Truyền thông điểm điểm trong lập trình song song truyền thông điệp

Truyền thông điểm-điểm

Nội dung bài học

- Khái niệm Tiến trình nguồn và Tiến trình đích
- Cấu trúc thông điệp
- Quá trình gửi/nhận thông điệp
- Gửi nhận thông điệp có ràng buộc
- Gửi nhận thông điệp không ràng buộc
- Các chế độ gửi

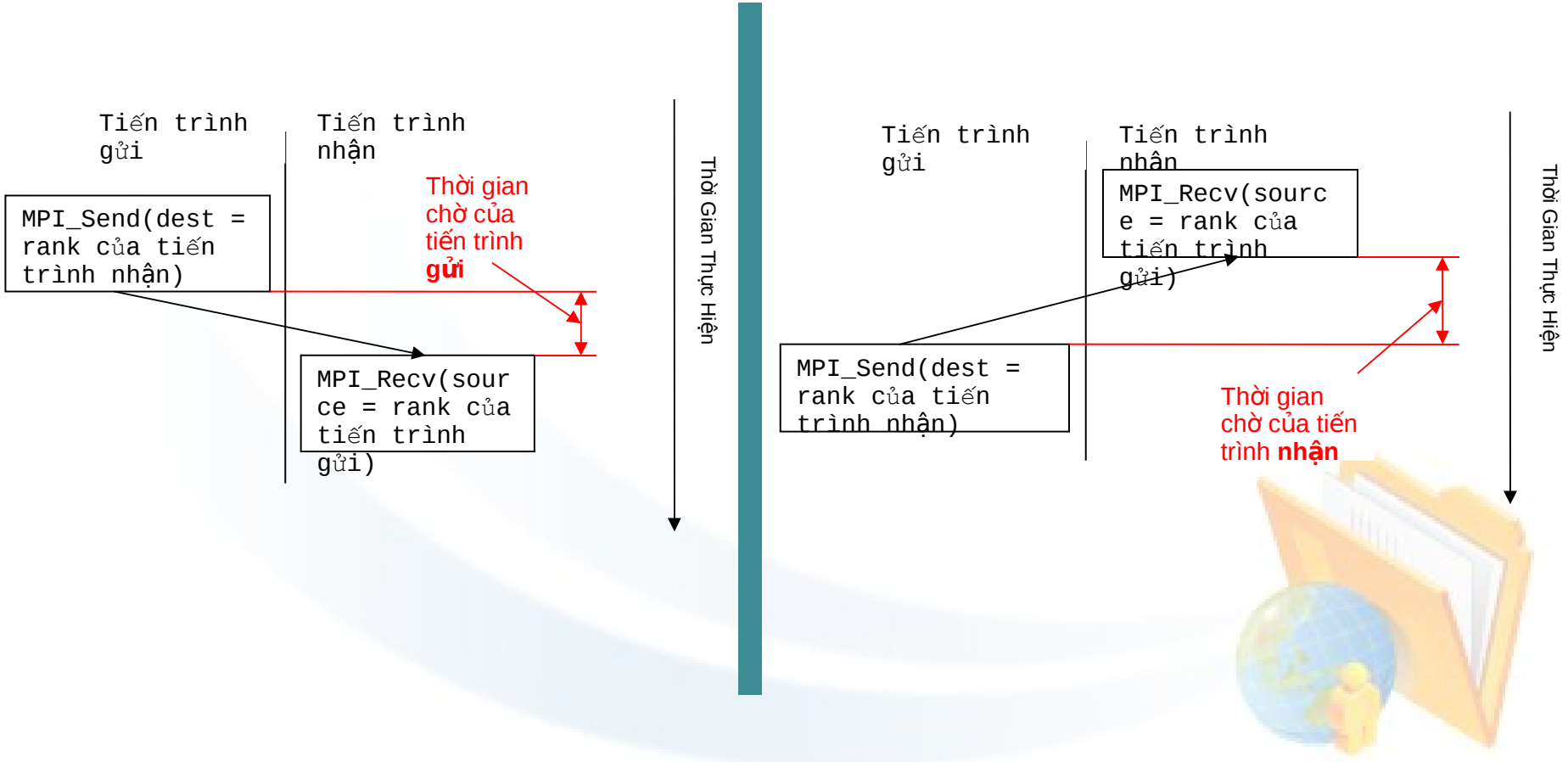
Truyền thông điểm-điểm

Tiến trình nguồn và tiến trình đích

- Truyền thông điểm điểm là truyền thông hai phía
  - Đòi hỏi khởi tạo quá trình truyền thông từ cả hai phía
  - Gồm một tiến trình gửi và một tiến trình nhận
  - Tiến trình gửi gọi là tiến trình nguồn
  - Tiến trình nhận gọi là tiến trình đích

Tiến trình nguồn và tiến trình đích

- Nói chung, tiến trình nguồn và tiến trình đích là không đồng bộ



Truyền thông điểm-điểm

Cấu trúc thông điệp

- Gồm 2 phần:
  - Thông tin về thông điệp
  - Nội dung thông điệp
- Thông tin về thông điệp gồm 4 phần:
  - Nguồn: tiến trình gửi
  - Đích: tiến trình nhận
  - Nhóm truyền thông: Nhóm tiến trình bao gồm cả nguồn và đích
  - Tag: Dùng để phân biệt với các thông điệp khác
    - Quan trọng
    - Cách dùng phụ thuộc từng chương trình.

Truyền thông điểm-điểm

Nội dung thông điệp

- Nội dung thông điệp gồm 3 phần:
  - Bộ đệm: Dữ liệu trao đổi
    - buffer
    - Thường là một mảng
  - Loại dữ liệu: Loại của dữ liệu trao đổi
    - datatype
    - Loại dữ liệu của mảng
  - Bộ đếm: Số lượng dữ liệu trong bộ đệm cần trao đổi
    - count: count * sizeof(dtype)
    - Số phần tử của mảng

int Array[count]


FOX
INTERACTIVE MEDIA

June 1, 2007

To Whom It May Concern:

I am writing in support of the application submitted by the TechMission Safe Families Initiative to the OJJDP FY 2007 National Juvenile Justice Programs.

The Safe Families program is a national online safety program for children and adolescents providing strategies to educate communities about the dangers children may encounter online, and strategies to keep our children safe. The Safe Families program has played an essential role in building online safety to protect our children from dangers on the Internet. The program has developed materials to equip parents, churches and community organizations to implement online safety programs. In addition, through a partnership with We Blocker, the Safe Families program provides free Internet filtering software and distributes this software to over 100,000 individuals and families.

I urge you to give this grant application every consideration so that the Safe Families program staff can continue to do the great work they have been doing throughout the nation. I thank you in advance for your assistance.

Sincerely,

Hernandez Nguen
Chief Security Officer
Fox Interactive Media

407 N. Maple Drive, Beverly Hills, CA 90210

Truyền thông điểm-điểm

Quá trình gửi nhận thông điệp

- Quá trình gửi thông điệp
  - Nguồn (định danh người gửi) là ngầm định
  - Thông tin và nội dung của thông điệp phải được khai báo rõ ràng bởi tiến trình gửi
- Thông điệp treo (a pending message)
  - Thông điệp đã gửi nhưng chưa được nhận
  - Không được lưu trong hàng đợi
  - Chứa các thuộc tính phục vụ quá trình nhận

Truyền thông điểm-điểm

Qúa trình gửi nhận thông điệp

- Quá trình nhận thông điệp
  - Chỉ định thông tin thông điệp để so khớp với các thông điệp treo
  - Thành công: thông điệp được nhận, quá trình nhận kết thúc
  - Không thành công: tiếp tục chờ đợi, so khớp, quá trình nhận chưa hoàn thành
  - Cung cấp đủ không gian lưu trữ cho nội dung thông điệp

Truyền thông điểm-điểm

Gửi nhận thông điệp có ràng buộc

- Dùng hai định tuyến (routine)
  - MPI_Send: gửi thông điệp
  - MPI_Recv: nhận thông điệp
- Cả hai định tuyến đều ràng buộc với tiến trình gọi nó
- Sự ràng buộc kết thúc khi quá trình truyền thông hoàn thành

- Khi nào hoàn thành truyền thông?
- Hiện tượng deadlock có tồn tại?

Truyền thông điểm-điểm

Định tuyến MPI_Send

buffer
count
datatype

} *Nội dung thông điệp*

destination
tag
communicator

} *Thông tin thông điệp (source được hiểu ngầm định)*

Truyền thông điểm-điểm

Định tuyến MPI_Send (tiếp)

```
MPI_Send(void* buf, int count, MPI_Datatype dtype, int dest,
int tag,    MPI_Comm comm)
```

Truyền thông điệp trong buf đến tiến trình dest

Giá trị trả về của hàm là mã lỗi

IN/OUT	Tên biến	Loại biến	Mô tả
IN	buf	biến con trỏ (không định kiểu)	trỏ đến vùng nhớ chứa dữ liệu
IN	count	int	số lượng phần tử trong dữ liệu
IN	dtype	MPI_Datatype	kiểu dữ liệu
IN	dest	int	rank của tiến trình đích
IN	tag	int	để xác định thông điệp
IN	comm	MPI_Comm	communicator của tiến trình gửi và nhận

Truyền thông điểm-điểm

Định tuyến MPI_Recv

buffer
count
datatype
} *Nội dung thông điệp*

source
tag
communicator
} *Thông tin thông điệp (destination được hiểu ngầm định) - để* so khớp với thông điệp treo

status → *Thông tin về thông điệp đã nhận*

Giá trị của tag, source có thể là các ký tự thay thế.

Truyền thông điểm-điểm

```
MPI_Recv(void* buf, int count, MPI_Datatype dtype, int source,
int tag, MPI_Comm comm, MPI_Status* status)
```

Nhận truyền thông điệp từ source đưa vào buf

Giá trị trả về của hàm là mã lỗi

IN/OUT	Tên biến	Loại biến	Mô tả
OUT	buf	Biến con trỏ (không định kiểu)	Trỏ đến vùng nhớ chứa dữ liệu
IN	count	int	Số lượng phần tử trong dữ liệu
IN	dtype	MPI_Datatype	kiểu dữ liệu
IN	source	int	Rank của tiến trình gửi
IN	tag	int	để xác định thông điệp
IN	comm	MPI_Comm	Communicator của tiến trình gửi và nhận
OUT	status	MPI_Status	Thông tin về thông điệp đã nhận

Truyền thông điểm-điểm

Một số lưu ý quan trọng về định tuyến MPI_Recv

- Nếu thông điệp nhận được có số lượng phần tử lớn hơn giá trị count, sẽ phát sinh lỗi.
- Tiến trình gửi và nhận phải đồng ý với nhau về loại dữ liệu truyền thông. Nếu không, kết quả là không xác định (MPI không kiểm tra được chính xác dữ liệu)
- Khi định tuyến trả về giá trị
  - Dữ liệu nhận đã được chuyển vào bộ đệm;
  - Đối số status cho biết giá trị các tham số source, tag, và số lượng dữ liệu thực sự nhận được

Truyền thông điểm-điểm

Ví dụ 1 – Truyền thông điểm điểm

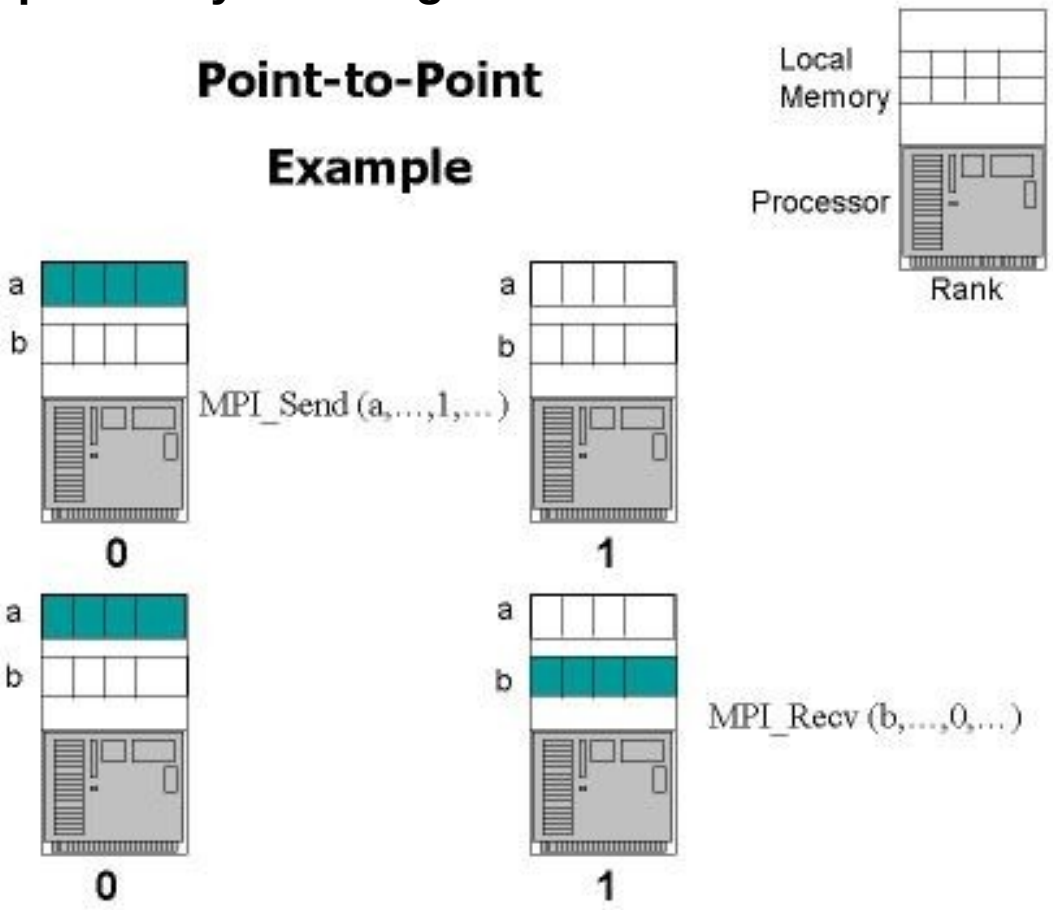
```
1.  /* simple send and receive */
2.  #include <stdio.h>
3.  #include <mpi.h>
4.  #include <math.h>
5.  int main (int argc, char **argv) {
6.      int myrank,i;
7.      MPI_Status status;
8.      double a[100],b[100];
9.      MPI_Init(&argc, &argv); /* Initialize MPI */
10.     MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* Get rank */
```

Truyền thông điểm-điểm

Ví dụ 1 – Truyền thông điểm điểm

```
11. if( myrank == 0 ){/* Send a message */
12.     for (i=0;i<100;++i) a[i]=sqrt(i);
13.     MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
14. }
15. else if( myrank == 1 ) /* Receive a message */
16.      MPI_Recv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD,
    &status );
17. MPI_Finalize(); /* Terminate MPI */
18. }
```

Ví dụ 1 – Truyền thông điểm điểm



Biên dịch: `mpicc comm_pp.c –o comm_pp -lm`

Truyền thông điểm-điểm

Ký tự thay thế, biến status

- Tiến trình nhận được phép nhận dữ liệu từ tiến trình gửi bất kỳ với tag bất kỳ.
- Bất kỳ nguồn nào: MPI_ANY_SOURCE
- Bất kỳ tag nào: MPI_ANY_TAG
- Có thể dùng ký tự thay thế cho nguồn hoặc tag hoặc cả hai

Truyền thông điểm-điểm

Ký tự thay thế, biến status

- Thông tin của tiến trình gửi nằm trong biến status
  - status.MPI_SOURCE
  - status.MPI_TAG
  - int MPI_Get_count(MPI_Status *status, MPI_Datatype dtype, int *count);
    - Kích thước của thông điệp
    - Số phần tử được gửi, các phần tử có loại dữ liệu xác định

Truyền thông điểm-điểm

Ví dụ 2 – Truyền thông điểm điểm

```
1. #include <stdio.h>
2. #include <mpi.h>
3. #include <math.h>
4. int main (int argc, char **argv) {
    5. int myrank,i;
    6. MPI_Status status;
    7. double a[100],b[300];
    8. MPI_Init(&argc, &argv); /* Initialize MPI */
    9. MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* Get rank */
```

Truyền thông điểm-điểm

Ví dụ 2 – Truyền thông điểm điểm

```
10. if( myrank == 0 ){   /* Send a message */
11.     for (i=0;i<100;++i) a[i]=sqrt(i);
12.     MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
13. }
14. else if( myrank == 1 ){ /* Receive a message */
15.      MPI_Recv( b, 300, MPI_DOUBLE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
    &status );
16.      MPI_Get_count(&status,MPI_DOUBLE,&count);
17.      printf("P:%d message came from rank %dn", myrank, status.MPI_SOURCE);
18.      printf("P:%d message had tag %dn",myrank,status.MPI_TAG);
19.      printf("P:%d message size was %dn",myrank,count);
20. }
21. MPI_Finalize(); /* Terminate MPI */
22. }
```

Truyền thông điểm-điểm

Ví dụ 2 – Truyền thông điểm điểm

Output:

P:1 message came from rank 0
P:1 message had tag 17
P:1 message size was 100

```
10.
11.
12.
13.
14.
15.                                                    MPI_COMM_WORLD,
16.
17.                                                    PI_SOURCE);
18.
19.
20.
21. MPI_Finalize(); /* Terminate MPI */
22. }
```

Truyền thông điểm-điểm

Trạng thái thời gian chạy

- Khi định tuyến MPI_Send được gọi, xảy ra 1 trong 2 trường hợp sau:
  - TH1: Thông điệp được lưu vào bộ đệm riêng của MPI, và được truyền tới đích sau, trong một tiến trình ngầm nào đó. Tiến trình gửi được phép chuyển sang lệnh khác ngay sau khi thông điệp chuyển hết vào bộ đệm.
  - TH2: Thông điệp vẫn nằm tại nơi nó được gọi, dưới dạng các biến của chương trình cho đến khi tiến trình đích sẵn sàng nhận.

- Thật ngạc nhiên khi ngay cả trong trường hợp 1, lời gọi MPI_Send có thể trả về giá trị *trước khi* bất kỳ *lời gọi không phải cục bộ* nào đó được bắt đầu.

Truyền thông điểm-điểm

Sự ràng buộc và sự hoàn thành

- Cả hai định tuyến MPI_Send và MPI_Recv đều bị ràng buộc với tiến trình gọi nó cho đến khi quá trình truyền thông hoàn thành.
- Quá trình truyền thông hoàn thành khi hoàn thành cả hai định tuyến MPI_Send và MPI_Recv
- Sự hoàn thành của định tuyến MPI_Recv là đơn giản và trực quan:
  - Khi so khớp đúng thông điệp, dữ liệu được chuyển tới đối số đầu ra của lời gọi
  - Ngay sau đó, các biến chứa thông điệp trong MPI_Recv sẵn sàng dùng cho lời gọi khác

Truyền thông điểm-điểm

Sự ràng buộc và sự hoàn thành (tiếp)

- Sự hoàn thành của định tuyến MPI_Send là đơn giản, nhưng không trực quan:
  - MPI_Send hoàn thành khi:
    - Thông điệp trong lời gọi không còn liên quan đến MPI
    - Về nguyên tắc, hoàn thành ngay sau khi thông điệp được đưa hết vào bộ đệm, thậm chí thông điệp chưa bị giải phóng khỏi tiến trình gửi

Truyền thông điểm-điểm

Sự ràng buộc và sự hoàn thành (tiếp)

- Sự hoàn thành của định tuyến MPI_Send là đơn giản, nhưng không trực quan:
  - Nếu kích thước thông điệp trong MPI_Send lớn hơn kích thước bộ đệm của MPI
    - Bộ đệm lúc này không thể dùng được
    - Thông điệp trong bộ đệm sẽ chiếm dữ không gian bộ đệm cho đến khi tiến trình đích bắt đầu nhận thông điệp hoặc khi bộ đệm đủ kích thước lưu trữ

Truyền thông điểm-điểm

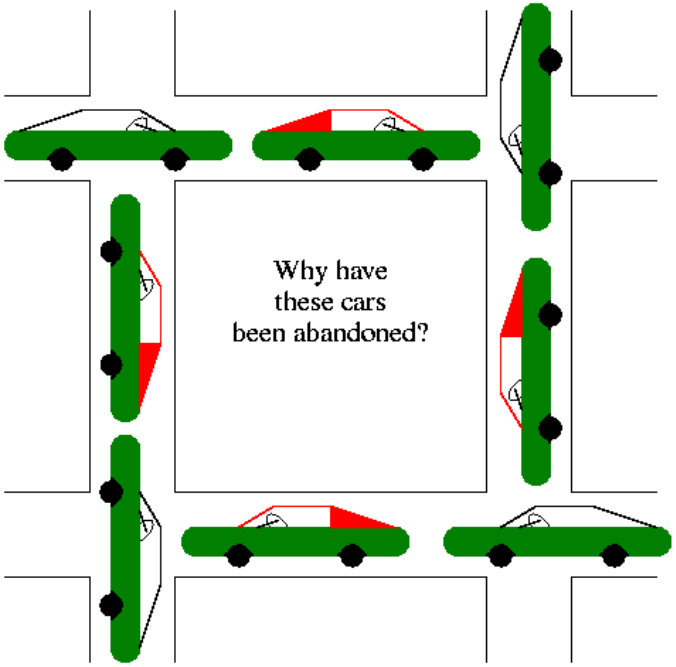
Sự ràng buộc và sự hoàn thành (tiếp)

- Định tuyến MPI_Recv so khớp với thông điệp treo dùng thông tin của thông điệp:
  - source, communicator
  - tag
- MPI không đảm nhận so khớp loại dữ liệu

Hiện tượng Deadlock

- DeadLock (bế tắc): là tình trạng hai hay nhiều tiến trình cùng chờ đợi một sự kiện nào đó xảy ra. Nếu không có sự tác động từ bên ngoài thì sự chờ đợi là vô hạn

- Qúa trình gửi nhận có ràng buộc có thể dẫn đến hiện tượng deadlock



Ví dụ 2 – Chương trình Deadlock

```
1.      if( myrank == 0 ) { /* Receive, then send a message */
2.              MPI_Recv(b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
3.              MPI_Send(a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
4.      }
5.      else if( myrank == 1 ) { /* Receive, then send a message */
6.              MPI_Recv(b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
7.              MPI_Send(a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
8.      }
9.      MPI_Finalize(); /* Terminate MPI */
```

Truyền thông điểm-điểm

Chương trình deadlock v1 – Không bị deadlock

```
1.      if( myrank == 0 ) { /* Receive a message, then send one */
2.              MPI_Recv(b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
3.              MPI_Send(a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
4.      }
5.      else if( myrank == 1 ) { /* Send a message, then receive one */
6.              MPI_Send(a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
7.              MPI_Recv(b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
8.      }
9.      MPI_Finalize(); /* Terminate MPI */
10.}
```

Chương trình deadlock v2

```
1.      if( myrank == 0 ) { /* Receive a message, then send one */
2.              MPI_Send(a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
3.              MPI_Recv(b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
4.      }
5.      else if( myrank == 1 ) { /* Send a message, then receive one */
6.              MPI_Send(a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
7.              MPI_Recv(b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
8.      }
```

Chương trình deadlock v3

```
1.  #define N 100000000;
2.        if( myrank == 0 ) { /* Receive a message, then send one */
3.                  MPI_Send(a, N, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
4.                  MPI_Recv(b, N, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &status );
5.        }
6.        else if( myrank == 1 ) { /* Send a message, then receive one */
7.                  MPI_Recv(b, N, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &status );
8.                  MPI_Send(a, N, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
9.        }
10.
```

Truyền thông điểm-điểm

Tránh hiện tượng deadlock

- Phân tích thiết kế chương trình phù hợp
- Tổ chức chương trình không phụ thuộc kích thước bộ đệm của MPI
- Có thể debug chương trình song song bằng các công cụ hỗ trợ như chức năng debug của ptp, hay một số công cụ hiển thị trực quan khác, …
- Sử dụng chế độ gửi nhận không ràng buộc

Truyền thông điểm-điểm

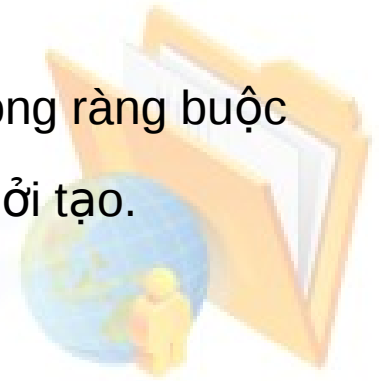
Truyền thông không ràng buộc

- Thực thi các định tuyến gửi, nhận mà không khóa tiến trình gọi nó.
- Tách quá trình khởi tạo và hoàn thành thành hai lời gọi MPI khác nhau.
  - Khởi tạo quá trình gửi/nhận
  - Hoàn thành quá trình gửi/nhận
- Giữa hai lời gọi, chương trình có thể thực hiện các định tuyền khác.
- Truyền thông tầng thấp là giống nhau, mặc dù giao diện của thư viện là khác nhau.
- Truyền thông ràng buộc và không ràng buộc có thể dùng lẫn với nhau.

Truyền thông điểm-điểm

Khởi tạo, hoàn thành, trình điều khiển

- Khởi tạo:
  - Khởi tạo quá trình gửi
  - Khởi tạo quá trình nhận
- Sau khi khởi tạo quá trình gửi/nhận, có hai cách để hoàn thành quá trình:
  - Kiểm tra xem quá trình đã hoàn thành hay chưa
  - Đợi cho đến khi hoàn thành
- Trình điều khiển:
  - Luôn được trả về bởi các định tuyền gửi/nhận không ràng buộc
  - Dùng để tham chiếu đến quá trình gửi/nhận đã khởi tạo.



Truyền thông điểm-điểm

Định tuyến khởi tạo MPI_Isend

```
MPI_Isend(void *buf, int count, MPI_Datatype dtype, int dest,
int tag, MPI_Comm comm, MPI_Request *request)
```

Không đọc/ghi biến nào trong lời gọi trước giai đoạn hoàn thành

IN/OUT	Tên biến	Loại biến	Mô tả
IN	buf	biến con trỏ (không định kiểu)	trỏ đến vùng nhớ chứa dữ liệu
IN	count	int	số lượng phần tử trong dữ liệu
IN	dtype	MPI_Datatype	kiểu dữ liệu
IN	dest	int	rank của tiến trình đích
IN	tag	int	để xác định thông điệp
IN	comm	MPI_Comm	communicator của tiến trình gửi và nhận
OUT	Request	MPI_Request	Định danh trình điều khiển

Truyền thông điểm-điểm

Định tuyến khởi tạo MPI_Irecv

```
MPI_Irecv(void* buf, int count, MPI_Datatype dtype, int source,
int tag, MPI_Comm comm, MPI_Request *request)
```

Không đọc/ghi biến nào trong lời gọi trước giai đoạn hoàn thành

IN/OUT	Tên biến	Loại biến	Mô tả
OUT	buf	Biến con trỏ (không định kiểu)	Trỏ đến vùng nhớ chứa dữ liệu
IN	count	int	Số lượng phần tử trong dữ liệu
IN	dtype	MPI_Datatype	kiểu dữ liệu
IN	source	int	Rank của tiến trình đích
IN	tag	int	để xác định thông điệp
IN	comm	MPI_Comm	Communicator của tiến trình gửi và nhận
OUT	request	MPI_Request	Định danh trình điều khiển

Truyền thông điểm-điểm

Định tuyến hoàn thành: MPI_Wait

```
int MPI_Wait(MPI_Request *request, MPI_Status* status)
```

- Biến request: xác định tiến trình khởi tạo gửi/nhận
- MPI_Wait trả về điều khiển khi tiến trình khởi tạo gửi/nhận hoàn thành
- Biến status:
  - Định tuyến khởi tạo gửi: mã lỗi cho quá trình gửi
  - Định tuyến khởi tạo nhận: thông tin của thông điệp nhận.

Truyền thông điểm-điểm

- **Định tuyến hoàn thành: MPI_Test**

```
int MPI_Test(MPI_Request *request, int *flag,
             MPI_Status* status)
```

  - Biến request: xác định tiến trình khởi tạo gửi/nhận
  - MPI_Test trả về điều khiển ngay khi gọi xong.
  - Biến flag:
    - true: định tuyến khởi tạo đã hoàn thành
    - false: định tuyến khởi tạo chưa hoàn thành
  - Biến status: khi (flag == true)
    - Định tuyến khởi tạo gửi: mã lỗi cho quá trình gửi
    - Định tuyến khởi tạo nhận: thông tin của thông điệp nhận.

Truyền thông điểm-điểm

Giảm trễ cho hệ thống mạng

```
MPI_IRECV(...,request)
...
arrived=FALSE
while (arrived == FALSE) {
   "work planned for processor to do while waiting for
   message data“
   MPI_TEST(request,arrived,status)
}
"work planned for processor to do with the message
   data"
```

Truyền thông điểm-điểm

Ví dụ 3 – Truyền thông không ràng buộc

```
1.  /* deadlock avoided */
2.  #include <stdio.h>
3.  #include <mpi.h>
4.  void main (int argc, char **argv){
5.        int myrank;
6.        MPI_Request request;
7.        MPI_Status status;
8.        double a[100], b[100];
9.        MPI_Init(&argc, &argv); /* Initialize MPI */
10.       MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* Get rank */
```

Truyền thông điểm-điểm

Ví dụ 3 – Truyền thông không ràng buộc

```
1.  /* deadlock avoided */
2.        if( myrank == 0 ) { /* Post a receive, send a message, then wait */
3.                MPI_Irecv( b, 100, MPI_DOUBLE, 1, 19, MPI_COMM_WORLD, &request );
4.                MPI_Send( a, 100, MPI_DOUBLE, 1, 17, MPI_COMM_WORLD );
5.                MPI_Wait( &request, &status );
6.        } else if( myrank == 1 ) { /* Post a receive, send a message, then wait */
7.                MPI_Irecv( b, 100, MPI_DOUBLE, 0, 17, MPI_COMM_WORLD, &request );
8.                MPI_Send( a, 100, MPI_DOUBLE, 0, 19, MPI_COMM_WORLD );
9.                MPI_Wait( &request, &status );
10.       }
11.       MPI_Finalize(); /* Terminate MPI */
12.}
```

Truyền thông điểm-điểm

Các chế độ gửi nhận

- Trong MPI có 4 chế độ gửi và một chế độ nhận
- 4 chế độ gửi:
  - Chế độ standard
  - Chế độ Synchronous
  - Chế độ Ready
  - Chế độ Bufferred
- Chế độ nhận: MPI_RECV và MPI_IRECV

Chế độ gửi	Hàm ràng buộc	Hàm không ràng buộc
Standard	MPI_SEND	MPI_ISEND
Synchronous	MPI_SSEND	MPI_ISSEND
Ready	MPI_RSEND	MPI_IRSEND
Bufferred	MPI_BSEND	MPI_IBSEND

Truyền thông điểm-điểm

Case study: Tìm kiếm song song

- Lập trình đếm số lần xuất hiện của một khoá (key) trong một dãy cho trước, sử dụng truyền thông điểm-điểm
- Dãy số cho trước được lưu trong một file text data.in gồm N phần tử có định dạng :
  - Số_phần tử_N
  - Phần_tử_thứ_1 phần_tử_thứ_2 .... phần_tử_thứ_N
- Các phần tử phân biệt bằng khoảng trắng.
- Yêu cầu chọn số tiến trình thực hiện là ước của số phần tử trong dãy.
- Khoá cần tìm kiếm được truyền vào dưới dạng tham số của chương trình. Ví dụ:
  - `$ mpirun -np 4 program 23`

Truyền thông điểm-điểm

Case study: Tìm kiếm song song

- Thiết kế chương trình
  - Tiến trình gốc (rank = 0)
  - Đọc dữ liệu vào từ file
    - Xác định số tiến trình tham gia np, số phần tử của mỗi miền.
    - Chia miền tính toán thành np miền con
    - Gửi dữ liệu các miền con cho các tiến trình tương ứng.
    - Tìm kiếm trên miền con mà nó đảm nhiệm
    - Nhận kết quả gửi về từ các tiến trình con.
    - Tính tổng số lần xuất hiện khoá key.
    - In kết quả ra màn hình.

Truyền thông điểm-điểm

Case study: Tìm kiếm song song

- Thiết kế chương trình :
  - Tiến trình con (rank > 0)
    - Nhận số phần tử được gửi đến từ tiến trình gốc
    - Nhận mảng dữ liệu tương ứng do tiến trình gốc gửi đến
    - Đếm số lần xuất hiện của khoá trong mảng dữ liệu mà tiến trình này đảm nhiệm
    - Gửi kết quả thu được về cho tiến trình gốc

Truyền thông điểm-điểm

Case study: Tìm kiếm song song

- Một số kiến thức trong C cần sử dụng:
  - Tham số của chương trình chính.
  - Cách chuyển một chuỗi sang số.
  - Các kĩ thuật làm việc với file.
  - Các kĩ thuật làm việc với con trỏ, mảng.

Truyền thông điểm-điểm

Case study: Tìm kiếm song song

```
1. #include <mpi.h>
2. #include <stdio.h>
3. #include <stdlib.h>
4. int rank0(int key);
5. int ranki(int key);

6. int main(int argc, char ** argv){
7.     int rank, key;
8.     MPI_Init(&argc, &argv);
9.     if (argc != 2) return -1;
10.    sscanf(argv[1], "%d", &key); // doc gia tri cua key
11.    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

12.    if (rank == 0)
13.            rank0(key);
14.    else
15.            ranki(key);

16.    MPI_Finalize();
17.    return 0;
18.}
```

Truyền thông điểm-điểm

Hàm đọc dữ liệu từ file data.in

```
1. #define FILE_NAME   "data.in”

2. int * loadData(int * NoElements){
3.      FILE * dataf;
4.      int * dataSet;
5.      int i;
6.      /* mo file */
7.      dataf = fopen(FILE_NAME, "r");
8.      if (!dataf){ /* loi mo file */
9.      printf("Loi khi mo file %s\n", FILE_NAME);
10.             return NULL;
11.     }
12.     /* doc so phan tu */
13.     fscanf(dataf, "%d", NoElements);
14.     /* cap phat vung nho cho mang du lieu */
15.     dataSet = (int*) malloc(sizeof(int) * (*NoElements));
16.     /* doc mang du lieu */
17.     for ( i=0 ; i< (*NoElements) ; i++ )
18.             fscanf(dataf, "%d", dataSet+i);
19.     fclose(dataf);
20.     return dataSet;
21.}
```

Truyền thông điểm-điểm

Hàm đếm số lần xuất hiện khoá key trong dãy arr

```
1. int search(int *arr, int ne, int key){
2.     int i;
3.     int count = 0;
4.     for ( i=0 ; i<ne ; i++ )
5.     if ( arr[i] == key )
6.           count++;
7.     return count;
8. }
```

Truyền thông điểm-điểm

Hàm rank0()

```
1. // dinh nghia cac tag dung trong truyen thong
2. #define NE  20
3. #define KEY 30
4. #define DATA       40
5. #define RESULT     60
6. int rank0(int key){
7.     int tne;                    // so phan tu cua day;
8.     int *dataSet;               // mang du lieu, duoc cap phat
   dong
9.     int np;                     // tong so tien trinh
10.    int sne;                    // so phan tu gui den cho moi
   tien                                  // trinh
11.    int count;                  // so lan xuat hien cua khoa
   key
12.    int i, tmp;
13.    MPI_Status status;
14.    /* lay ve so tien trinh */
15.    MPI_Comm_size(MPI_COMM_WORLD, &np);
16.    /* doc du lieu tu file */
17.    dataSet = loadData(&tne);
18.    /* tinh so phan tu gui cho moi tien trinh */
19.    sne = tne / np;
```

Truyền thông điểm-điểm

(tiếp)

```
1. /* Gui du lieu cho cac tien trinh con*/
2. for ( i=1 ; i<np ; i++ ){
3.     MPI_Send(&sne, 1, MPI_INT, i, NE, MPI_COMM_WORLD);
4.     MPI_Send(dataSet + i*sne,sne, MPI_INT, i, DATA, MPI_COMM_WORLD);
5. }
6. /* dem so lan xuat hien khoa tren mien con ma no dam nhiem */
7. count = search(dataSet, sne, key);

8. /* nhan du lieu tu cac tien trinh con gui ve */
9. for ( i=1 ; i<np ; i++ ){
10.        MPI_Recv(&tmp, 1, MPI_INT, i, RESULT, MPI_COMM_WORLD, &status);
11.        count += tmp;
12.}
13./* thong bao ket qua */
14.printf("So lan xuat hien khoa key trong day la %d\n", count);
15.free(dataSet);
16.return 0;
17.}
```

Truyền thông điểm-điểm

Hàm ranki()

```
1. int ranki(int key){
2.      int ne;         // so phan tu cua mang con
3.      int *buff;      // mang con
4.      int count;
5.      MPI_Status status;

6.      /* Nhan du lieu tu tien trinh 0 */
7.      MPI_Recv(&ne, 1, MPI_INT, 0, NE, MPI_COMM_WORLD, &status);
8.      buff = (int*) malloc(sizeof(int) * ne);
9.      MPI_Recv(buff, ne, MPI_INT, 0, DATA, MPI_COMM_WORLD,
   &status);

10.     /* thuc hien thao tac tim kiem */
11.     count = search(buff, ne, key);
12.     /* gui ket qua ve cho tien trinh goc */
13.     MPI_Send(&count, 1, MPI_INT, 0, RESULT, MPI_COMM_WORLD);
14.     return 0;
15.}
```

Truyền thông điểm-điểm