Vào ra song song

Hà nội, 6/2008

Đại học Bách khoa Hà Nội

Center of High Performance Computing

Hanoi University of Technology

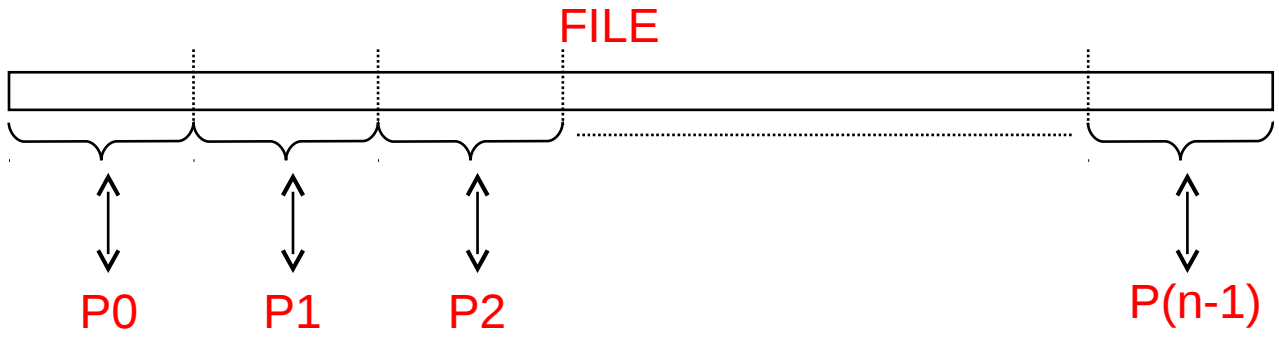
{hpcc@mail.hut.edu.vn}

Nội dung bài học

- Các cách tiếp cận vào ra song song
- Vào/ra dùng con trỏ riêng biệt
- Vào/ra dùng khoảng cách
- Khái niệm Khung nhìn
- Vào/ra cộng tác
- Vào/ra dùng con trỏ dùng chung

Các công nghệ vào/ra song song

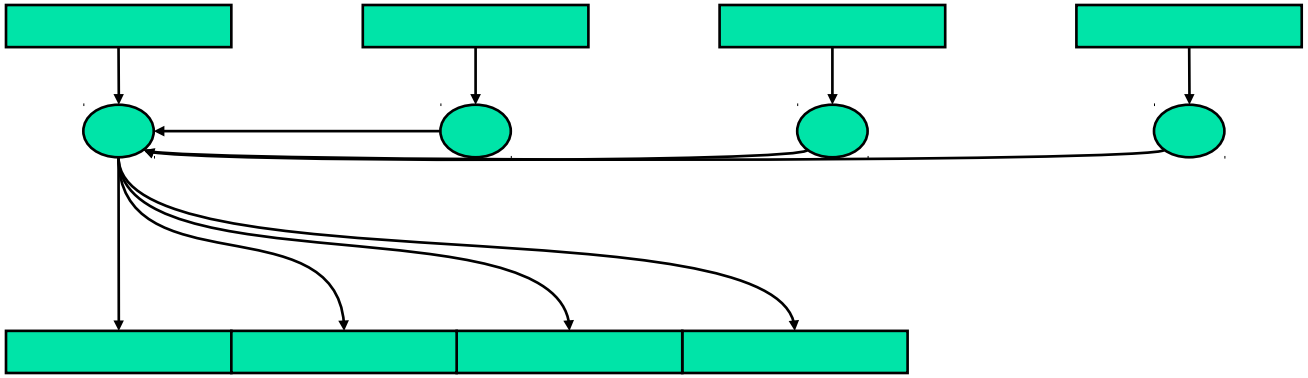
- Vào/ra song song là quá trình nhiều tiến trình của chương trình song song cùng truy cập một tệp tin chung



- Các công nghệ lưu trữ:
  - RAID: RAID 0, RAID 1, RAID 5, RAID 10, RAID 53
  - Distributed File Systems:NFS , SMB, DCE/DFS
  - Parallel File Systems:GPFS, PFS, PVFS
  - Storage Area Networks:CXFS, GFS, Lustre, SANergy

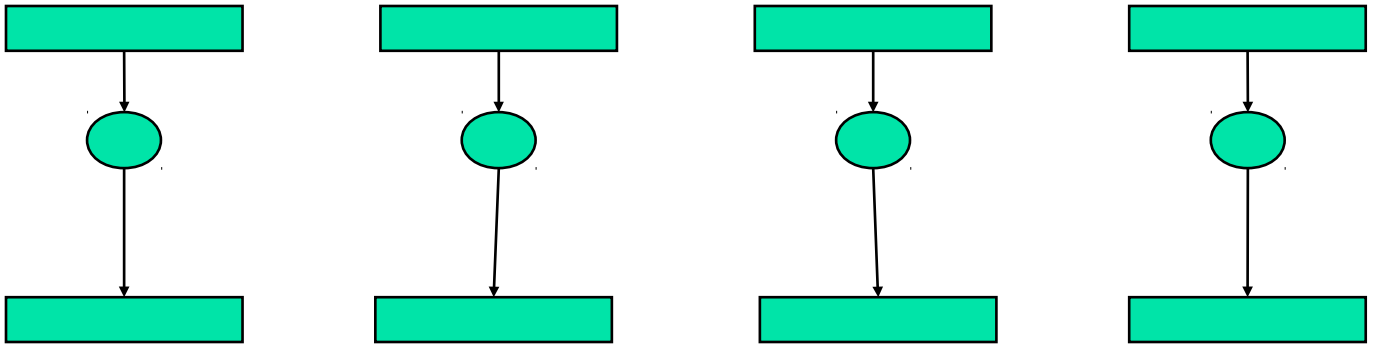
Các cách tiếp cận vào ra song song

- Vào ra dùng 1 tác vụ đơn:
  - Tất cả tiến trình gửi dữ liệu cho rank 0, sau đó rank 0 ghi dữ liệu ra tệp tin
  - Cần gom kết và phát tán dữ liệu
  - Hiệu năng giới hạn bởi khả năng của rank 0
  - Không tận dụng công nghệ lưu trữ song song



Các cách tiếp cận vào ra song song

- Vào/ra kết hợp
  - Mỗi tiến trình đọc/ghi vào một tệp tin riêng
  - Tăng tính song song
  - Hiệu năng cao
  - Nhiều tệp tin nhỏ, quản lý khó khăn
  - Có chương trình phân chia dữ liệu và tập hợp dữ liệu



Tại sao cần dùng vào/ra song song

- Vào/ra không song song đơn giản, nhưng
  - Hiệu năng thấp (một tiến trình ghi vào một tệp tin)
  - Không có sự tương tác giữa các chương trình (Mỗi tiến trình chỉ tương tác với 1 tệp tin)

- Vào/ra song song:
  - Hiệu năng cao
  - Tính khả chuyển
  - Tính thuận tiện
  - Một tệp tin có thể dùng cho các chương trình khác nhau (các chương trình ảo hóa, ...)

Cài đặt vào/ra song song sử dụng MPI

- Quá trình ghi giống với gửi dữ liệu, quá trình đọc giống với nhận dữ liệu
- Bất kỳ hệ thống vào/ra song song nào cũng cơ chế để:
  - Định nghĩa các phép toán cộng tác
    - MPI Communicator
  - Định nghĩa dữ liệu không liên tục trong bộ nhớ và tệp tin
    - Kiểu dữ liệu của MPI: dữ liệu cơ bản và dẫn xuất
  - Kiểm tra sự hoàn thành của các phép toán không ràng buộc
    - Đối tượng request trong MPI

Chuẩn vào/ra trong MPI

- Tất cả các định tuyến bắt đầu bằng MPI_File_
  - open, read, write, seek, close
- Kí tự không đồng bộ “i”: iread etc.
- Kí tự vị trí tuyệt đối “_at”: read_at
- Kí tự cộng tác “_all”: read_all etc
- Kí tự cộng tác không ràng buộc: “_begin” “_end”
- Kí tự con trỏ tệp tin dùng chung: “_shared”
- MPI_Type để tạo kiểu dữ liệu dẫn xuất

Một số định tuyến vào/ra cơ bản

MPI_File_open(MPI_Comm comm, char *file, int mode, MPI_Info info, MPI_File *fh)

(note: mode = MPI_MODE_RDONLY, MPI_MODE_RDWR, MPI_MODE_WRONLY, MPI_MODE_CREATE, MPI_MODE_EXCL, MPI_MODE_DELETE_ON_CLOSE, MPI_MODE_UNIQUE_OPEN, MPI_MODE_SEQUENTIAL, MPI_MODE_APPEND)

MPI_File_close(MPI_File *fh)

MPI_File_read(MPI_File fh, void *buf, int count, MPI_Datatype type, MPI_Status *status)

MPI_File_read_at(MPI_File fh, int offset, void *buf, int count,
MPI_Datatype type, MPI_Status *status)

MPI_File_seek(MPI_File fh, MPI_Offset offset, in whence);

(chú ý: whence = MPI_SEEK_SET, MPI_SEEK_CUR, or MPI_SEEK_END)

MPI_File_write(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

MPI_File_write_at(MPI_File fh, MPI_Offset, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

MPI_File_sync(MPI_File fh);

Vào/ra sử dụng con trỏ riêng biệt

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include "mpi.h"
4.  #define FILESIZE 1000

5.  int main(int argc, char **argv){
6.          int rank, nprocs;
7.          MPI_File fh;
8.          MPI_Status status;
9.          int bufsize, nints;
10.         int buf[FILESIZE];

11.         MPI_Init(&argc, &argv);
12.         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
13.         MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

14.         bufsize = FILESIZE/nprocs;
15.         nints = bufsize/sizeof(int);

16.         MPI_File_open(MPI_COMM_WORLD, "datafile",  MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
17.         MPI_File_seek(fh, rank * bufsize, MPI_SEEK_SET);
18.         MPI_File_read(fh, buf, nints, MPI_INT, &status);
19.         MPI_File_close(&fh);
20.         MPI_Finalize();
21. }
```

Vào ra sử dụng khoảng cách

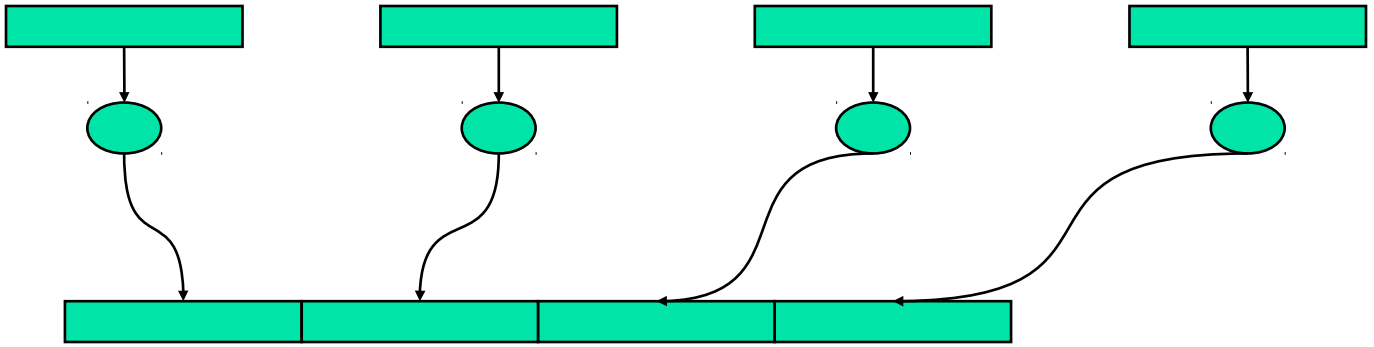
```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  #include "mpi.h"
4.  #define FILESIZE 1000

5.  int main(int argc, char **argv){
6.          int rank, nprocs;
7.          MPI_File fh;
8.          MPI_Status status;
9.          int bufsize, nints;
10.         int buf[FILESIZE];
11.
12.         MPI_Init(&argc, &argv);
13.         MPI_Comm_rank(MPI_COMM_WORLD, &rank);
14.         MPI_Comm_size(MPI_COMM_WORLD, &nprocs);

15.         bufsize = FILESIZE/nprocs;
16.         nints = bufsize/sizeof(int);

17.         MPI_File_open(MPI_COMM_WORLD, "datafile", MPI_MODE_RDONLY, MPI_INFO_NULL, &fh);
18.         MPI_File_read_at(fh, rank*bufsize, buf, nints, MPI_INT, &status);
19.         MPI_File_close(&fh);
20.         MPI_Finalize();
21. }
```

Ghi dữ liệu vào tệp tin

- Dùng **MPI_File_write** hoặc **MPI_File_write_at**
- Dùng **MPI_MODE_WRONLY** hoặc **MPI_MODE_RDWR** cho tham số flag trong định tuyến **MPI_File_open**
- Nếu tệp tin chưa tồn tại, cần truyền giá trị **MPI_MODE_CREATE** cho biến flag trong **MPI_File_open**
- Có thể dùng nhiều giá trị cho biến flag, bằng cách sử dụng toán từ bitwise-or ‘|’ trong ngôn ngữ C

Khái niệm khung nhìn

- Các tiến trình ghi dữ liệu vào cùng một tệp tin



- Bao gồm các thông tin
  - Offset (vị trí từ đầu file): cho biết các thao tác IO sẽ bắt đầu từ vị trí nào trong file
  - Kiểu dữ liệu sẽ được ghi hoặc đọc
  - Cách thức sắp xếp dữ liệu trong file (partern)
- **`MPI_File_set_view`** gán vùng dữ liệu của tệp tin cho các tiến trình

MPI_File_set_view

MPI_Set_view(MPI_File fh, MPI_Offset disp, MPI_Datatype etype,
MPI_Datatype filetype, char *datarep, MPI_Info info)

MPI_Set_view xác định phần nào của tệp tin có thể truy cập từ tiến trình đưa ra

disp: Khoảng cách tính theo byte từ vị trí đầu tệp tin

etype: Kiểu dữ liệu cơ bản của tệp tin (integer, doubles, ...).

filetype: Xác định sự phân tán dữ liệu đối với các tiến trình.
Thường trùng với giá trị của etype hoặc dẫn xuất từ etype

datarep: cách tổ chức dữ liệu

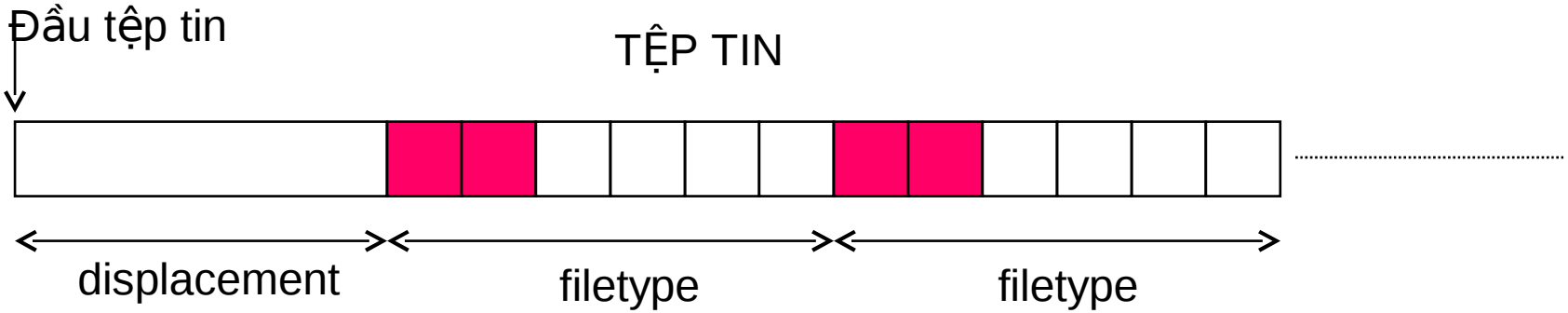
Tổ chức dữ liệu

- Biểu diễn cách thức tổ chức dữ liệu trong file
- Có thể được định nghĩa tùy biến
- Mặc định
  - Native:
    - Dữ liệu trong file và trong bộ nhớ được tổ chức giống nhau
  - Internal
    - Được thực hiện hoàn toàn bởi sản phẩm MPI
    - Thực hiện việc chuyển đổi kiểu khi cần thiết
  - External32
    - Sử dụng một cách thức tổ chức chuẩn
    - Tất cả các cài đặt đều hỗ trợ External 32

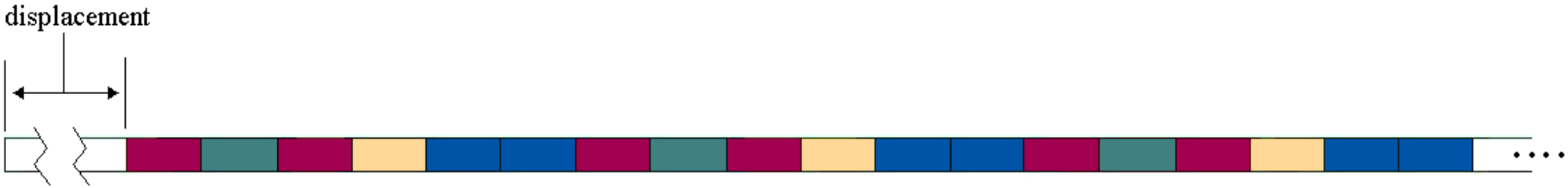
Ví dụ về khung nhìn

etype = MPI_INT

filetype = 2 số MPI_INT cùng với 1 khoảng trống gồm 4 số MPI_INT

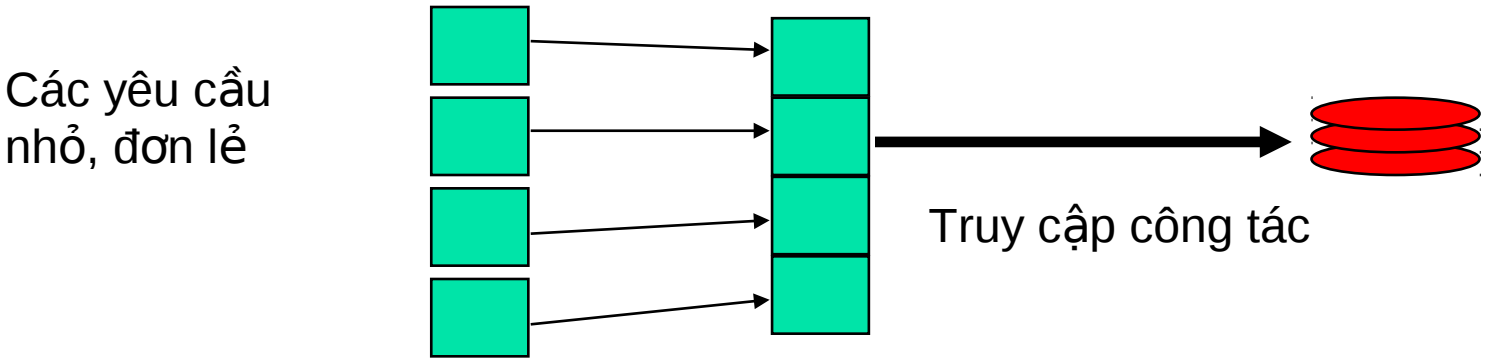


Kết hợp các khung nhìn



Vào/ra cộng tác trong MPI

- Các tiến trình có thể đọc file bằng các lệnh đọc riêng rẽ
- Có thể cải thiện hiệu năng bằng cách tập trung tất cả các lệnh đọc/ghi vào một lệnh đọc/ghi theo nhóm duy nhất.



Vào/ra cộng tác trong MPI

- **`MPI_File_read_all`**, **`MPI_File_read_at_all`**, **`MPI_File_write_all`, `MPI_File_write_at_all`**

- **`_all`** chỉ định tất cả các tiến trình trong communicator khai báo ở định tuyến **`MPI_File_open`** đều phải gọi định tuyến vào/ra này.

- Mỗi tiến trình chỉ truy cập thông tin của nó – danh sách các đối số là giống với các định tuyến không phải cộng tác.

Vào/ra cộng tác không ràng buộc trong MPI

- Vào/ra cộng tác là vào/ra có ràng buộc
  - Định tuyến chưa trả về giá trị cho đến khi việc đọc/ghi dữ liệu bộ đệm là an toàn
  - Vẫn ràng buộc với tiến trình gọi nó.

- Vào/ra không ràng buộc
  - MPI_File_iread(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Request *request)
  - MPI_File_iwrite(MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Request *request)
  - Giống với truyền thông không ràng buộc
  - Trả về đối tượng request thay vì status
  - Sử dụng MPI_Test và MPI_Wait cho quá trình hoàn thành

Vào/ra cộng tác không ràng buộc trong MPI

- Vào/ra cộng tác không ràng buộc:
  - Sử dụng đặc tính của vào/ra cộng tác, nhưng hoạt động theo kiểu không ràng buộc
- Khởi tạo đọc/ghi
  - MPI_File_read_all_begin(MPI_File fh, void *buf, int count, MPI_Datatype datatype)
  - MPI_File_write_all_begin(MPI_File fh, void *buf, int count, MPI_Datatype datatype)
- Hoàn thành đọc/ghi
  - MPI_File_read_all_end(MPI_File fh, void *buf, MPI_Status *status)
  - MPI_File_write_all_end(MPI_File fh, void *buf, MPI_Status *status)

Con trỏ file dùng chung

- Sử dụng một con trỏ file dùng chung cho tất cả các tiến trình
- Tất cả các thao tác đọc ghi đều dịch con trỏ chung đến vị trí mới phụ thuộc vào kích thước dữ liệu dịch chuyển
- Tất cả các tiến trình cần dùng chung một khung nhìn đối với file cần đọc
- int MPI_File_read_shared

  (MPI_File fh,

  void *buf, int count,

  MPI_Datatype datatype,

  MPI_Status *status)

Đảm bảo tính nhất quán khi vào/ra

- Đảm bảo tính nhất quán khi:
  - Nhiều tiến trình truy cập một tệp tin
  - Một hay nhiều tiến trình ghi vào cùng tệp tin
- MPI đảm bảo tính nhất quán tốt khi tất cả tiến trình trong communicator của hàm MPI_File_open đều truy cập tệp tin, nếu không tính nhất quán là kém
- Người dùng có thể tạo tính nhất quán khi MPI không điều khiển được

Ví dụ 1

- Mở tệp tin với **MPI_COMM_WORLD**. Mỗi tiến trình ghi vào một vùng riêng của tệp tin và đọc lại dữ liệu đã ghi

Tiến trình 0

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=0,cnt=100)
MPI_File_read_at(off=0,cnt=100)
```

Tiến trình 1

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=100,cnt=100)
MPI_File_read_at(off=100,cnt=100)
```

- MPI đảm bảo dữ liệu sẽ được đọc chính xác

Ví dụ2

- Giống ví dụ 1, nhưng mỗi tiến trình đọc dữ liệu đã được ghi bởi tiến trình khác
- MPI không đảm bảo dữ liệu tự động được đọc chính xác

Tiến trình 0

```
/* incorrect program */
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=0,cnt=100)
MPI_Barrier
MPI_File_read_at(off=100,cnt=100)
```

Tiến trình 1

```
/* incorrect program */
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=100,cnt=100)
MPI_Barrier
MPI_File_read_at(off=0,cnt=100)
```

- **Quá trình đọc không đảm bảo lấy về chính xác dữ liệu đã được ghi bởi tiến trình khác!**

Cải thiện ví dụ 2

- Người dùng phải tiến hành thêm một số bước để đảm bảo tính đúng đắn
- Có 3 cách:
  - Thiết lập chế độ atomicity
  - Đóng tệp tin, sau đó mở lại tệp tin đó
  - Đảm bảo không tồn tại chuỗi ghi nào xảy ra cùng lúc với chuỗi đọc/ghi khác trên các tiến trình khác
- Chuỗi: là tập các phép toán giữa các hàm open, close, file_sync
- Chuỗi ghi: chuỗi gồm toàn các phép toán ghi

Ví du 2 – giải pháp 1

- *Thiết lập chế độ atomicty*

Tiến trình 0

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_set_atomicity(fh1,1)
MPI_File_write_at(off=0,cnt=100)
MPI_Barrier
MPI_File_read_at(off=100,cnt=100)
```

Tiến trình 1

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_set_atomicity(fh2,1)
MPI_File_write_at(off=100,cnt=100)
MPI_Barrier
MPI_File_read_at(off=0,cnt=100)
```

- MPI_File_set_atomicity thiết lập tính nhất quán cho các phép toán truy cập dữ liệu
- MPI_File_set_atomicity là định tuyến cộng tác → các tiến trình trong communicator đều phải gọi với cùng tham số

Ví du 2 – giải pháp 2

- *Đóng tệp tin, sau đó mở lại tệp tin đó*

Tiến trình 0

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=0,cnt=100)
MPI_File_close
MPI_Barrier
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_read_at(off=100,cnt=100)
```

Tiến trình 1

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=100,cnt=100)
MPI_File_close
MPI_Barrier
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_read_at(off=0,cnt=100)
```

Ví du 2 – giải pháp 3

Tiến trình 0

```
MPI_File_open(MPI_COMM_WORLD,…)
MPI_File_write_at(off=0,cnt=100)
MPI_File_sync

MPI_Barrier

MPI_File_sync /*collective*/

MPI_File_sync /*collective*/

MPI_Barrier

MPI_File_sync
MPI_File_read_at(off=100,cnt=100)
MPI_File_close
```

Tiến trình 1

```
MPI_File_open(MPI_COMM_WORLD,…)

MPI_File_sync /*collective*/

MPI_Barrier

MPI_File_sync
MPI_File_write_at(off=100,cnt=100)
MPI_File_sync

MPI_Barrier

MPI_File_sync /*collective*/
MPI_File_read_at(off=0,cnt=100)
MPI_File_close
```

- *Dùng công thức sync-barrier-sync*

Kết luận

- Cách tiếp cận vào ra song song
- Khái niệm liên quan vào ra song song
- Các kiểu vào ra song song trong MPI
- Vấn đề đảm bảo tính nhất quán dữ liệu