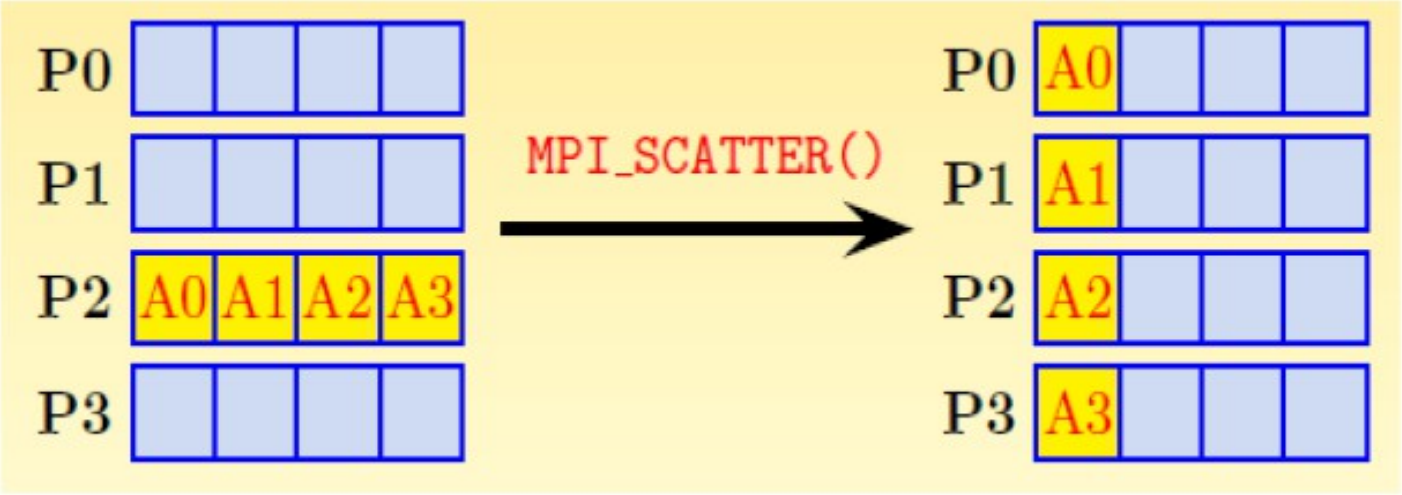
- int MPI_Allreduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
  - MPI_Reduce + MPI_Bcast

Các hàm của MPI (8)
Phép toán của MPI_Reduce (collective)

Operation	Meaning	Datatypes
MPI_MAX	Maximum	C integers and floating point
MPI_MIN	Minimum	C integers and floating point
MPI_SUM	Sum	C integers and floating point
MPI_PROD	Product	C integers and floating point
MPI_LAND	Logical AND	C integers
MPI_BAND	Bit-wise AND	C integers and byte
MPI_LOR	Logical OR	C integers
MPI_BOR	Bit-wise OR	C integers and byte
MPI_LXOR	Logical XOR	C integers
MPI_BXOR	Bit-wise XOR	C integers and byte
MPI_MAXLOC	max-min value-location	Data-pairs
MPI_MINLOC	min-min value-location	Data-pairs

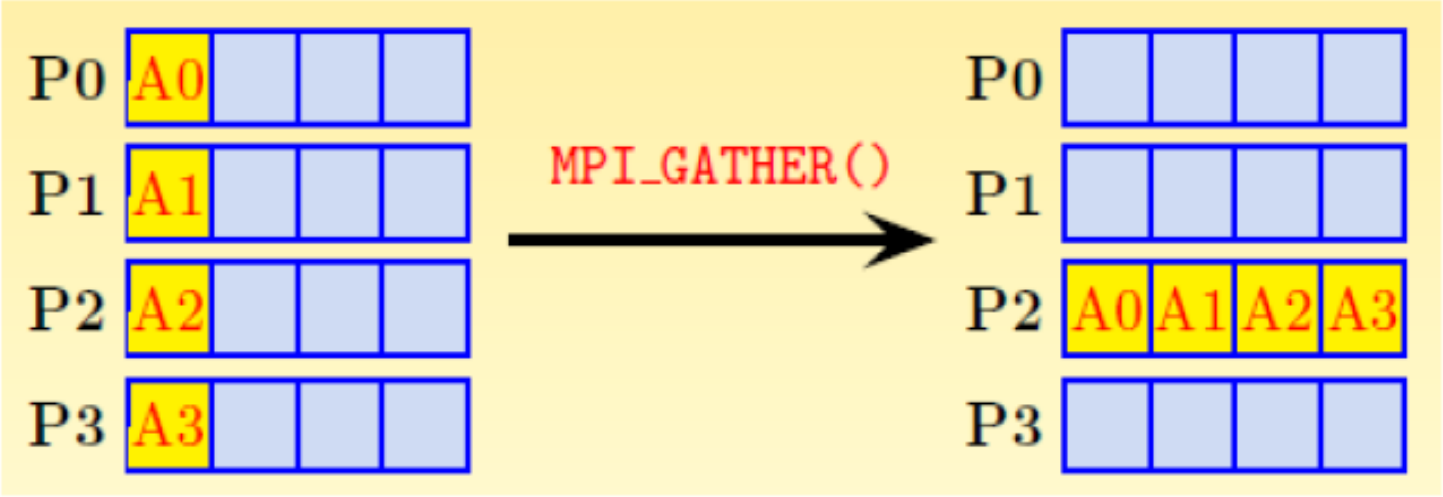
Các hàm của MPI (9)
Gửi nhận dữ liệu (collective)

- int MPI_Scatter(void *sendbuf, int sendcount,
  MPI_Datatype senddatatype, void *recvbuf,
  int recvcount, MPI_Datatype recvdatatype,
  int src, MPI_Comm comm);



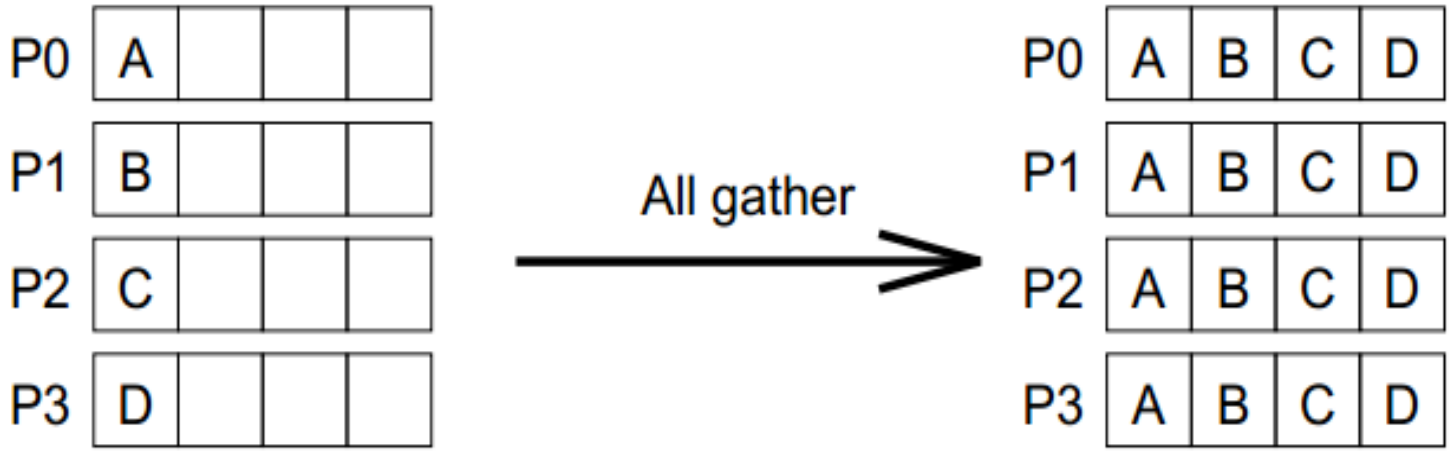
Các hàm của MPI (10)
Gửi nhận dữ liệu (collective)

- int MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, int target, MPI_Comm comm);



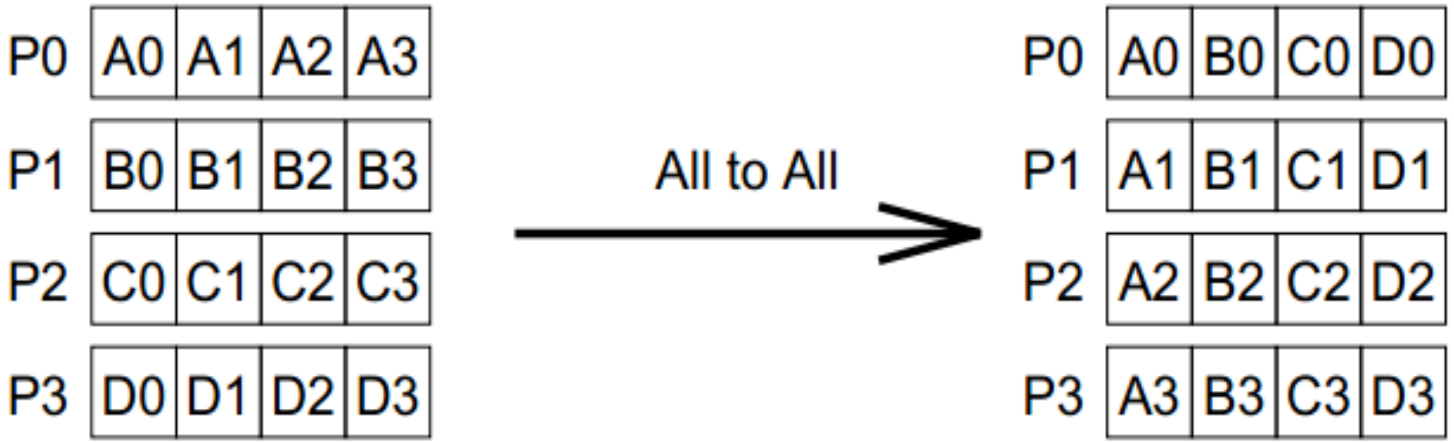
Các hàm của MPI (11)
Gửi nhận dữ liệu (collective)

- int MPI_Allgather(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, MPI_Comm comm);



Các hàm của MPI (12)
Gửi nhận dữ liệu (collective)

- int MPI_Alltoall(void *sendbuf, int sendcount, MPI_Datatype senddatatype, void *recvbuf, int recvcount, MPI_Datatype recvdatatype, MPI_Comm comm);



Chương trình hello world

- Minh họa chương trình đơn giản
  - Mỗi tiến trình gửi thông điệp

    I am ... of ...
- Sử dụng
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_rank
  - MPI_Comm_size

Chương trình hello.c

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

int main(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("I am %d of %d\n", rank, size);

    MPI_Finalize();
    return 0;
}
```

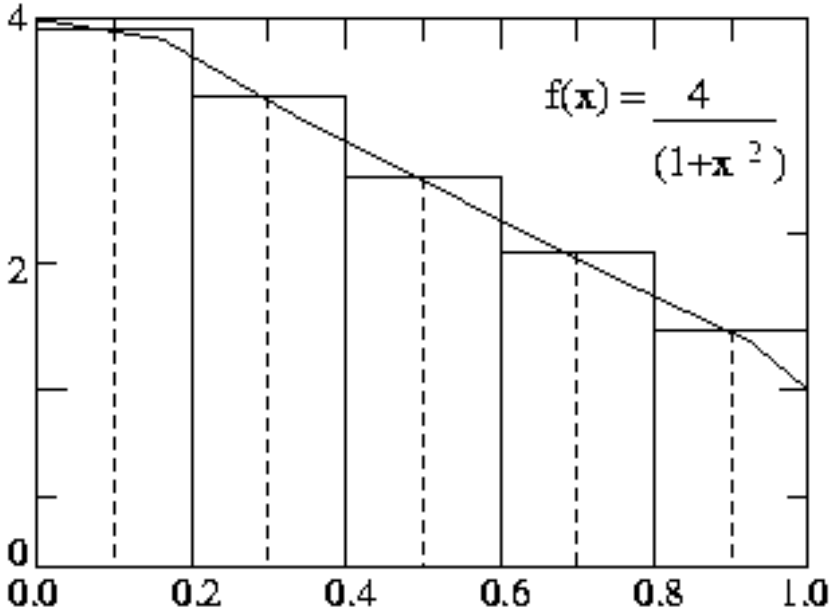
Tính số PI

$$\int_0^1 \frac{1}{1+x^2}\,dx = \arctan(x)\Big|_0^1$$

$$= \arctan(1) - \arctan(0)$$

$$= \arctan(1)$$

$$= \frac{\pi}{4}$$



Chương trình cpi.c (1)

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

double f(double);

double f(double a) {
    return (4.0 / (1.0 + a*a));
}

int main(int argc,char *argv[]) {
    int    n, myid, numprocs, i, namelen;
    double PI25DT = 3.14159265358979323846264643;
    double mypi, pi, h, sum, x;
    double startwtime, endwtime;
    char   processor_name[MPI_MAX_PROCESSOR_NAME];
```

Chương trình cpi.c (2)

```
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myid);
MPI_Get_processor_name(processor_name,&namelen);

fprintf(stdout,"Process %d of %d is on %s\n",
        myid, numprocs, processor_name);
fflush(stdout);

n = 10000;                    /* default # of rectangles */
```

Chương trình cpi.c (3)

```
if (myid == 0)
    startwtime = MPI_Wtime();

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

h   = 1.0 / (double) n;
sum = 0.0;
/* A slightly better approach starts from large i and works back */
for (i = myid + 1; i <= n; i += numprocs) {
    x = h * ((double)i - 0.5);
    sum += f(x);
}
mypi = h * sum;

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

Chương trình cpi.c (4)

```
    if (myid == 0) {
        endwtime = MPI_Wtime();
        printf("pi is approximately %.16f, Error is %.16f\n",
               pi, fabs(pi - PI25DT));
        printf("wall clock time = %f\n", endwtime-startwtime);
        fflush(stdout);
    }

    MPI_Finalize();
    return 0;
}
```

Giải thuật tuần tự bubble sort

```
1.    procedure BUBBLE_SORT(n)
2.    begin
3.        for i := n - 1 downto 1 do
4.            for j := 1 to i do
5.                compare-exchange(a_j, a_{j+1});
6.    end BUBBLE_SORT
```

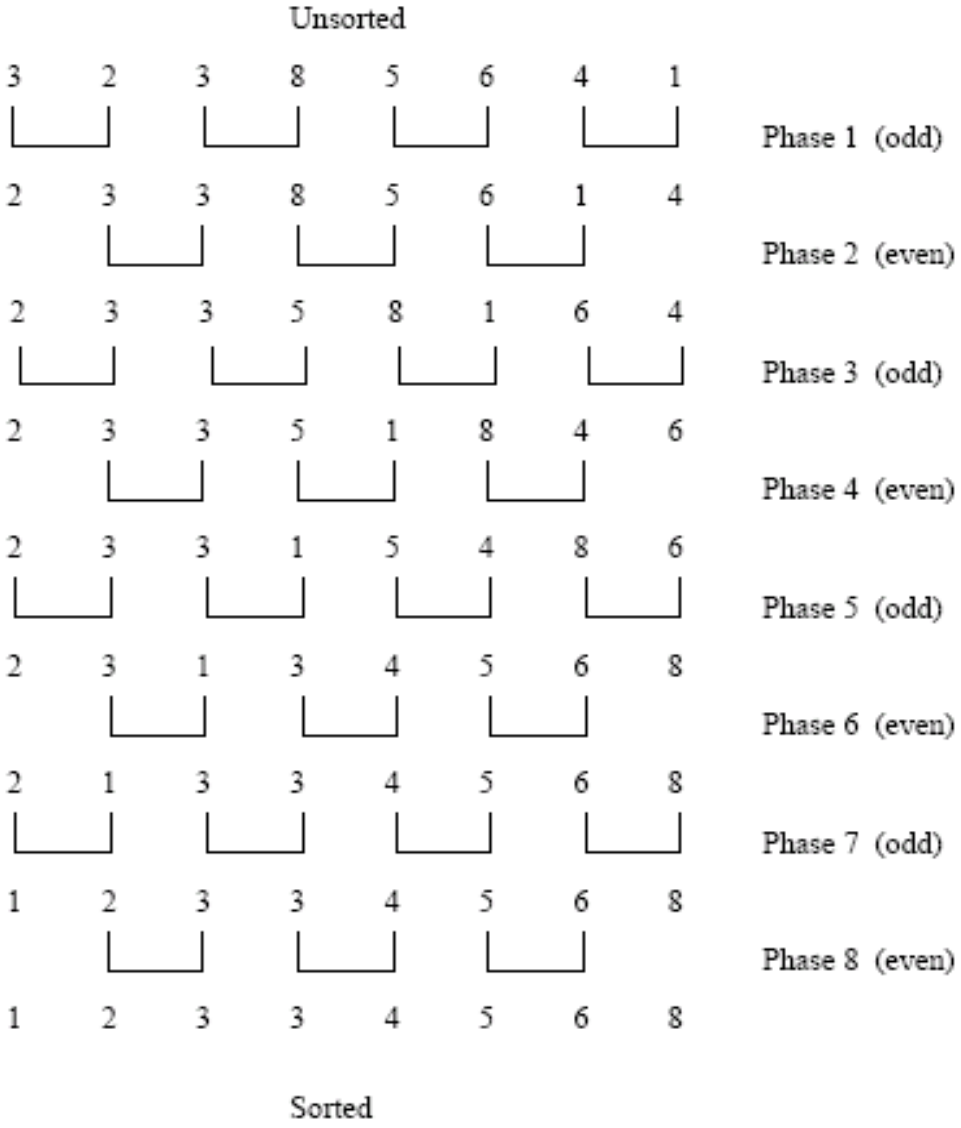
- Độ phức tạp O(n^2)
- Giải thuật tuần tự
- Biến thể của bubblesort: odd-even transposition => dễ song song

Giải thuật tuần tự Odd-even transposition sort

```
1.   procedure ODD-EVEN(n)
2.   begin
3.       for i := 1 to n do
4.       begin
5.           if i is odd then
6.               for j := 0 to n/2 − 1 do
7.                   compare-exchange(a_{2j+1}, a_{2j+2});
8.           if i is even then
9.               for j := 1 to n/2 − 1 do
10.                  compare-exchange(a_{2j}, a_{2j+1});
11.      end for
12.  end ODD-EVEN
```

Ví dụ về Odd-even transposition sort

- Cài đặt PC cluster
- Biên dịch và thực thi
- **Các ví dụ minh họa**



Giải thuật song song Odd-even transposition sort

```
1.    procedure ODD-EVEN_PAR(n)
2.    begin
3.        id := process's label
4.        for i := 1 to n do
5.        begin
6.            if i is odd then
7.                if id is odd then
8.                    compare-exchange_min(id + 1);
9.                else
10.                   compare-exchange_max(id − 1);
11.           if i is even then
12.               if id is even then
13.                   compare-exchange_min(id + 1);
14.               else
15.                   compare-exchange_max(id − 1);
16.       end for
17.   end ODD-EVEN_PAR
```

Giải thuật song song Odd-even transposition sort

Even Phase

```
P_i, i = 0, 2, 4, …, n - 2 (even)        P_i, i = 1, 3, 5, …, n - 1 (odd)

recv(&A, P_i+1);                         send(&A, P_i-1);  /* even phase */
send(&B, P_i+1);                         recv(&B, P_i-1);
if (A > B) B = A;                        if (A > B) A = B;/* exchange */
```

where the number stored in $P_{i\ (\text{even})}$ is B and the number stored in $P_{i\ (\text{odd})}$ is A.

Odd Phase

```
P_i, i = 1, 3, 5, …, n - 3 (odd)         P_i, i = 2, 4, 6, …, n - 2 (even)

send(&A, P_i+1);                         recv(&A, P_i-1);    /* odd phase */
recv(&B, P_i+1);                         send(&B, P_i-1);
if (A > B) A = B;                        if (A > B) B = A; /* exchange */
```

Giải thuật song song Odd-even transposition sort

Combined

P_i, i = 1, 3, 5, …, n - 3 (odd)

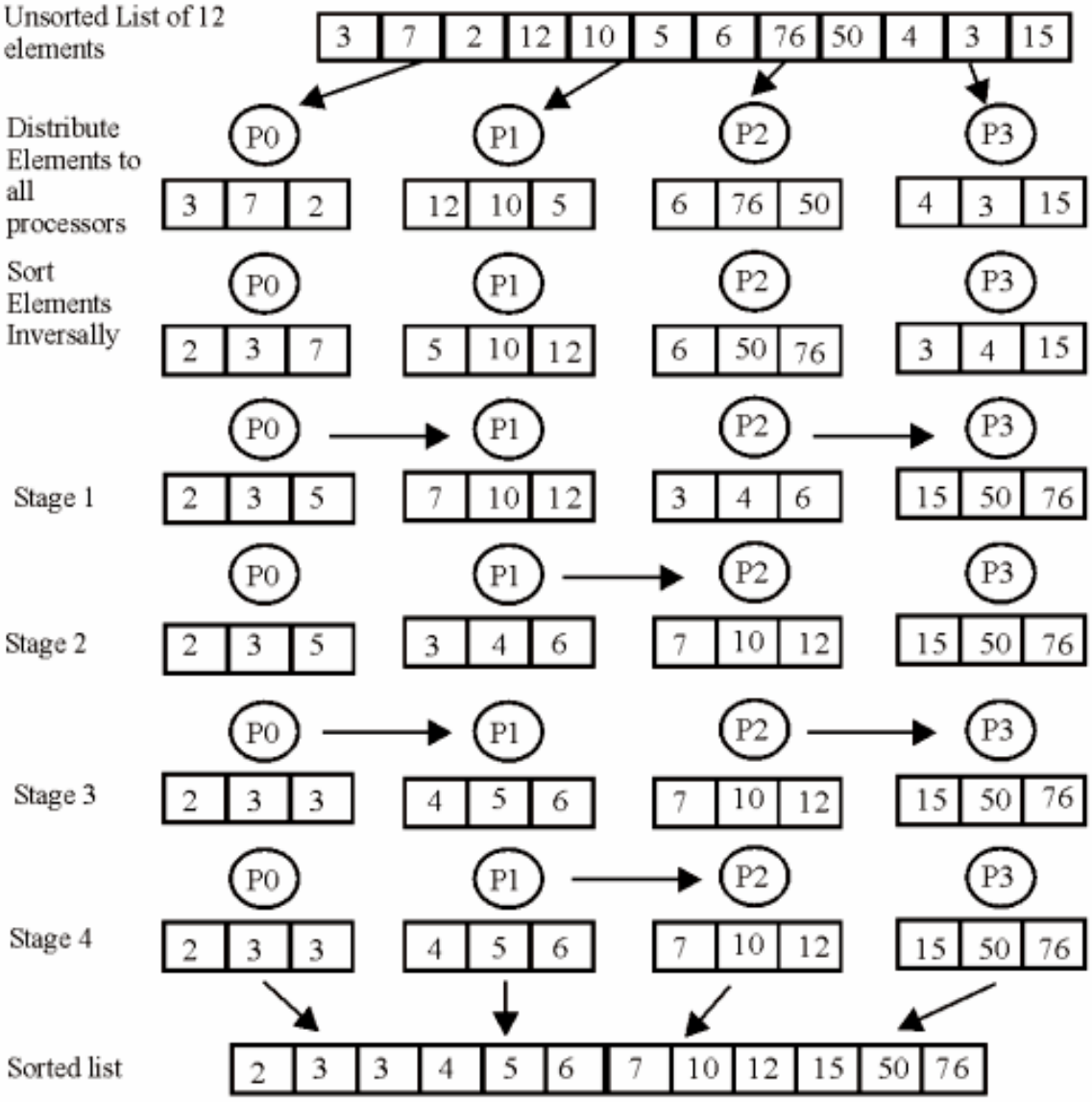
```
send(&A, P_{i-1});
recv(&B, P_{i-1});
if (A > B) A = B;
if (i <= n-3) {
    send(&A, P_{i+1});
    recv(&B, P_{i+1})
    if (A > B) A = B;
}
```

P_i, i = 0, 2, 4, …, n - 2 (even)

```
recv(&A, P_{i+1});  /* even phase */
send(&B, P_{i+1});
if (A > B) B = A;
if (i >= 2) {   /* odd phase */
  recv(&A, P_{i-1});
  send(&B, P_{i-1});
  if (A > B) B = A;
}
```

Giải thuật song song (4 proc.) Odd-even transposition sort

- Cài đặt PC cluster
- Biên dịch và thực thi
- **Các ví dụ minh họa**



Chtrình odd-even.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <mpi.h>

#define N 128000

/* compare function */
int cmpfunc (const void * a, const void * b) {
   return (*(int*)a - *(int*)b);
}

/* qsort array v; array is of length n */
void seqsort(int * v, int n) {
      qsort(v, n, sizeof(int), cmpfunc);
}
```

Chtrình odd-even.c (2)

```
void swap(int *x, int *y) {
  int temp = *x;
  *x = *y;
  *y = temp;
}

int main(int argc, char **argv) {
  int n = N;      /* The total number of elements to be sorted */
  int *arr;       /* The array of elements to be sorted */
  int *value;
  double elapsed_time;
  int i, j, ph, nlocal;
  FILE *file;
  MPI_Status status;
  int rank;       /* The rank of the calling process */
  int size;       /* The total number of processes */
```

Chtrình odd-even.c (3)

```
if (argc!=2) {
  fprintf(stderr, "Usage: mpirun -np <num_procs> %s <out_file>\n", argv[0]);
  exit(1);
}
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);

arr  = (int *)malloc(n*sizeof(int));
value = (int *)malloc(n*sizeof(int));

if(rank==0) {
      elapsed_time = - MPI_Wtime();
      srand(time(0));
      for(i=0; i<n; i++)
         arr[i]=random();
}
```

Chtrình odd-even.c (4)

```
MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
nlocal = n/size;

/* Send data */
MPI_Scatter(arr, nlocal, MPI_INT, value, nlocal, MPI_INT, 0,
                MPI_COMM_WORLD);

for(ph=0; ph<size; ph++) {
 if(ph%2 == 0) {                    /* even phase */
  if(rank%2 == 0) {                 /* even rank */
    MPI_Send(&value[0], nlocal, MPI_INT, rank+1, 0, MPI_COMM_WORLD);
    MPI_Recv(&value[nlocal], nlocal, MPI_INT, rank+1, 0,
                MPI_COMM_WORLD,&status);
    seqsort(value, nlocal*2);
  }
```

Chtrình odd-even.c (5)

```
  else {                    /* odd rank */
    MPI_Recv(&value[nlocal], nlocal, MPI_INT, rank-1, 0,
                    MPI_COMM_WORLD, &status);
    MPI_Send(&value[0], nlocal, MPI_INT, rank-1, 0, MPI_COMM_WORLD);
    seqsort(value, nlocal*2);
    for(i=0;i<nlocal;i++)
      swap(&value[i],&value[i+nlocal]);
  }
}
else {                      /* odd phase */
  if((rank%2 == 1) && (rank != (size-1))) {        /* odd rank */
    MPI_Send(&value[0], nlocal, MPI_INT, rank+1, 0, MPI_COMM_WORLD);
    MPI_Recv(&value[nlocal], nlocal, MPI_INT, rank+1, 0,
                    MPI_COMM_WORLD,&status);
    seqsort(value, nlocal*2);
  }
```

Chtrình odd-even.c (6)

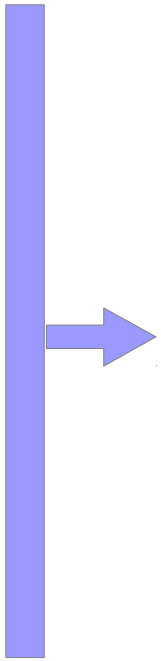
```
  else if(rank != 0 && rank != (size-1)) {          /* even rank */
    MPI_Recv(&value[nlocal], nlocal, MPI_INT, rank-1, 0,
                  MPI_COMM_WORLD,&status);
    MPI_Send(&value[0], 1, MPI_INT, rank-1, 0, MPI_COMM_WORLD);
    seqsort(value, nlocal*2);
    for(i=0;i<nlocal;i++)
      swap(&value[i],&value[i+nlocal]);
  }
 }
}

MPI_Gather(value, nlocal, MPI_INT, arr, nlocal, MPI_INT, 0,
                  MPI_COMM_WORLD);
```

Chtrình odd-even.c (7)

```
  if(rank==0) {
    elapsed_time += MPI_Wtime();

    file = fopen(argv[1], "w");
    for (i = 0; i < n; i++)
      fprintf(file, "%d\n", arr[i]);
    fclose(file);
    printf("odd-even sort %d ints on %d procs: %f secs\n", n, size, elapsed_time);
  }

  MPI_Finalize();

  return 0;
}
```

Chtrình bsort.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <mpi.h>

#define N 100000

void swap(int * v, int i, int j) {
  int t = v[i]; v[i] = v[j]; v[j] = t;
}

void bubblesort(int * v, int n) {
  int i, j;
  for (i = n-2; i >= 0; i--)
    for (j = 0; j <= i; j++)
      if (v[j] > v[j+1]) swap(v, j, j+1);
}
```



```
/* compare function */
int cmpfunc(const void * a, const void * b) {
  return (*(int*)a - *(int*)b);
}

/* qsort array v; array is of length n */
void seqsort(int * v, int n) {
  qsort(v, n, sizeof(int), cmpfunc);
}
```

Chtrình bsort.c (2)

```
/* merge two sorted arrays v1, v2 of lengths n1, n2, respectively */
int * merge(int * v1, int n1, int * v2, int n2) {
  int * result = (int *)malloc((n1 + n2) * sizeof(int));
  int i = 0, j = 0, k;
  for (k = 0; k < n1 + n2; k++)  {
     if (i >= n1) {
        result[k] = v2[j]; j++;
     } else if (j >= n2) {
        result[k] = v1[i]; i++;
     } else if (v1[i] < v2[j]) { // indices in bounds as i < n1 && j < n2
        result[k] = v1[i]; i++;
     } else {
        result[k] = v2[j]; j++;
     }
  } // for
  return result;
}
```

Chtrình bsort.c (3)

```
int main(int argc, char ** argv) {
  int n = N;
  int *data = NULL;
  int c, s, o, step, p, id, i;
  int *chunk;
  int *other;
  MPI_Status status;
  double elapsed_time;
  FILE *file;
  if (argc!=2) {
    fprintf(stderr, "Usage: mpirun -np <num_procs> %s <out_file>\n", argv[0]);
    exit(1);
  }

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &p);
  MPI_Comm_rank(MPI_COMM_WORLD, &id);
```

Chtrình bsort.c (4)

```
if (id == 0) {
  struct timeval tp;
  // compute chunk size
  c = n/p; if (n%p) c++;
  // generate data
  gettimeofday(&tp, 0);
  srandom(tp.tv_sec + tp.tv_usec);
  srand(time(NULL));
  data = (int *)malloc(p*c*sizeof(int));
  for(i=0;i<n;i++)
     data[i] = random();

  for (i = n; i < p*c; i++)  // pad data with 0 -- doesn't matter
     data[i] = 0;

  elapsed_time = - MPI_Wtime();  // start the timer
}
```

Chtrình bsort.c (5)

```
MPI_Barrier(MPI_COMM_WORLD); // Barrier synchronization

MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);   // broadcast size

c = n/p; if (n%p) c++;   // compute chunk size

// scatter data
chunk = (int *)malloc(c * sizeof(int));
MPI_Scatter(data, c, MPI_INT, chunk, c, MPI_INT, 0, MPI_COMM_WORLD);
free(data);
data = NULL;

// compute size of own chunk and sort it
s = (n >= c * (id+1)) ? c : n - c * id;
bubblesort(chunk, s);
```

Chtrình bsort.c (6)

```
// up to log_2 p merge steps
for (step = 1; step < p; step = 2*step) {
  if (id % (2*step)!=0) {
    // id is no multiple of 2*step: send chunk to id-step and exit loop
    MPI_Send(chunk, s, MPI_INT, id-step, 0, MPI_COMM_WORLD); break;
  }
  // id is multiple of 2*step: merge in chunk from id+step (if it exists)
  if (id+step < p) {
    o = (n >= c * (id+2*step)) ? c * step : n - c * (id+step); // compute chunk size
    other = (int *)malloc(o * sizeof(int)); // receive other chunk
    MPI_Recv(other, o, MPI_INT, id+step, 0, MPI_COMM_WORLD, &status);
    // merge and free memory
    data = merge(chunk, s, other, o); free(chunk); free(other);
    chunk = data;
    s = s + o;
  } // if
} // for
```

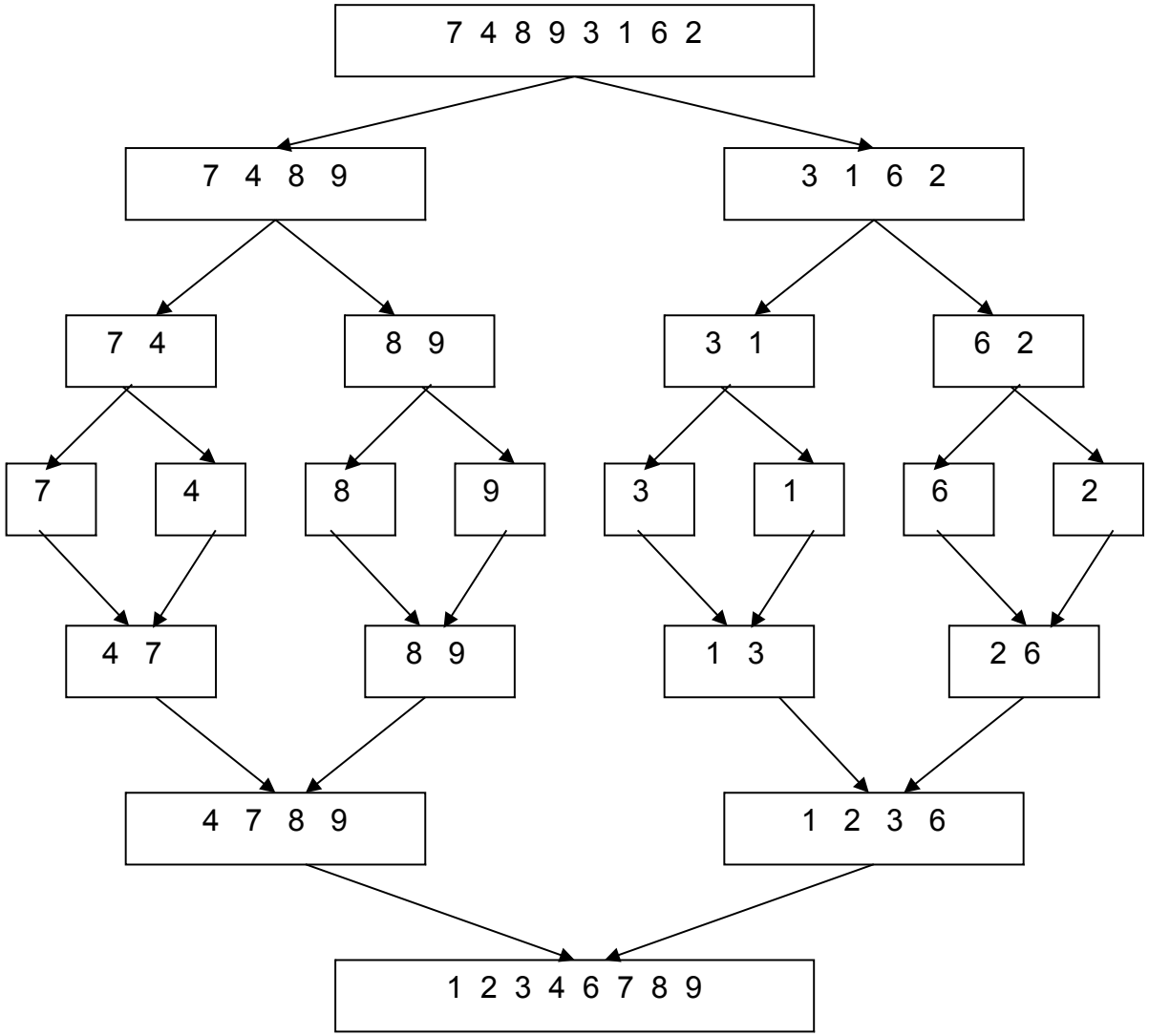
Chtrình bsort.c (7)

```
  // write sorted data to out file and print out timer
  if (id == 0) {
    elapsed_time += MPI_Wtime();  // stop the timer

    file = fopen(argv[1], "w");
    for (i = (s-n); i < s; i++)
      fprintf(file, "%d\n", chunk[i]);
    fclose(file);
    printf("Bubblesort %d ints on %d procs: %f secs\n", n, p, elapsed_time);
  }

  MPI_Finalize();
  return 0;
} // main
```

Giải thuật tuần tự Merge sort

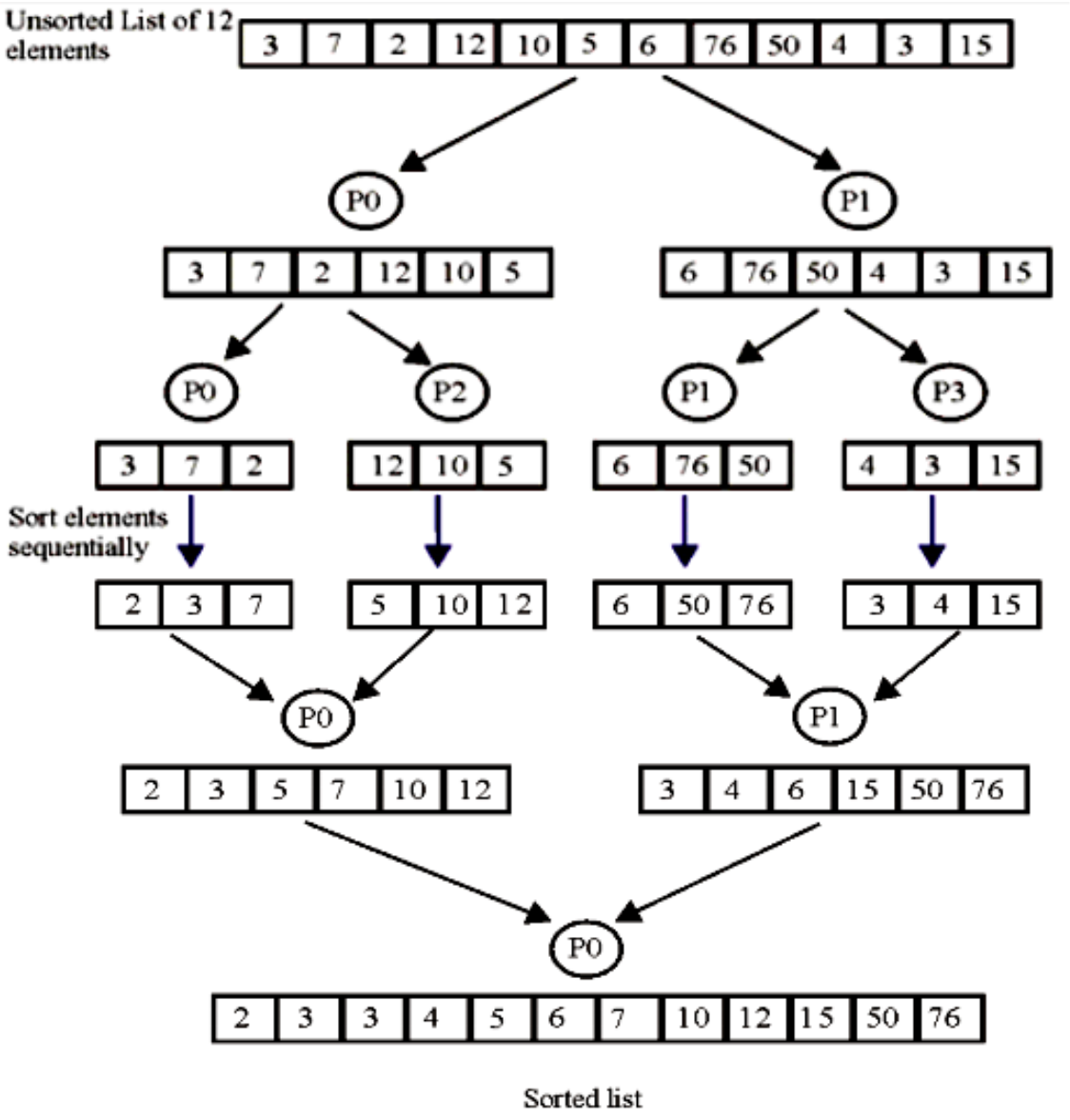
```
1   procedure MERGE-SORT(l, h)
2   begin
3       if (l < h) then
4       begin
5           m = (l + h)/2;
6           MERGE-SORT(l, m);
7           MERGE-SORT(m+1, h);
8           MERGE(l, m, h);
9        end
10  end MERGE-SORT
```

Ví dụ về Merge sort



Giải thuật song song (4 proc.) Merge sort

- Cài đặt PC cluster
- Biên dịch và thực thi
- **Các ví dụ minh họa**



Chtrình merge-sort.c (1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <mpi.h>

#define N 100000

int* merge(int *A, int asize, int *B, int bsize);
void swap(int *v, int i, int j);
void m_sort(int *A, int min, int max);

void swap(int * v, int i, int j) {
  int t = v[i]; v[i] = v[j]; v[j] = t;
}
```

Chtrình merge-sort.c (2)

```
int* merge(int *A, int asize, int *B, int bsize) {
        int ai = 0, bi = 0, ci = 0, i;
        int csize = asize+bsize;
        int *C = (int *)malloc(csize*sizeof(int));
        while ((ai < asize) && (bi < bsize)) {
                if (A[ai] <= B[bi])  { C[ci] = A[ai]; ci++; ai++; }
                else { C[ci] = B[bi]; ci++; bi++; }
        }
        if (ai >= asize)
                for (i = ci; i < csize; i++, bi++)  C[i] = B[bi];
        else if (bi >= bsize)
                for (i = ci; i < csize; i++, ai++)  C[i] = A[ai];

        for (i = 0; i < asize; i++) A[i] = C[i];
        for (i = 0; i < bsize; i++) B[i] = C[asize+i];
        return C;
}
```

Chtrình merge-sort.c (3)

```
void  m_sort(int *A, int min, int max) {
      int *C;              /* dummy, just to fit the function */
      int mid = (min+max)/2;
      int lowerCount = mid - min + 1;
      int upperCount = max - mid;
      /* If the range consists of a single element, it's already sorted */
      if (max == min) {
            return;
      } else {
            /* Otherwise, sort the first half */
            m_sort(A, min, mid);
            /* Now sort the second half */
            m_sort(A, mid+1, max);
            /* Now merge the two halves */
            C = merge(A + min, lowerCount, A + mid + 1, upperCount);
      }
}
```

Chtrình merge-sort.c (4)

```
main(int argc, char **argv) {
        int n= N;
        int *data;
        int *chunk;
        int *other;
        int m, id, p, s = 0, i, step;
        MPI_Status status;
        double elapsed_time;
        FILE *file;
        if (argc!=2) {
          fprintf(stderr, "Usage: mpirun -np <num_procs> %s <out_file>\n", argv[0]);
          exit(1);
        }

        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&id);
        MPI_Comm_size(MPI_COMM_WORLD,&p);
```

Chtrình merge-sort.c (5)

```
if(id==0) {
        struct timeval tp;
        int r = n%p;
        s = n/p;
        // generate data
        gettimeofday(&tp, 0);
        srandom(tp.tv_sec + tp.tv_usec); srand(time(NULL));
        data = (int *)malloc((n+s-r)*sizeof(int));
        for(i=0;i<n;i++)
                data[i] = random();

        if(r!=0) {
                for(i=n;i<n+s-r;i++)  data[i]=0;
                s=s+1;
        }
```

Chtrình merge-sort.c (6)

```
        // start timer
        elapsed_time = - MPI_Wtime();

        MPI_Bcast(&s,1,MPI_INT,0,MPI_COMM_WORLD);
        chunk = (int *)malloc(s*sizeof(int));
        MPI_Scatter(data,s,MPI_INT,chunk,s,MPI_INT,0,
                                          MPI_COMM_WORLD);
        m_sort(chunk, 0, s-1);
        /* showVector(chunk, s, id); */
} else {
        MPI_Bcast(&s,1,MPI_INT,0,MPI_COMM_WORLD);
        chunk = (int *)malloc(s*sizeof(int));
        MPI_Scatter(data,s,MPI_INT,chunk,s,MPI_INT,0,
                                          MPI_COMM_WORLD);
        m_sort(chunk, 0, s-1);
}
```

Chtrình merge-sort.c (7)

```
step = 1;
while(step < p) {
  if(id%(2*step)==0) {
    if(id+step<p) {
      MPI_Recv(&m,1,MPI_INT,id+step,0,MPI_COMM_WORLD,&status);
      other = (int *)malloc(m*sizeof(int));
      MPI_Recv(other,m,MPI_INT,id+step,0,MPI_COMM_WORLD,&status);
      chunk = merge(chunk,s,other,m);
      s = s+m; }
   } else {
      int near = id-step;
      MPI_Send(&s,1,MPI_INT,near,0,MPI_COMM_WORLD);
      MPI_Send(chunk,s,MPI_INT,near,0,MPI_COMM_WORLD);
      break;
  }
  step = step*2;
}
```

Chtrình merge-sort.c (8)

```
    if(id==0) {
      // stop the timer
      elapsed_time += MPI_Wtime();
      file = fopen(argv[1], "w");
      for (i = (s-n); i < s; i++)
              fprintf(file, "%d\n", chunk[i]);
      fclose(file);
      printf("Mergesort %d ints on %d procs: %f secs\n", n, p, elapsed_time);
    }

    MPI_Finalize();
}
```

Tài liệu tham khảo

1. Argonne National Lab. MPICH2 : High-performance and Widely Portable MPI. 2012. http://www.mcs.anl.gov/research/projects/mpich2/.
2. P. Pacheco. *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011.
3. William Gropp. Tutorial on MPI: The Message-Passing Interface. 1995.
4. Đỗ Thanh Nghị, Nguyễn Văn Hòa, Đỗ Hiệp Thuận. *Lập trình song song*. NXB Đại học Cần thơ, 2014.

Cám ơn