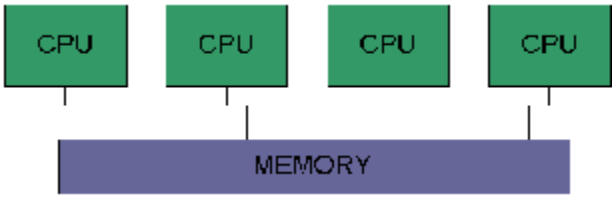
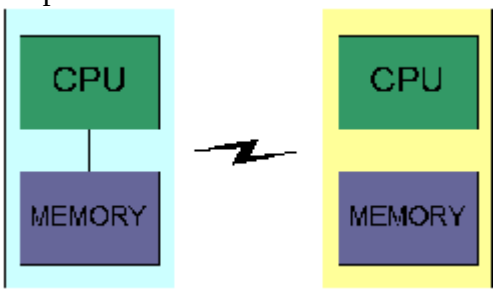
Chương 2: Các mô hình lập trình song song

I. Giới thiệu

- Mô hình chia sẻ bộ nhớ



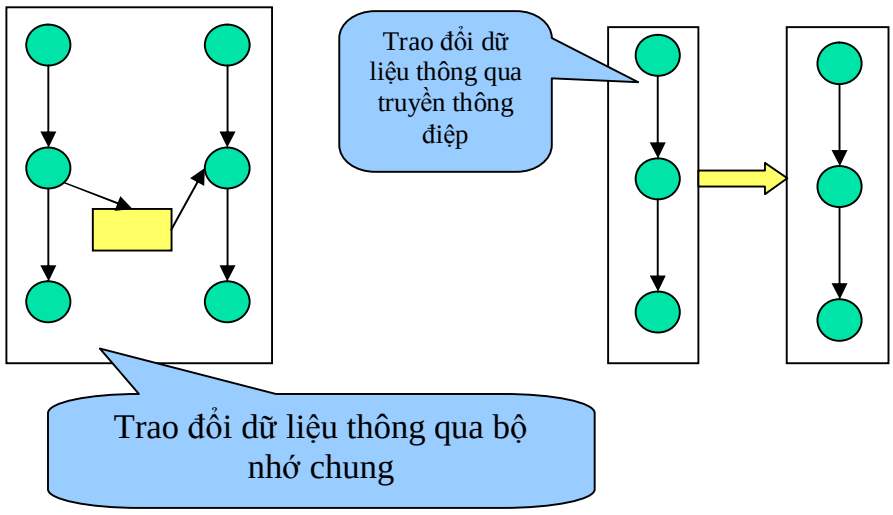
- Mô hình bộ nhớ phân tán:



- Các công cụ lập trình song song

	Bộ nhớ chung	**Bộ nhớ phân tán**
Công cụ hệ thống	Threads (pthread)	Sockets
Công cụ chuyên biệt	OpenMP Pthread	MPI PVM **Globus Toolkit 4 (GT4)**

- Mô hình trao đổi dữ liệu:



II. Lập trình chia sẻ bộ nhớ

Giả thiết rằng chúng ta có một hệ thống đa bộ xử lý đối xứng *SMP*. Đó là hệ thống trong đó tất cả các bộ xử lý là như nhau, không có những bộ xử lý đặc biệt để xử lý vào/ra, cũng không có bộ xử lý được gán cho nhiệm vụ đặc biệt nào khác. Đây là mô hình chung cho các hệ thống đa xử lý.

Để nghiên cứu về song song, chúng ta không nhất thiết phải có hệ đa bộ xử lý mức vật lý. Trong môi trường *UNIX, WINDOWS chúng ta có thể tạo ra nhiều tiến trình khác nhau trong hệ thống và chúng được sử dụng để mô phỏng lập trình đa bộ xử lý.*

Trong lập trình thủ tục tuần tự (như với C, C++, Pascal, Fortran), ta có thể mô tả bài toán một cách độc lập với các ngôn ngữ lập trình. Khi đã có mô tả về thuật toán ta dễ dàng cài đặt trên các ngôn ngữ lập trình tuần tự khác nhau bởi vì hầu hết các ngôn ngữ lập trình thủ tục đều sử dụng các lệnh và cấu trúc điều khiển chuẩn như: *tuần tự, rẽ nhánh if-then, các cấu trúc lặp (for, while, repeat), v.v.*

Tương tự như vậy, trong môi trường lập trình chia sẻ bộ nhớ có hai ràng buộc quan trọng mà chúng ta phải chú ý:

(i) *Một tiến trình có thể chờ một khoảng thời gian bất kỳ giữa hai câu lệnh cần thực hiện.* Giả sử bộ xử lý *P* thực hiện một chương trình có một 100 câu lệnh, bộ xử lý *Q* thực hiện chương trình có 10 câu lệnh và cùng bắt đầu thực hiện. Thậm chí, tất các câu lệnh có tốc độ thực hiện như nhau thì cũng không thể nói rằng *Q* sẽ kết thúc trước *P.*

(ii) *Không thể xem các lệnh thực hiện là đơn thể ở mức các ngôn ngữ lập trình.* Ví dụ, một lệnh đơn giản như: $a = a + b$ sẽ là một dãy các lệnh trong ngôn ngữ máy. Mà ta cũng biết rằng, các tiến trình và hệ điều hành chỉ nhận biết được các câu lệnh của ngôn ngữ máy.

1. Lập trình chia sẻ bộ nhớ dựa vào tiến trình

Yêu cầu đầu tiên của xử lý song song là phải tạo ra được một số các tiến trình cần thiết cho bài toán và khả năng huỷ bỏ chúng khi phần việc xử lý song song kết thúc để giải phóng bộ nhớ và các thiết bị mà các tiến trình đã chiếm giữ. Việc huỷ bỏ các tiến trình phải không cản trở hoạt động của những tiến trình khác.

VD: Cấu trúc một chương trình có N tiến trình song song

Tạo N tiến trình:

```
id = create_process(N);
```

=> Ta có N+1 tiến trình (một tiến trình chủ)

Phân công nhiệm vụ cho các tiến trình:

```
id = create_process(N);
  switch(id)
      {
```

```
                case 1: … do NhiemVu1 …(s1); break;
                case 2: … do NhiemVu2 …(s2); break;
                     . . .
                case N: … do NhiemVuN …(sn); break;
        }
```

Thu nhận kết quả tính toán:

```
join_process(N, 0);
// Các lệnh phải chờ
s=0;
For (i=1,n)
s=s+si;
```

Tiến trình chủ sẽ thu thập kết quả tính toán của các tiến trình khác và thực hiện các công việc còn lại, còn những tiến trình khác kết thúc. Khi đó chúng ta viết

`join_process(N, id);` *id* là tiến trình còn tiếp tục hoạt động.

Nếu ta đặt sau nó một số câu lệnh thì:

- Các câu lệnh này sẽ không được thực hiện cho đến khi tất cả các tiến trình đều thực hiện *join_process().*
- Sau đó chỉ còn lại một tiến trình chủ hoạt động.

Cách thức trao đổi dữ liệu giữa các tiến trình:

Một mặt một tiến trình có thể muốn giữ một phần dữ liệu cục bộ cho riêng mình, không cho những tiến trình khác nhìn thấy/truy cập tới những dữ liệu đó. Mặt khác, nó cũng muốn trao đổi thông tin với các tiến trình khác. Xử lý vấn đề che giấu hay chia sẻ thông tin như thế nào còn tuỳ thuộc vào mô hình mà chúng ta áp dụng, *dựa vào tiến trình hay luồng.*

- *Các tiến trình trong UNIX, WINDOWS được sử dụng như các đơn vị tính toán độc lập*. Khi muốn sử dụng bộ nhớ chung, ta cần phải xin cấp phát bộ nhớ và sau khi sử dụng xong phải giải phóng chúng. *Người lập trình phải có trách nhiệm giải phóng bộ nhớ chia sẻ một cách tường minh khi chúng không còn cần thiết sử dụng.* Có hai hàm cơ sở:
  - *shared(m, &id):* cấp phát *m* byte bộ nhớ chia sẻ cho tiến trình *id*.
  - *free_shm():* giải phóng bộ nhớ đã được cấp.
- *Đối với các luồng, tất cả các thông tin, theo mặc định, là nhìn thấy được.* Do vậy, trong mô hình này cần phải cố gắng để che giấu thông tin.

Ví dụ: Cho trước một đoạn chương trình tính tổng của hai vector:

```
for(i = 0; i < N; i++){          //  (1)
      C[i] = A[i] + B[i];
}
```

Thực hiện song song hoá đoạn chương trình này như thế nào?

Tương tự như ví dụ nêu trên, giả sử ta có M tiến trình. Chúng ta có thể chia N phần tử thành M phần (thường ta giả thiết N chia hết cho M) và gán từng phần đó cho mỗi tiến trình. Chu trình trên có thể viết thành:

```
for(j = id * N/M; j < (id+1)*N/M; j++){
      C[j] = A[j] + B[j];
}
```

Trong đó, *id* là số hiệu của tiến trình, chạy từ 0 đến M-1. Tiến trình thứ *i* xử lý N/M phần tử liên tiếp kể từ *i*N/M+1*, ví dụ hình 3-1 (a).

Hoặc ta có thể cho phép các tiến trình truy cập xen kẽ vào các phần tử của mảng như sau:

Tiến trình $\mathbf{P_i}$ bắt đầu từ phần tử thứ *i*, sau đó bỏ qua M phần tử để xử lý phần từ tiếp theo, nghĩa là nó truy cập đến i, i+M, i+2M, v.v., ví dụ hình 3-1 (b).

Chu trình (1) khi đó được viết như sau:

```
for(j = id; j < N; j+=M){
      C[j] = A[j] + B[j];}
```

Ví dụ: Khi N = 15 và M = 5 thì việc gán các phần tử của vector cho các tiến trình sẽ được thực hiện theo cách trên như sau:

P_1	P_2	P_3	P_4	P_5
1	4	7	10	13
2	5	8	11	14
3	6	9	12	15

(a)

P_1	P_2	P_3	P_4	P_5
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

(b)

Hình 3- Các cách phân chia chu trình của một mảng tuần tự

2. Lập trình chia sẻ bộ nhớ dựa vào luồng (thread)

Các luồng của một tiến trình có thể chia sẻ với nhau về không gian địa chỉ chương trình, các đoạn dữ liệu và môi trường xử lý, đồng thời cũng có vùng dữ liệu riêng để thao tác.

Các tiến trình và các luồng trong hệ thống song song cần phải được đồng bộ, song việc *đồng bộ giữa các luồng được thực hiện hiệu quả hơn đối với các tiến trình.* Đồng bộ giữa các tiến trình đòi hỏi tốn thời gian hoạt động của hệ thống, trong khi đối với các luồng thì việc đồng bộ chủ yếu tập trung vào sự truy cập các biến chung (global) của chương trình.

Nhiều hệ điều hành hiện nay hỗ trợ đa luồng như: SUN Solaris, Window NT, Windows 2000, OS/2, v.v. Hiện nay đã có một chuẩn cho việc lập trình song song dựa trên các luồng đó là *Pthread của IEEE Portable Operating System Interface, POSIX.*

III. Tính toán song song phân tán: mô hình gửi/nhận thông báo (cluster computing, Grid computing)

Tính toán phân tán là những tính toán được thực hiện trên cơ sở kết hợp khả năng tính toán và truyền thông của hai hay nhiều máy tính trên mạng.

Mô hình tính toán phân tán có những *ưu điểm sau*:

- Cho phép chia sẻ dữ liệu được lưu trữu ở nhiều máy tính khác nhau.
- Chia sẻ với nhau về một số chức năng chính của máy tính.
- Độ tin cậy cao hơn. Trong trường hợp có một máy tính bị trục trặc thì những máy tính khác có thể thay thế để hoàn thành nhiệm vụ của hệ thống.
- Tính kinh tế: thường đầu tư vào hệ phân tán sẽ thấp hơn đầu tư cho hệ tập trung.

Tuy nhiên, hệ tính toán phân tán cũng đứng trước *nhiều thách thức:*

+ Những vấn đề liên quan đến việc *quản trị hệ thống, vấn đề đảm bảo an toàn hệ thống, bảo mật thông tin,* v.v.

+ Xử lý trong các hệ thống phân tán không có bộ nhớ chia sẻ để trao đổi dữ liệu với nhau. Sự trao đổi được thực hiện bằng cách gửi/nhận thông báo.

Hiện nay có nhiều công cụ lập trình được sử dụng cho tính toán phân tán ở nhiều mức độ trừu tượng khác nhau, như: PVM, **MPI**, Globus Toolkit 4 v.v.

1) Mô hình gửi/nhận thông báo

Giống như mô hình chia sẻ bộ nhớ, các đơn vị xử lý song song trong mô hình gửi/nhận thông báo là các *tiến trình*. Tuy nhiên cũng có một số điểm khác nhau giữa hai mô hình này, trong mô hình gửi/nhận thông báo:

- Các tiến trình có thể thực hiện trên những bộ xử lý khác nhau và không truy cập được vào không gian bộ nhớ chia sẻ.
- Các tiến trình phân tán trao đổi dữ liệu với nhau qua hệ thống mạng cục bộ hoặc mạng diện rộng. Việc truyền thông và đồng bộ hoá hoạt động của các tiến trình được thực hiện thông qua hai phương thức *send()* và *receive().*
- Tất cả các biến là cục bộ của các tiến trình. Vì thế, những vấn đề về xung đột dữ liệu (cần phải khoá dữ liệu khi một tiến trình truy cập), hay tranh chấp thông tin (bài toán loại trừ nhau) không xuất hiện trong mô hình tính toán phân tán.

Nói chung có hai mô hình gửi/nhận thông báo:

- *Gửi/nhận thông báo theo cơ chế dị bộ:* Trong mô hình này, *một kênh truyền thông được giả thiết là có khả năng tiếp nhận không bị giới hạn*. Khả năng không giới hạn được *cài đặt trong thực tế bằng cách sử dụng bộ đệm* (buffer) để tiếp nhận các thông điệp gửi đến cho mỗi tiến trình. Do vậy, tiến trình gửi sẽ không phải chờ tiến trình nhận sẵn sàng nhận mà cứ gửi khi có dữ liệu. Ở đây, hai tiến trình gửi và nhận có thể hoạt động gần như đọc lập với nhau và thông điệp có thể nhận được

sau một khoảng thời gian nào đó (lâu bất kỳ) kể từ khi nó được gửi đi. Tuy nhiên, tiến trình nhận muốn nhân dữ liệu thì phải chờ cho đến khi có thông điệp của một tiến trình khác gửi cho nó. **Có một số yêu cầu sau trong truyền thông di bộ:**

- Khi tiến trình A gửi đi một thông điệp cho tiến trình B thì sau đó nó cần phải được biết xem B có nhận được hay không, nghĩa là A phải chờ để nhận được câu trả lời khẳng định của B. Việc phân phát thông điệp cũng không thể đảm bảo rằng không bị thất bại. Nếu A gửi đi một thông điệp cho B và A không nhận được câu trả lời từ B thì nó sẽ không biết là thông điệp đó đã được gửi đến đích B hay chưa? (có thể là tiến trình B không nhận được hoặc câu trả lời của B không đến được A).
- Tất cả các thông điệp đều phải đưa vào bộ đệm (hàng đợi), nhưng trong *thực tế không gian hàng đợi là hữu hạn*. Khi có quá nhiều thông điệp được gửi đi thì phương thức gửi sẽ bị chặn lại. Điều này vi phạm ngữ nghĩa của mô hình gửi/nhận thông báo dị bộ.

§ *Gửi/nhận thông báo theo cơ chế đồng bộ:* Trong mô hình này, *tiến trình gửi bị chặn lại cho đến khi tiến trình nhận sẵn sàng nhận.* Ở đây, sự truyền thông và đồng bộ hoá luôn gắn chặt với nhau.

Hệ thống gửi/nhận thông báo *đồng bộ* hoàn toàn giống như hệ thống điện thoại, kênh truyền thông bị chặn lại trong quá trình đàm thoại. Hệ truyền thông dị bộ lại giống với hệ thống bưu chính, người nhận phải chờ cho đến khi có thư được gửi đến.

Chúng ta hãy phân tích thêm để hiểu rõ sự phát triển của hai mô hình trên.

*) Cơ chế gửi/nhận thông báo đồng bộ:

Ưu điểm: Làm cho nhiều vấn đề trong đồng bộ hoá và việc cấp phát bộ nhớ động trở lên đơn giản hơn.

Nhược điểm:

- Việc gắn chặt các tiến trình với thời gian phân phát thông điệp cũng được xem như là điều kiện ràng buộc bổ sung đòi hỏi trong khi thiết kế và thực thi chương trình.
- Việc bắt tiến trình gửi phải chờ dẫn đến việc làm giảm tính đồng thời của hệ thống.
- Ngoài ra, để cài đặt hiệu quả các hệ thống truyền thông đồng bộ đòi hỏi phải có những phần cứng đặc biệt để đảm bảo rằng sự truyền thông phải cực nhanh và sự trao đổi dữ liệu không ảnh hưởng tới sự tính toán của hệ thống. Mà các mạng truyền thông nhanh có nhiều nút mạng trao đổi dữ liệu với nhau là rất đắt tiền. *Vì những lý do trên, nên hệ gửi/nhận thông báo dị bộ làm việc trên mạng cục bộ đã được phát triển mạnh mẽ hơn.*

Các mô hình lập trình dựa trên cơ chế gửi/nhân thông báo dị bộ.

§ Các yêu cầu và trả lời qua lại giữa *khách* (Client) và *chủ* (Server) – Mô hình hướng tâm:

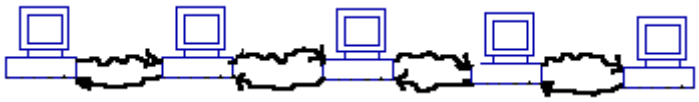
Mô hình này rất hay gặp. Đây là mô hình mà các máy tính chỉ có quan hệ gửi-nhận dữ liệu với một máy -- máy "chủ". Trong suốt quá trình tính toán, chúng không cần đến nhau.

Để cho rõ hơn, chúng ta trình bày tư tưởng phân chia miền dựa vào ví dụ lấy tích phân hàm số $\int_0^{p/2} \cos(x)dx$.

Do $\int_0^{p/2} \cos(x)dx = \int_0^{p/4} \cos(x)dx + \int_{p/4}^{p/2} \cos(x)dx$

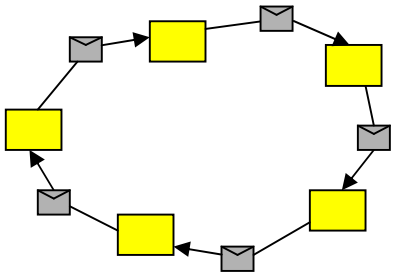
mà chúng ta có thể sử dụng 2 máy tính chạy song song. Máy thứ nhất tính giá trị $\int_0^{p/4} \cos(x)dx$, và máy thứ hai tính giá trị $\int_{p/4}^{p/2} \cos(x)dx$. Cuối cùng máy chủ sẽ cộng các kết quả.

§ Mô hình "đường-ống"



Mô hình đường ống là mô hình các máy tính được hình dung là xếp thành một hàng và mỗi máy tính gửi nhận dữ liệu cho 2 máy kề bên.

§ Mô hình "vòng-tròn"



Mô hình vòng tròn là mô hình các máy tính được hình dung là xếp thành một hàng và mỗi máy tính gửi nhận dữ liệu cho 2 máy kề bên.

Ngoài ra còn có mô hình: Hình sao, lưới 2D, lưới 3D, …

2. Lập trình song song phân tán

Lập trình theo mô hình gửi/nhận thông báo trong hệ thống nhiều máy tính có thể thực hiện theo ba cách:

Cách 1: Sử dụng ngôn ngữ lập trình song song đặc biệt, ví dụ Occam được thiết kế để sử dụng với các Transputer (Inmos 1986)

Cách 2: Sử dụng ngôn ngữ lập trình bậc cao (tuần tự) được mở rộng bằng cách bổ sung thêm các từ khoá và cú pháp mở rộng để xử lý việc trao đổi thông điệp, ví dụ CC++ (mở rộng của C++)

Cách 3: Sử dụng những ngôn ngữ lập trình bậc cao và các thư viện gồm những thủ tục xử lý việc trao đổi thông điệp, ví dụ ngôn ngữ C/C++ và hệ chương trình thư viện để chạy với PVM, MPI, …

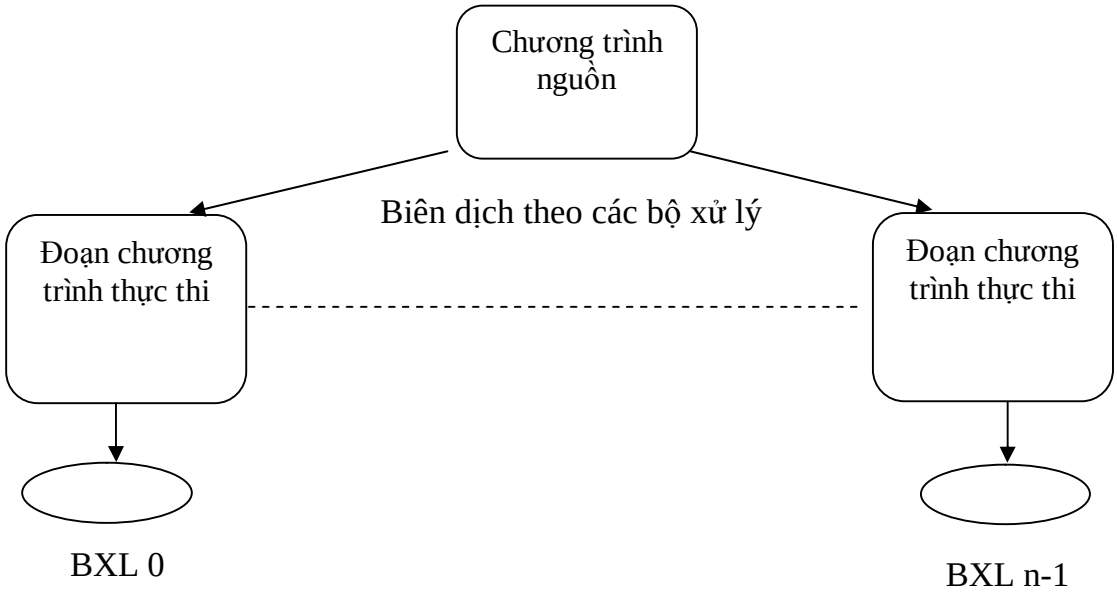
Sau đây chúng ta tập trung vào cách thứ ba. Trong hệ thống trao đổi thông điệp thì vấn đề tạo lập các tiến trình để thực hiện trên những bộ xử lý khác nhau và việc gửi, nhận thông điệp là quan trọng nhất.

Các bước xây dựng chương trình tính toán song song trên cơ sở trao đổi thông báo

Bước 1: Tạo các tiến trình con

Một chức năng quan trọng trong lập trình song song là tạo lập ra nhiều tiến trình để thực hiện những công việc con của một chương trình song song. Nói chung, một chương trình bắt đầu thực hiện như một tiến trình và sau đó phát sinh ra nhiều tiến trình con để khai thác khả năng song song của bài toán. Có hai cách tạo lập tiến trình: tạo lập tĩnh và tạo lập động.

- *Tạo lập tiến trình tĩnh*: số tiến trình được xác định trước khi thực hiện. Trong các hệ thống này thường có một *tiến trình điều khiển* còn được gọi là *tiến trình "chủ"* (*master*), những tiến trình khác được gọi là *tiến trình tớ* (slave). Đây là mô hình SPMD – sẽ có một đoạn mã chung cho tất cả các tiến trình. Sau khi chương trình nguồn được viết với các lệnh phân chia công việc cho từng tiến trình, nó sẽ được dịch sang mã thực thi được cho những tiến trình đó. Quá trình này được mô tả như hình 3-4.



Hình: Dịch đơn chương trình, đa thao tác dữ liệu

Ví dụ điển hình là hệ thư viện **MPI** được xây dựng theo cách tạo lập tĩnh như trên.

- *Tạo lập tiến trình động:* Các tiến trình có thể được tạo lập mới hoặc bị huỷ bỏ có điều kiện và số lượng tiến trình có thể thay đổi trong quá trình thực hiện. Mô hình cho phép thực hiện tạo lập động là MPMD (MIMD), trong đó những chương trình khác nhau có thể thực hiện trên những bộ xử lý khác nhau.

Bước 2: Trao đổi dữ liệu giữa các tiến trình thông qua các hàm *send()* và *receive()*

Việc gửi một thông điệp được thực hiện bằng cách xác định địa chỉ *của một hay tất cả các tiến trình nhận* theo một kênh truyền thông.

Để lựa chọn thông điệp, tiến trình nhận có thể dựa vào tiến trình gửi, kênh truyền thông, hay thẻ bài (*tag*) của thông điệp, v.v.

Có các dạng gửi/nhận như sau:

1. *Gửi thông điệp cho một tiến trình id:*

```
send(id: int, message: message_type);
send(id: int, tag: int, message: message_type);
```

2. *Gửi thông điệp tới một kênh truyền thông:* một thông điệp có thể gửi cho tất cả các tiến trình trên cùng một kênh *mych (my channel).*

```
send(mych: channel, message: message_type);
```

3. *Nhận thông điệp từ một kênh:* để nhận một thông điệp đang chờ đợi từ một kênh thì có thể sử dụng lời gọi hàm sau:

```
receive(mych: channel, message: message_type);
```

4. *Nhận từ một địa chỉ nguồn:*

```
receive(source_id: int, msg: message_type);
```

5. Nếu thông điệp được ghi thẻ thì tiến trình nhận có thể phân loại thông điệp trong hộp nhận và chọn thông điệp theo thẻ xác định.

```
receive(id: int, tag: int, msg: message_type);
```

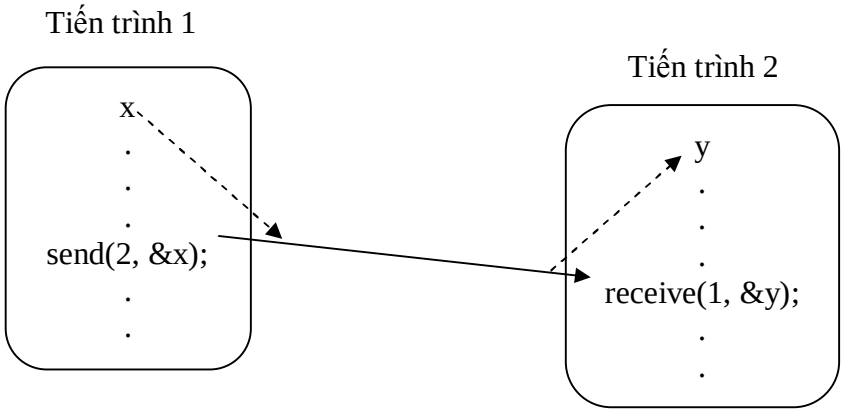
Ví dụ:

```
send(dest_id, &x );
```

trong tiến trình nguồn và gọi

```
receive(source_id, &y);
```

ở tiến trình đích để gửi giá trị dữ liệu *x* từ tiến trình nguồn (*source-id*) sang biến *y* cho tiến trình đích. Tất nhiên là *x, y* phải có cùng kiểu (kiểu tương thích với nhau) và cùng kích cỡ.



Hình : Sự trao đổi thông điệp giữa hai tiến trình

Như đã phân tích, việc gửi và nhận thông điệp có thể thực hiện một cách đồng bộ hoặc dị bộ. Trong mô hình dị bộ thì các thông điệp được gửi đi và được đưa vào bộ đệm để sau đó gửi dần tới cho các tiến trình đích. Để linh hoạt cho việc trao đổi, người ta thường gán cho mỗi thông điệp một thẻ bài (*tag*) và nó được sử dụng để phân biệt các thông điệp trong quá trình trao đổi.

3. Một số vấn đề trong lập trình song song phân tán

3.1. Truy vấn trên kênh

Nếu tiến trình gửi bị ngừng hoạt động hoặc thông điệp gửi đi nhưng không đến được hộp thư của người nhận thì tiến trình nhận sẽ bị chặn lại để chờ mãi mãi (dẫn đến tình trạng treo máy).

Để xử lý vấn đề này, hầu hết các chương trình thư viện cung cấp các hàm truy vấn để biết các trạng thái của kênh. Lời gọi *receive() chỉ* được thực hiện khi có những thông điệp đang chờ trên kênh truyền thông. Ngược lại, tiến trình này đi thực hiện những công việc khác. Để thực hiện được những công việc trên, chúng ta sử dụng các hàm sau:

1. *Kiểm tra xem trên kênh có những thông điệp gửi đến cho tiến trình hay không?*

```
empty(ch: channel);
```

2. *Hàm gọi để xác định xem thông điệp đang có trên kênh có phải được gửi từ tiến trình id và có thẻ tag?*

```
probe(id: int, tag: int);//id - tiến trình nhận
```

3.2. Truyền thông theo nhóm

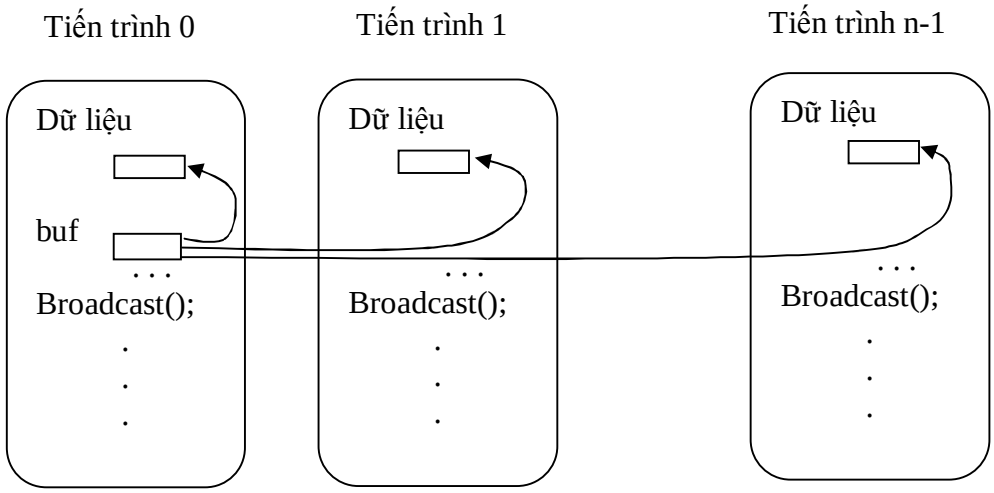
Nhiều chương trình phân tán cần phát tán và nhận dữ liệu từ nhiều tiến trình phân tán, nghĩa là cần trao đổi với từng nhóm trong chương trình song song. Để thực hiện truyền thông theo nhóm, chúng ta có thể sử dụng các hàm:

1. *Broadcast():* phát tán cùng một thông điệp cho tất cả các tiến trình trên kênh *mych*.

```
Broadcast(mych:channel,tag:int, msg:message_type);
```

Hoạt động của lệnh *Broadcast()* được mô tả như hình 3-6. Các tiến trình tham gia trao đổi trong phát tán dữ liệu phải được xác định. Trong hình 3-6, tiến trình số 0 được xem như tiến trình gốc chứa dữ liệu ở mảng *buf* để phát tán cho những tiến trình khác.

Theo qui ước của mô hình SPMD, mọi tiến trình đều thực hiện cùng một chương trình nên trong hình 3-6 tất cả các tiến trình đều gọi hàm *Broadcast().* Hành động phát tán dữ liệu sẽ không thực hiện được cho đến khi tất cả các tiến trình đều thực hiện lời gọi *Broadcast().*



Hình 3-6 Hoạt động phát tán dữ liệu

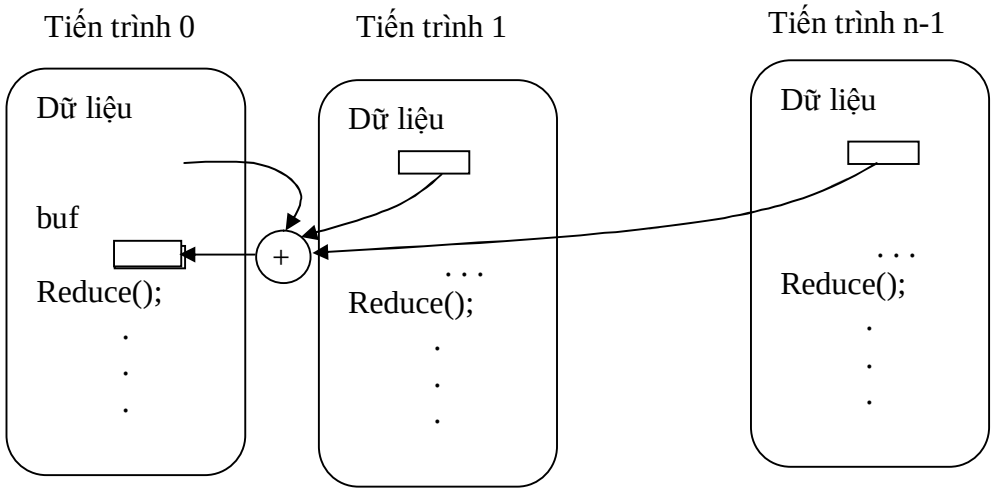
2. *Reduce():* thực hiện phép toán số học/logic trong nhóm các tiến trình và gửi kết quả tới tiến trình đích.

```
Reduce(mych:channel,op:op_type, res:Result_type,
       root: int,tag:int, msg:message_type);
```

Trong đó,

Op_type = {MAX, MIN, SUM, PROD, LAND, LOR, BAND, BOR, LXOR, BXOR}}

Ví dụ: hình 3-7 mô tả hàm ***Reduce()*** tập hợp các giá trị từ ***n tiến trình*** và thực hiện phép cộng (SUM) ở tiến trình gốc.



Hình 3-7 Hoạt động của hàm Reduce()

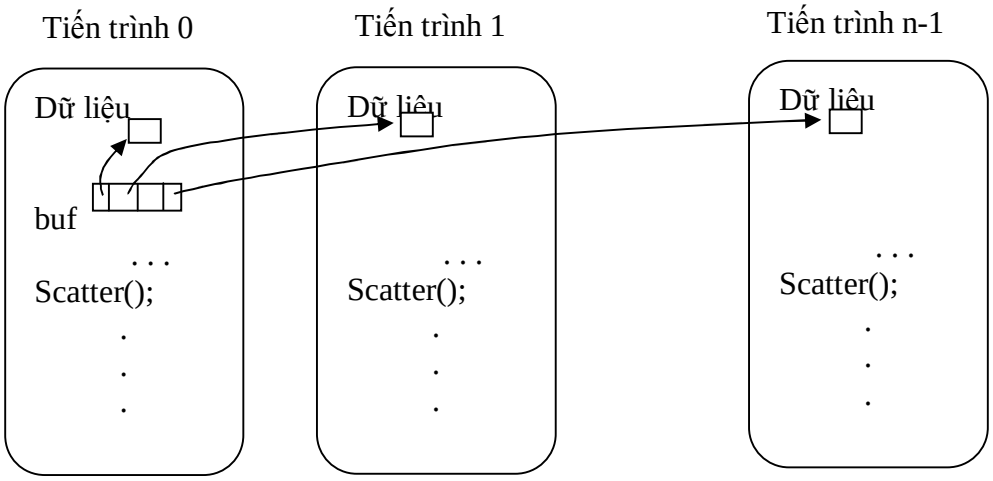
Trong mô hình SIMD lệnh Reduce() sẽ không thực hiện được cho đến khi tất cả các tiến trình đều thực hiện lời gọi ***Reduce().***

3. ***Scatter():*** phân tán công việc cho các tiến trình. Dữ liệu ở mảng ***buff*** được chia nhỏ thành ***n đoạn*** và phân tán cho ***n*** tiến trình trên kênh ***mych***.

```
Scatter(mych:channel, n:int, Buff[N]:DataType);
```

Hàm này được sử dụng để gửi phần tử thứ ***i*** của một mảng dữ liệu tới cho tiến trình thứ ***i***.

Hình 3-8 mô tả hoạt động của ***Scatter()***. Tương tự trường hợp của ***Broadcast(),*** lệnh ***Scatter***() sẽ không thực hiện được cho đến khi tất cả các tiến trình đều thực hiện lời gọi ***Scatter().***

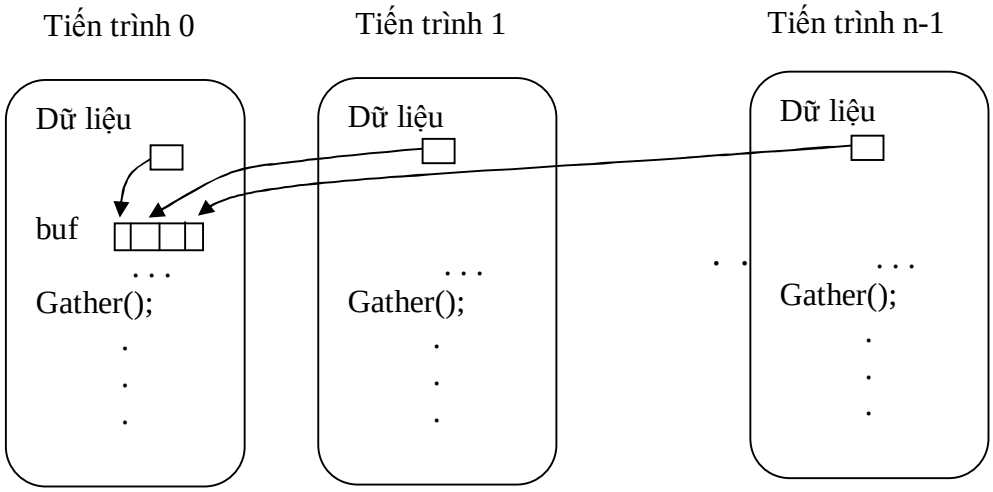


Hình 4-8 Hoạt động của hàm Scatter()

4. ***Gather():*** ngược lại so với hàm ***Scatter(),*** dữ liệu được gửi đi theo hàm ***Scatter()*** được xử lý bởi những tiến trình nhận được và sau đó được tập hợp lại cho một tiến trình.

```
Gather(mych:channel,Buff[N]:DataType,root:int);
```

Ngược lại hàm *Scatter()*, dữ liệu từ tiến trình thứ *i* được nhận về ở tiến trình gốc và được đưa vào phần tử thứ *i* của mảng *buf*, được mô tả như hình 3-9.



Hình 3-9 Hoạt động của hàm Gather()

5. *Barrier():* thực hiện việc đồng bộ hoá những tiến trình cùng gia nhập một kênh truyền thông. Mỗi tiến trình phải chờ cho đến khi tất cả các tiến trình khác trên kênh đạt đến điểm đồng bộ hoá bằng lời gọi *Barrier()* trong chương trình.

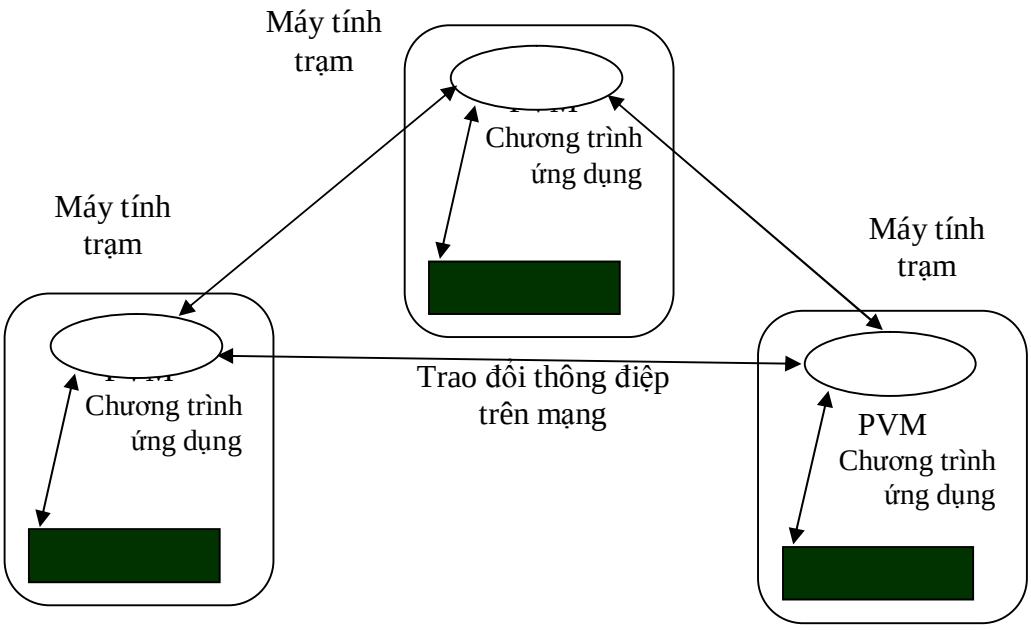
```
Barrier(mych:channel);
```

IV- Lập trình trên cụm máy tính

PVM cung cấp môi trường phần mềm để gửi/nhận thông báo cho các hệ máy tính thuần nhất và cả không thuần nhất. PVM có một tập hợp các hàm thư viện được viết bằng C/C++ hoặc Fortran.

Tập các máy tính được sử dụng trong mạng phải được định nghĩa theo các mức ưu tiên để chạy các chương trình. Điều này được thực hiện trên tập *máy ảo song song PVM.* Cách thực hiện tốt nhất là tạo ra một danh sách tên gọi của các máy tính và đặt ở *hostfile.*

Tệp này được PVM đọc để thực hiện các chương trình. Mỗi máy tính có thể chạy một hay nhiều tiến trình (chương trình ứng dụng). Các chương trình ứng dụng chạy trong các máy tính thông qua các tiến trình của PVM để trao đổi với nhau trên mạng như hình 3-10. Các tiến trình PVM yêu cầu đủ thông tin để chọn lựa đường truyền thông dữ liệu.

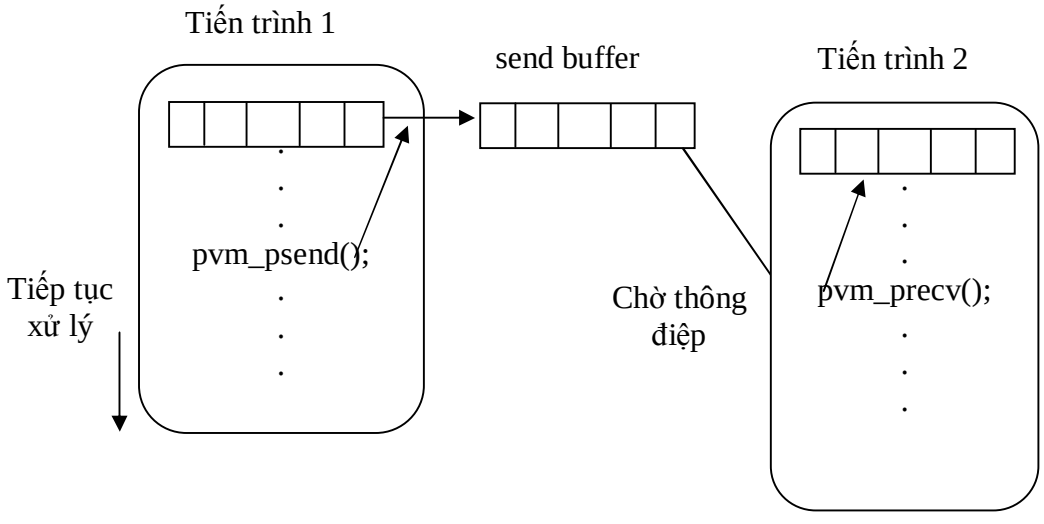


Hình 3-10 Sự trao đổi thông điệp của các máy tính trong hệ PVM

Các chương trình của PVM thường được tổ chức theo mô hình *chủ-tớ (master-slave)*, trong đó tiến trình chủ được thực hiện trước tiên, sau đó các tiến trình tớ sẽ được tạo ra trong tiến trình chủ đó. Hàm phát sinh tiến trình mới trong PVM là: *pvm_spawn().* Một tiến trình muốn tham gia vào hệ PVM thì nó phải ghi danh bằng cách gọi hàm *pvm_mytid(). Các tiến trình muốn được huỷ bỏ thì gọi hàm pvm_exit().*

Các chương trình trao đổi thông điệp với nhau thông qua các hàm *pvm_send()* và *pvm_recv().* Tất cả các thủ tục gửi đều không bị chặn (dị bộ) còn các thủ tục nhận thì hoặc bị chặn (được đồng bộ) hoặc không bị chặn. Các thao tác chính của việc gửi và nhận dữ liệu được thực hiện ở các bộ đệm *buffer.*

Nếu dữ liệu được gửi đi là một danh sách các mục có cùng kiểu thì trong PVM sử dụng pvm_psend() và *pvm_precv().* Hình 3-11 mô tả hoạt động của hai tiến trình trao đổi một mảng dữ liệu với nhau.



Hình 3-11 Gọi hàm pvm_psend() và pvm_precv()

Khi dữ liệu gửi đi là phức tạp, gồm nhiều kiểu khác nhau thì chúng phải được đóng gói lại (*pack*) để gửi đến *buffer*, sau đó tiến trình nhận lại mở gói (*unpack*) để nhận về dữ liệu tương ứng. Đó là các hàm:

pvm_pkint() và *pvm_upkint()* cho dữ liệu kiểu int
pvm_pkfloat() và *pvm_upkfloat()* cho dữ liệu kiểu float
pvm_pkstr() và *pvm_upkstr()* cho dữ liệu kiểu string, v.v.

Lưu ý: thứ tự mở gói để lấy dữ liệu ra phải đúng theo thứ tự mà chúng được đóng gói ở tiến trình gửi. Bộ đệm *buffer* để gửi dữ liệu là mặc định và nó phải được khởi tạo ở tiến trình gửi bằng lệnh *pvm_initsend().*

Tương tự, các lệnh khác về trao đổi thông điệp theo nhóm như: *pvm_bcast(), pvm_scatter(), pvm_gather(), pvm_reduce(), v.v.*

Ví dụ: Xét một chương trình gồm một chương trình chủ và một chương trình tớ. Chương trình này làm nhiệm vụ đơn giản là cộng 1000 số được sinh ngẫu nhiên (được lưu ở …/pvm3/src/rand_data.txt) với nhau.

/* Chương trình chính – tiến trình chủ */

```
#include<stdio.h>
#include<stdlib.h>
#include<pvm3.h>
#define SLAVE “spsum” // tên trình Slave
#define PROC 10 // số tiến trình
#define N 1000 // độ dài dữ liệu
main(){
  int mytid, tids[PROC];
  int n = N, nproc = PROC;
  int no, i, who, msgtype;
  int data[N], resul[PROC],tot=0;
  char fn[255];
  FILE *fn;

  mytid = pvm_mytid();//Đăng ký vào PVM
  /* Các tiến trình tớ bắt đầu làm việc
  Phát sinh 10 tiến trình con
  */
  no = pvm_spawn(SLAVE,(char**)0,0,””,nproc,tids);
  if(no < nproc){
     printf(“Gặp khó khăn trong việc tạo ra các tiến trình tớ\n”);
     for(i=0; i < no; i++) pvm_kill(tids[i]);
     pvm_exit(); exit(1);
  }
  /* Mở tệp để đọc và khởi tạo Data */
  strcpy(fn, getenv(“HOME”);
  strcat(fn,”/pvm3/src/rand_data.txt”);
  if((fn = fopen(fn, “r”){
     printf(“Không mở được tệp: %s\n”, fn);
     exit(1);
    }
  for(i=0; i < n; i++) fscanf(fn, “%d”, &data[i]);

  /* Phát tán dữ liệu cho các tiến trình tớ */
    pvm_initsend(PvmDataDefault); // khởi tạo buffer
     msgtype = 0;
     pvm_pkint(&nproc, 1, 1);
     pvm_pkint(tids, nproc,1);
```

```
    pvm_pkint(&n, 1, 1);
    pvm_pkint(data, n, 1);

  pvm_mcast(tids, nproc, msgtag);// Phát tán (1)
  /* Nhận kết quả từ các tiến trình tớ */
  msgtype = 5;
  for(i=0; i < nproc; i++){/*Nhận kết quả tính tổng con từ tiến trình tớ*/
    pvm_recv(-1, msgtype);
    pvm_upkint(&who, 1, 1);
    pvm_upkint(&result[who], 1, 1);
    printf("%d từ %d\n", result[who], who]);
    }
  /* Tính tổng những tổng con nhận được */
  for(i=0; i < nproc; i++) tot += result[i];
  printf("Tổng số là: %d \n\n", tot);
  pvm_exit();
  return 0;
  }
```

/* Chương trình slave */

```
#include<stdio.h>
#include "pvm3.h"
#define PROC 10
#define N 1000
main(){
  int mytid, tids[PROC];
  int n, me, i, nproc, x, msgtype, master;
  int data[N], sum;
  mytid = pvm_mytid();//Đăng ký vào PVM

  /* Nhận dữ liệu từ master */
  msgtype = 0;
  pvm_recv(-1, msgtype); /*Nhận dữ liệu từ (1) */
  pvm_upkint(&nproc, 1, 1);
  pvm_upkint(tids, nproc,1);
  pvm_upkint(&n, 1, 1);
  pvm_upkint(data, n, 1);

  /* Xác định tên tid */
  for(i=0; i < nproc; i++)
     if(mytid == tids[i]){me = i; break;}
  /* Cộng từng phần dữ liệu tớ tiến trình tớ */
  x = n / nproc;
  low = me * x;
```

```
high = low + x;
for(i = low; i < high; i++)
   sum += data[i];// Tổng con
/* Gửi kết quả cho master */
pvm_initsend(PvmDataDefault);
pvm_pkint(&me, 1, 1);
pvm_pkint(&sum, 1, 1);
msgtype = 5;
master = pvm_parent();
pvm_send(master, msgtype);
pvm_exit();
return (0);
}
```

V- Đánh giá các chương trình song song

Thời gian thực hiện song song

Để đánh giá được độ phức tạp tính toán của các thuật toán song song, ngoài số bước tính toán chúng ta còn cần đánh giá thời gian truyền thông của các tiến trình. Trong một hệ thống gửi/nhận thông báo, thời gian gửi/nhận thông báo cũng phải được xem là thời gian thực hiện của thuật toán.

Thời gian thực hiện song song, ký hiệu là t_p gồm hai phần t_{comp} và t_{comm}

$$t_p = t_{comp} + t_{comm}$$

Trong đó, t_{comp} là thời gian tính toán và t_{comm} - thời gian truyền thông dữ liệu.

Thời gian tính toán t_{comp} được xác định giống như thuật toán tuần tự. Khi có nhiều tiến trình thực hiện đồng thời thì chỉ cần tính thời gian thực hiện của tiến trình phức tạp nhất (thực hiện lâu nhất).

Thời gian truyền thông t_{comm} lại phụ thuộc vào kích cỡ của các thông điệp, vào cấu hình kết nối mạng đường truyền và cả cách thức truyền tải thông điệp, v.v. Công thức ước lượng thời gian truyền thông được xác định như sau:

$$t_{comm} = t_{startup} + n * t_{data}$$

Trong đó,

+ $t_{startup}$: Nó bao gồm cả thời gian để đóng gói thông điệp ở nơi gửi và thời gian mở gói ở nơi nhận. Để đơn giản chúng ta giả thiết thời gian này là *hằng số*.

+ t_{data} là thời gian cần thiết để gửi một từ dữ liệu (một mục dữ liệu) từ nơi gửi tới nơi nhận, được giả thiết là hàng số và *n* là số từ dữ liệu được trao đổi trong hệ thống.

Ví dụ: Giả sử cần thực hiện cộng *n* số trên hai máy tính, trong đó mỗi máy cộng n/2 số với nhau và tất cả các số đó được lưu ở máy tính thứ nhất. Kết quả của máy tính thứ hai khi được tính xong sẽ được gửi về máy tính thứ nhất để nó công hai kết quả bộ phận với nhau. Bài toán này được phát biểu như sau:

1. Máy tính thứ nhất gửi n/2 số cho máy tính thứ hai

2. Cả hai máy tính cộng n/2 số một cách đồng thời
3. Máy tính thứ hai gửi kết quả tính được về máy tính thứ nhất
4. Máy tính thứ nhất cộng hai kết quả để có kết quả cuối cùng.

Thời gian tính toán (ở bước 2 và 4):

$$t_{comp} = n/2 + 1$$

Thời gian truyền thông (ở bước 1 và 3):

$$t_{comm} = (t_{startup} + n/2 * t_{data}) + (t_{startup} + t_{data})$$
$$= 2*t_{startup} + (n/2 + 1) * t_{data}$$

Độ phức tạp tính toán là O(n) và độ phức tạp truyền thông cũng là O(n), do vậy độ phức tạp nói chung của thuật toán trên cũng là O(n).

VI- Kết luận

Để áp dụng cơ chế xử lý song song, chúng ta không nhất thiết phải có hệ đa bộ xử lý vật lý. *UNIX, Win32 hỗ trợ để tạo ra nhiều tiến trình khác nhau trong hệ thống và chúng được sử dụng để mô phỏng lập trình đa bộ xử lý.*

Hệ thống UNIX cung cấp những đặc tính như tín hiệu điều khiển *Semaphore* và *bộ nhớ chia sẻ* để các tiến trình có thể xử lý song song như chúng ta cần. Trong đó chúng ta có thể lập trình chia sẻ bộ nhớ dựa vào tiến trình để giải quyết các bài toán đặt ra.

Nhiều hệ điều hành hiện nay hỗ trợ *đa luồng*, ví dụ Window, OS/2, và UNIX. Một số ngôn ngữ lập trình, ví dụ C++, Java cũng hỗ trợ lập trình đa luồng.

Một tiến trình là bức tranh về sự hoạt động của một chương trình. Mỗi tiến trình được kết hợp với một hoặc nhiều luồng. Các luồng có thể xem như các tập con của một tiến trình.

Các luồng của một tiến trình có thể chia sẻ với nhau về không gian địa chỉ chương trình, các đoạn dữ liệu và môi trường xử lý, đồng thời cũng có vùng dữ liệu riêng để thao tác.

Một trong các mô hình lập trình song song đang được sử dụng phổ biến hiện nay là mô hình gửi/nhận thông báo. Trong mô hình gửi/nhận thông báo, các tiến trình chia sẻ với nhau kênh truyền thông. Kênh truyền thông là khái niệm trừu tượng của đường truyền thông vật lý giữa các tiến trình.

Các kênh được truy cập bởi hai phương thức: send() và receive(). Để bắt đầu trao đổi được với nhau thì một tiến trình phải gửi đi (*send*) một thông điệp vào kênh truyền thông và có một tiến trình khác yêu cầu nhận (*receive*) thông điệp đó. Sự trao đổi được hoàn tất khi dữ liệu đã được gửi từ địa chỉ nguồn tới đích.

Mô hình gửi/nhận thông báo nêu trên được sử dụng rất hiệu quả để lập trình song song theo cụ máy tính. Một môi trường cho hệ thống nhiều máy tính, nhất là các cụm máy tính đã được phát triển và được sử dụng phổ biến hiện nay là PVM . PVM cung cấp môi trường phần mềm để gửi/nhận thông báo cho các hệ máy tính thuần nhất và cả không thuần nhất. PVM có một tập hợp các hàm thư viện được viết bằng C hoặc Fortran.

7

Bài tập

1. Cho đoạn chương trình

```
printf("I am here\n");
id = create_process(15);
printf("%d is here\n", id);
```

Bao nhiêu dòng in ra " … is here", số hiệu của tiến trình in ra có theo một trật tự cố định qua các lần thực hiện hay không?

2. Viết một đoạn chương trình để tạo ra hai tiến trình và thực hiện tính 2+4+6+8 song song.

3. Viết chương trình tuần tự và chương trình song song để tính phép biến đổi ảnh *fractal* dựa trên công thức:

$$Z_{i+1} = z_i^3 + (c-1)z_i - c$$

Trong đó, $z_0 = 0$, c là số phức xác định toạ độ của điểm ảnh.

4. Viết chương trình song song với độ phức tạp O(log n) để tính giá trị của đa thức

$$F = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_{n-1}x^{n-1}$$

5. Viết chương trình song song để tính n! sử dụng hai tiến trình thực hiện đồng thời. Mỗi tiến trình tính gần một nửa dãy kết quả và một tiến trình chủ kết hợp hai kết quả lại.

6.Viết chương trình song song để tìm ra tất cả các số nguyên tố nhỏ hơn hoặc bằng N.

7.Viết chương trình song song theo luồng (Thread) để tính xác định giá trị cực đại của N phần tử.

8.Viết chương trình song song theo luồng trong mô hình chia sẻ bộ nhớ để thực hiện nhân hai ma trận cấp *n´n.*

9.Xây dựng hệ xử lý hình ống để tính sin x

$$\sin x = x - x/3 + x/5 - x/7 + x/9 + \ldots$$

10.Cho biết kết quả in ra của đoạn chương trình sau:

```
j = 0;
k = 0;
forall (i = 1; i <= 2; i++){
      j += 10;
      k += 100;}
printf("i = %d, j = %d, k = %d\n",i,j,k);
```

11.Người ta viết đoạn chương trình sau để thực hiện chuyển vị ma trận

```
forall (i = 1; i < n; i++)
  forall(j = 0; j < n; j++)
     a[i][j] = a[j][i];
```

Đoạn chương trình trên có thực hiện được không? nếu không thực hiện được thì hãy viết lại chương trình khác để thực hiện được việc chuyển vị ma trận.