ĐẠI HỌC KINH DOANH VÀ CÔNG NGHỆ HÀ NỘI

KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

GIÁO TRÌNH

KHÁI QUÁT GIẢI THUẬT SONG SONG

Chủ biên : TS. Hoàng Xuân Thảo

Biên soạn: GS. Trần Anh Bảo

ThS. Trần Văn Ước

(Dùng cho chương trình đào tạo hệ đại học)

Lưu hành nội bộ

HÀ NỘI - 2015

MỤC LỤC

LỜI NÓI ĐẦU

Việc nghiên cứu thiết kế các máy tính song song, và các thuật toán song song cũng như các ngôn ngữ lập trình hỗ trợ lập trình song song bắt đầu được quan tâm từ những năm 70, cho đến nay các ứng dụng của chúng đã lan rộng khắp các lĩnh vực của đời sống như đánh giá khả năng rủi ro về tài chính: dùng để mô hình hoá các xu hướng trên thị trường, hỗ trợ quyết định như phân tích thị trường, dự báo thời tiết, trí tuệ nhân tạo như thiết kế robot, xử lý ảnh ứng dụng trong công nghệ nhận dạng, điều khiển tự động, mô phỏng, trình chiếu hệ thống lớn … Bài toán nhân ma trận thưa với véc tơ, hay gặp trong các lời giải lặp của hệ phương trình tuyến tính, hệ phương trình giá trị riêng, véc tơ riêng... Khi ma trận kích thước lớn, việc thực hiện nhân ma trận với véc tơ lặp đi lặp lại nhiều lần yêu cầu khối lượng xử lý lớn mà xử lý tuần tự không đáp ứng được vì vậy việc nhân ma trận thưa với véc tơ song song là cần thiết và có vai trò quan trọng.

Giáo trình được xây dựng gồm các nội dung chính:

Chương 1: Trình bày tổng quan về xử lý song song, thuật toán song song và giới thiệu lập trình song song với MPI sử dụng Visual của Microsoft;

Chương 2: Trình bày về các thuật toán thiết kế cho nhân ma trận thưa với véc tơ song song;

Chương 3: Trình bày một số kết quả thực nghiệm trên một số bộ dữ liệu cho chương trình nhân ma trận thưa với véc tơ song song.

Chương 4: Giới thiệu lập trình song song

Chương 5: Thuật toán song song

Mặc dù đã hết sức cố gắng, tuy nhiên giáo trình cũng không tránh khỏi những thiếu sót. Chúng tôi rất mong được sự góp ý của các độc giả, đặc biệt đối với các đồng nghiệp và sinh viên để giáo trình ngày càng hoàn thiện.

Chương 1 - TỔNG QUAN XỬ LÝ SONG SONG

1.1. Khái niệm xử lý song song

Tính toán song song hay xử lý song song là quá trình xử lý thông tin trong đó nhiều đơn vị dữ liệu được xử lý đồng thời bởi một hay nhiều bộ xử lý để giải quyết một bài toán.

Nói cách khác: Xử lý song song là quá trình xử lý thông tin trong đó nhiều đơn vị dữ liệu được xử lý đồng thời bởi nhiều bộ xử lý để giải quyết một bài toán

1.2. Vì sao phải xử lý song song

- **Yêu cầu của người sử dụng:**

  + Cần thực hiện một khối lượng lớn công việc

  + Thời gian xử lý phải nhanh

- **Yêu cầu thực tế:**

+ Trong thực tế không tồn tại máy tính có bộ nhớ vô hạn và khả năng tính toán vô hạn.

+ Trong thực tế có nhiều bài toán mà máy tính xử lý tuần tự (XLTT) kiểu von Neumann không đáp ứng được.

+ Sử dụng hệ thống nhiều BXL để thực hiện những tính toán nhanh hơn những hệ đơn BXL.

+ Giải quyết được những bài toán lớn hơn, phức tạp hơn

Nhiều lĩnh vực mới như đồ họa máy tính, trí tuệ nhận tạo, phân tích số, v.v. đòi hỏi phải xử lý một khối lượng dữ liệu rất lớn ,những vấn đề về xử lý ngôn ngữ tự nhiên, nhận dạng, xử lý ảnh ba chiều (3-D), dự báo thời tiết v.v. đều đòi hỏi phảixử lý dữ liệu với tốc độc rất cao, với khối lượng dữ liệu rất lớn. Hầu hết những bài toán này, những máy tính xử lý tuần tự là không đáp ứng yêu cầu thực tế.do đó cần phải có những hệ thống máy tính thật mạnh mới đáp ứng được những yêu cầu của thực tế.

Mặc dù tốc độ xử lý của các Bộ xử lý tăng nhiều trong những năm qua, nhưng do giới hạn về vật lý nên khả năng tính toán của chúng không thể tăng mãi được.

Điều này dẫn tới là muốn tăng được khả năng tính toán của các hệ thống máy tính thì đích cuối cùng là phải khai thác được khả năng xử lý song song của chúng. Ngày càng xuất hiện nhiều bài toán mà những hệ thống đơn một bộ xử lý không đáp ứng được yêu cầu xử lý về thời gian, do đó đòi hỏi phải sử dụng những hệ thống đa bộ xử lý và đòi hỏi phải xử lý song song.

1.3. Sự khác nhau cơ bản giữa XLSS và XLTT

Trong tính toán song song, một số bộ xử lý cùng kết hợp với nhau để giải quyết cùng một vấn đề cho nên giảm được thời gian xử lý vì mỗi thời điểm có thể có nhiều phép toán được thực hiện đồng thời. Trong tính toán tuần tự với một bộ xử lý thì mỗi thời điểm chỉ thực hiện được một phép toán.

Mục đích của xử lý song song là tận dụng các khả năng của các hệ đa bộ xử lý để thực hiện những tính toán nhanh hơn trên cơ sở sử dụng nhiều bộ xử lý đồng thời. Cùng với tốc độ xử lý nhanh hơn, việc xử lý song song cũng sẽ giải quyết được những bài toán lớn hơn.

Xử lý tuần tự	Xử lý song song
Mỗi thời điểm chỉ thực hiện được một phép toán	*Mỗi thời điểm có thể thực hiện được nhiềuphép toán*
Thời gian thực hiện phép toán chậm	*Thời gian thực hiện phép toán nhanh*

1.4. Tiêu chí để đánh giá 1 thuật toán song song

Đối với thuật toán tuần tự:

- Thời gian thực hiện thuật toán.
- Không gian bộ nhớ.
- Khả năng lập trình.

✦ Đối với thuật toán song song

• Các tiêu chuẩn như thuật toán tuần tự.

• Những tham số về số BXL: số BXL, tốc độ xử lý.

• Khả năng của các bộ nhớ cục bộ.

• Sơ đồ truyền thông.

• Thao tác I/O.

1.5. Khái niệm hệ thống tính toán song song

Máy tính song song là một tập các tài nguyên tính toán có khả năng truyền thông và kết hợp với nhau để giải quyết các bài toán lớn trong khoảng thời gian chấp nhận được.Tài nguyên tính toán: CPU, RAM, …Máy tính song song là cách tiếp cận phổ biến nhất để xây dựng các siêu máy tính.Hệ thống tính toán song song chính là một máy tính song song.

1.6. Phân loại hệ thống tính toán song song theo mô hình Flynn

SISD (Single Instruction, Single Data): giống như máy tuần tự

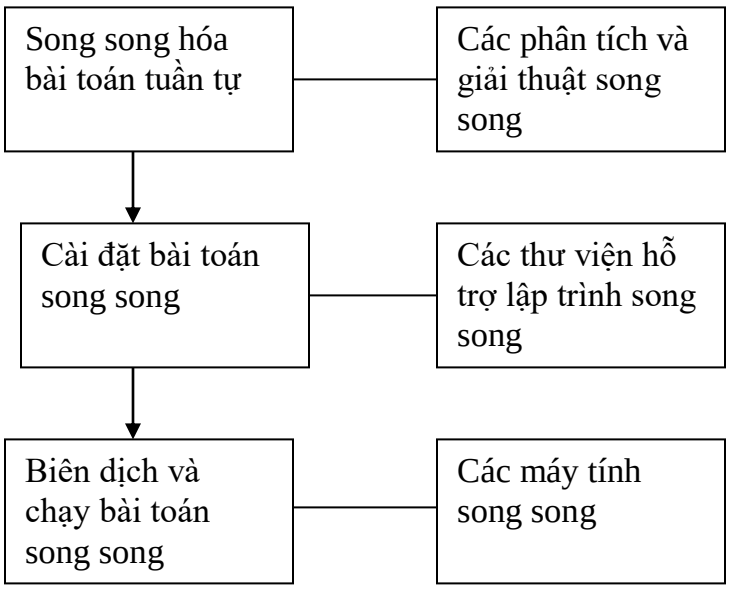
SIMD (Single Instruction, Multiple Data): song song hóa về mặt dữ liệu

MISD: Multiple Instruction, Single Data: chia sẻ bộ nhớ

MIMD: Multiple Instruction, Multiple Data: máy tính song song thực sự

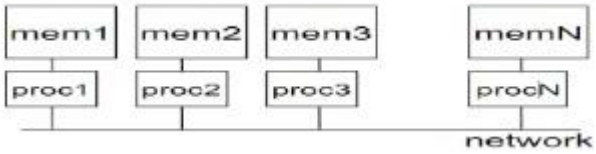
1.7. Chương trình song song

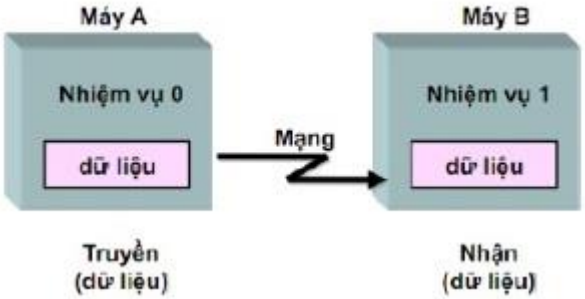
1.7.1. Các bước tổng quát phát triển ứng dụng song song



1.7.2. Phân loại chương trình song song

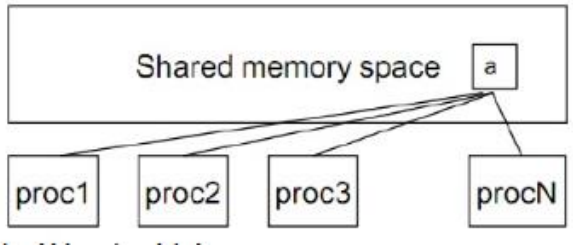
- Theo mô hình truyền thông điệp





- Mỗi tiến trình có một vùng nhớ riêng. Việc trao đổi dữ liệu, kết quả thực hiện dưới dạng thông điệp.

Theo mô hình bộ nhớ chia sẻ



Tồn tại một vùng không gian nhớ chung.

1.8. Giải thuật song song

1.8.1. Song song hóa bài toán tuần tự

Thiết kế giải thuật song song là chia bài toán thành các bài toán nhỏ hơn và gán bài toán nhỏ cho các bộ vi xử lý khác nhau để thực hiện song song.Quá trình thiết kế giải thuật song song là quá trình song song hóa bài toán tuần tự

1.8.2. Khả năng song song hóa

Không phải giải thuật nào cũng có khả năng song song hóa. Những giải thuật không thể song song hóa:Các tham số đầu vào cho bước i+1 chính là các kết quả đầu ra của bước thứ i.Ví dụ : bài toán Fibonaci

1.8.2. Trình tự song song hóa

Mô hình 5 pha:

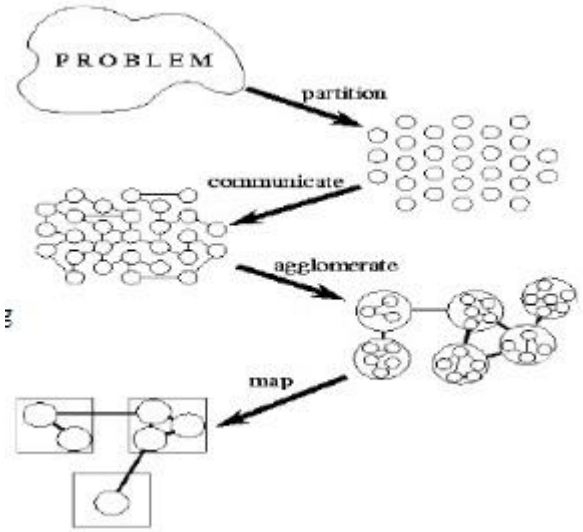
Xác định rõ vấn đề

Phân hoạch

Truyền thông

Gom kết

Ánh xạ



- Pha 1,2,3: tìm kiếm khả năng songsong.

- Pha 4,5: tối ưu tính song song

1.9. Một số ví dụ về ý tưởng song song

1.9.1. Tính giai thừa

Ví dụ: Giả sử cần thực hiện tính n! trên hai máy tính, trong đó mỗi máy nhân n/2số với nhau và n lưu ở máy tính thứ nhất.

Kết quả của máy tính thứ hai khi được tính xong sẽ được chuyển về máy tính thứ nhất để nó nhân hai kết quả bộ phận với nhau.

Bài toán này được phát biểu như sau:

B1.Máy tính thứ nhất gửi n cho máy tính thứ hai

B2.Cả hai máy tính thực hiện nhân n/2 số một cách đồng thời

P1: 1*2*3…*(ndiv2).

P2: (ndiv2+1)*((ndiv2+2)*….*n

B3.Máy tính thứ hai chuyển kết quả tính được về máy tính thứ nhất

B4.Máy tính thứ nhất nhân hai kết quả để có kết quả cuối cùng.

Thời gian tính toán (ở bước 2 và 4):

tcomp = n/2 + 1

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

tcomm = (tstartup + tdata) + (tstartup + tdata)

= 2*tstartup + 2 * tdata

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

Hiện nay, nhiều ngôn ngữ lập trình song song đang được sử dụng như: Posix, MPI, OpenMP, VPM , Fortran 90, CUBE C, Occam, C-Linda, PVM với C/C++, CDC 6600, JAVA v.v. là công cụ quan trọng cho phép chúng ta cài đặt thuật toán song song trên những mô hình máy tính hỗ trợ việc xử lý song song. Xử lý song song là một vấn đề phức tạp, khó khăn trước mắt chính là sự thay đổi về tư duy thuật toán (lâu nay chúng ta đã quen với cách nhìn vấn đề một cách tuần tự), xây dựng

mô hình máy tính song song, kỹ thuật cài đặt chương trình …. nhưng những gì mà lập trình song song đem lại thì thật là to lớn, không thể phủ nhận.

1.9.2. Tìm phần tử trên mảng

Ta xét bài toán tìm phần tử a trên mảng A kích thước n.

Trong xử lý tuần tự: ta dùng một bộ xử lý duyệt từ phần tử đầu đến phần tử cuối của mảng.

Trong xử lý song song, giả sử ta có một mô hình song song m bộ xử lý, ta chia việc cho mỗi bộ xử lý đồng thực hiện tìm kiếm, một bộ xử lý tìm kiếm trên (n div m)phần tử. Trong quá trình thực hiện, bộ xử lý nào tìm thấy phần tử a hoặc đã duyệt qua hết rồi nhưng không tìm thấy thì phải gửi thông điệp để hệ thống xử lý nhận biết, điều khiển quá trình xử lý.

Chúng ta dễ nhận thấy là độ phức tạp của xử lý song song có thể sẽ lớn hơn xử lý tuần tự rất nhiều, bởi vì cần có sự trao đổi thông tin và sự đồng bộ các tiến trìnhtrong quá trình thực hiện xử lý bài toán,vấn đề chúng ta cần quan tâm ở đây chính là thời gian thực hiện chương trình.

Một trong những mục đích chính của xử lý song song là nghiên cứu, xây dựng những thuật toán thích hợp để cài đặt trên các máy tính song song, nghĩa là phát triển các thuật toán song song nhằm giải quyết các bài toán đặt ra trong thực tế.

Chương 2 - KIẾN TRÚC MÁY TÍNH SONG SONG

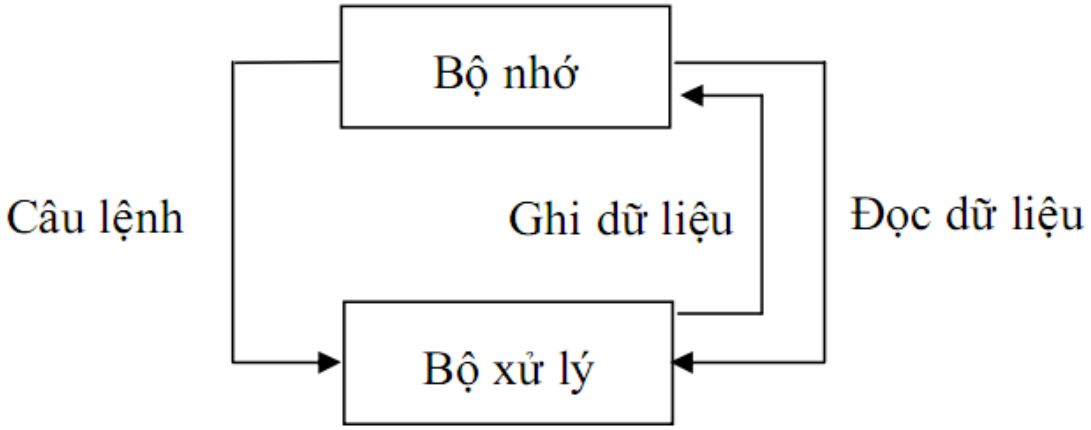
Máy tính được xây dựng từ các khối cơ sở:

- Bộ nhớ: để lưu trữ dữ liệu
- Các đơn vị logic và số học: thực hiện các phép toán được ký hiệu là ALU
- Các phần tử điều khiển:Bộ điều khiển CU và các thiết bị vào/ra dữ liệu
- Các Bus trao đổi dữ liệu.

Cách thức liên kết các khối trên với nhau cho ta biết kiến trúc của máy tính.

Trong các hệ thống máy tính thông thường có hai khối quan trọng nhất là bộ nhớ và BXL. BXL xử lý dữ liệu được lưu trữ trong bộ nhớ thông qua các chỉ thị (các câu lệnh). Các câu lệnh cũng được lưu trong bộ nhớ. Trong hệ thống dữ liệu được thực hiện theo cả hai chiều, đọc và ghi vào bộ nhớ.

Hình 2-1 mô tả hoạt động của mô hình máy tính kiểu von Neumann



Hình 2-1: Sự liên kết giữa bộ nhớ và bộ xử lý

2.1. Phân loại máy tính song song

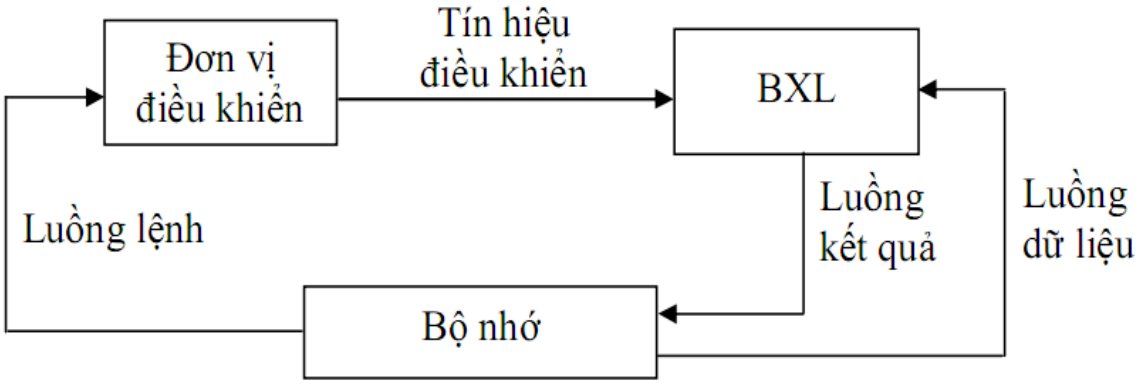
2.1.1. Mô hình SISD

(Single Instruction stream, Single Data Stream - Đơn luồng lệnh, đơn luồng dữ liệu)

Máy tính loại SISD chỉ có một CPU, ở mỗi thời điểm thực hiện một chỉ lệnh và chỉ đọc, ghi một mục dữ liệu.

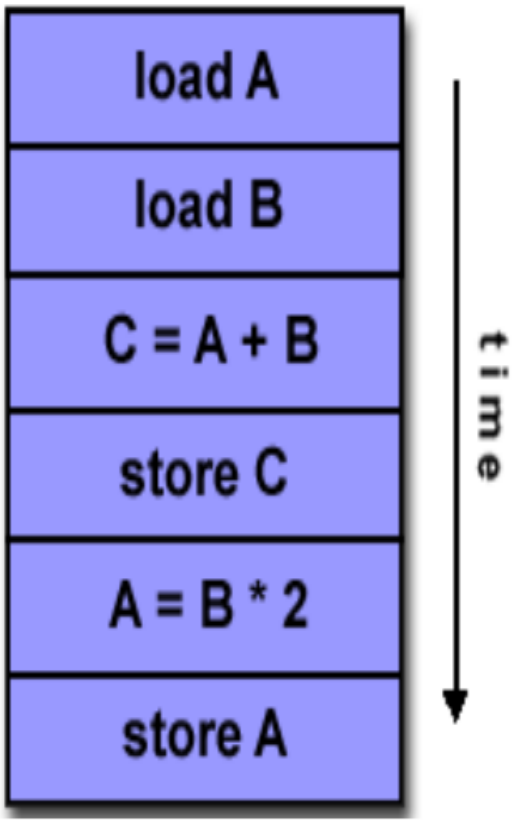
Tất cả các máy tính SISD chỉ có một thanh ghi register được gọi là bộ đếm chương trình (program counter) được sử dụng để nạp địa chỉ của lệnh tiếp theo và kết quả là thực hiện theo một thứ tự xác định của các câu lệnh.

Hình 2-2 mô tả hoạt động của máy tính theo mô hình SISD.



Hình 2-2: Mô hình của kiến trúc SISD

Mô hình SISD còn được gọi là SPSD (Single Program Single Data), đơn chương trình và đơn dữ liệu. Đây chính là mô hình máy tính kiểu von Neumann.



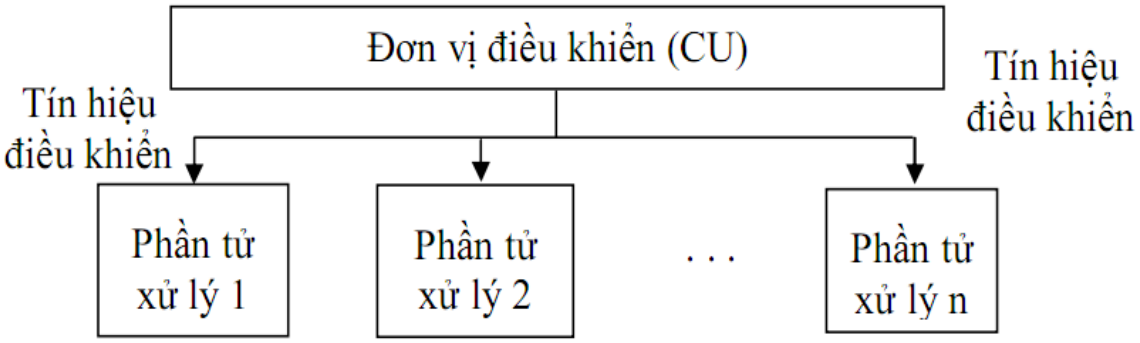
2.1.2. Mô hình SIMD

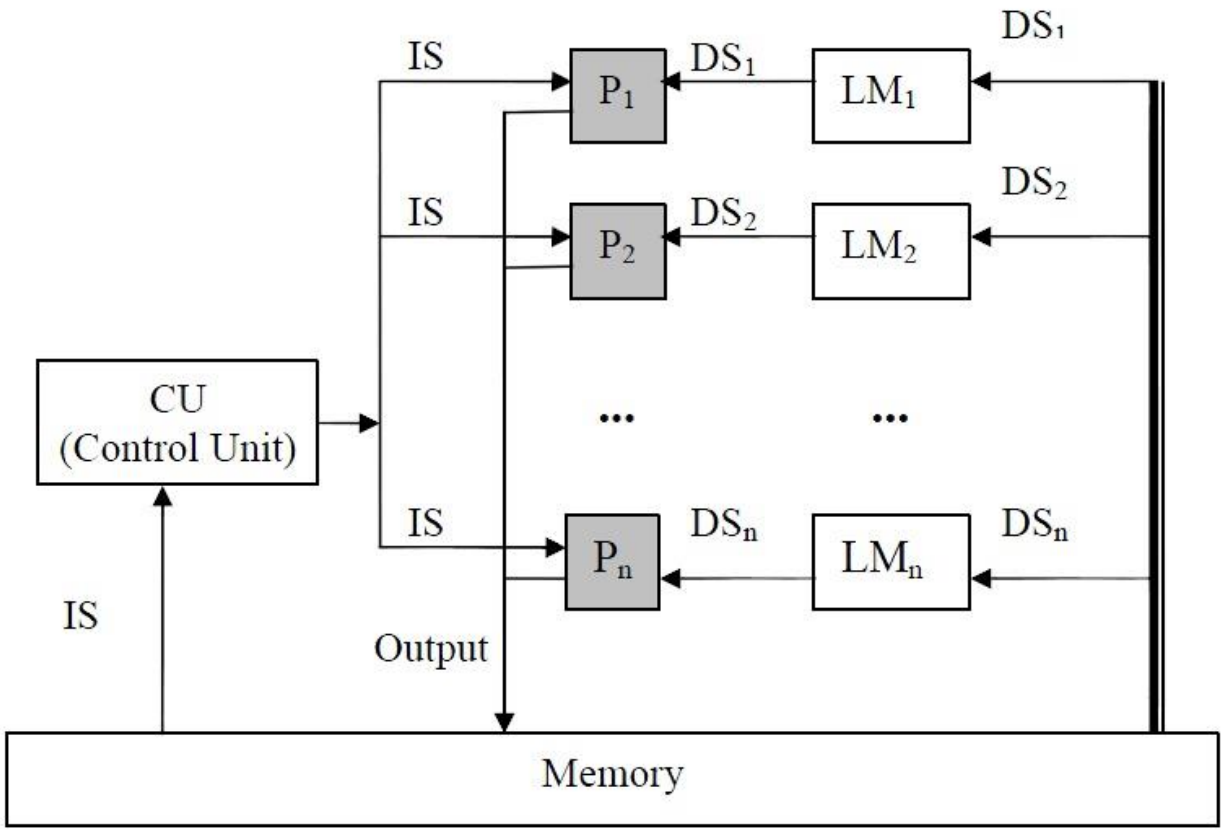
(Single Instruction Stream, Multiple Data Stream - Đơn luồng lệnh, đa luồng dữ liệu)

Máy tính loại SIMD có một đơn vị điều khiển để điều khiển nhiều đơn vị xử lý (nhiều hơn một đơn vị) thực hiện theo một luồng các câu lệnh.

CU phát sinh tín hiệu điều khiển tới tất cả các phần tử xử lý, những BXL này cùng thực hiện một phép toán trên các mục dữ liệu khác nhau, nghĩa là mỗi BXL có luồng dữ liệu riêng.

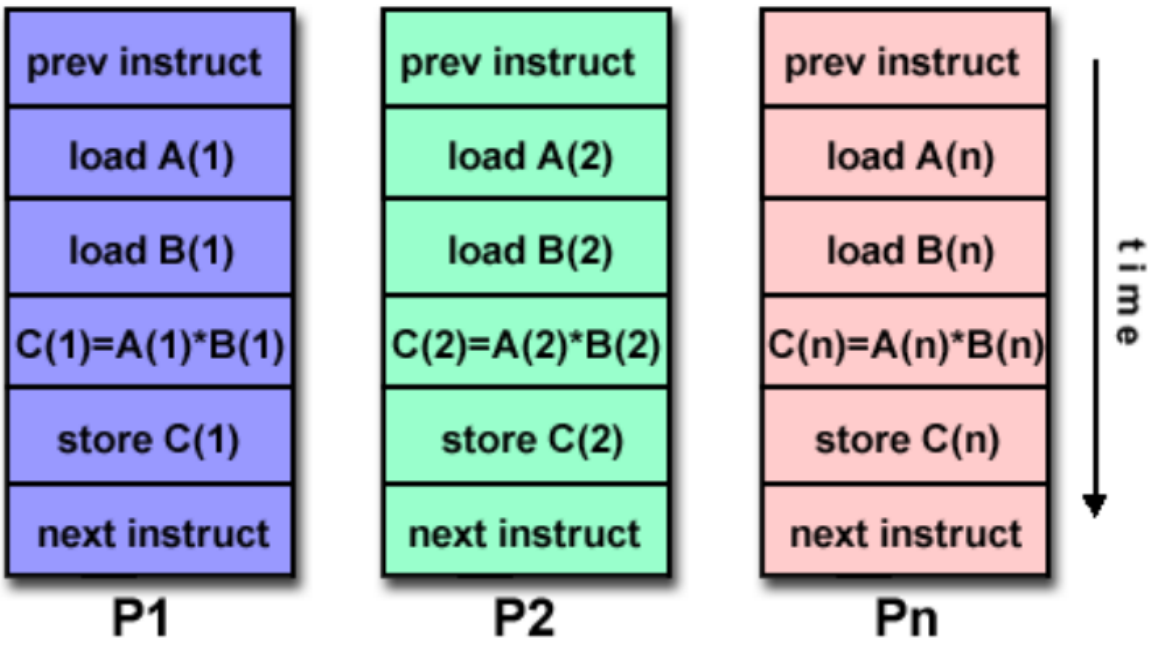
Hình 2-3 mô tả hoạt động của máy tính theo mô hình SIMD, còn được gọi là SPMD.





Hình 2-3: Mô hình của kiến trúc SIMD

Mô hình SIMD còn được gọi là SPMD, đơn chương trình và đa dữ liệu. Đây chính là mô hình máy tính phổ biến có trên thị trường như: DAP và Connection Machine CM-2.



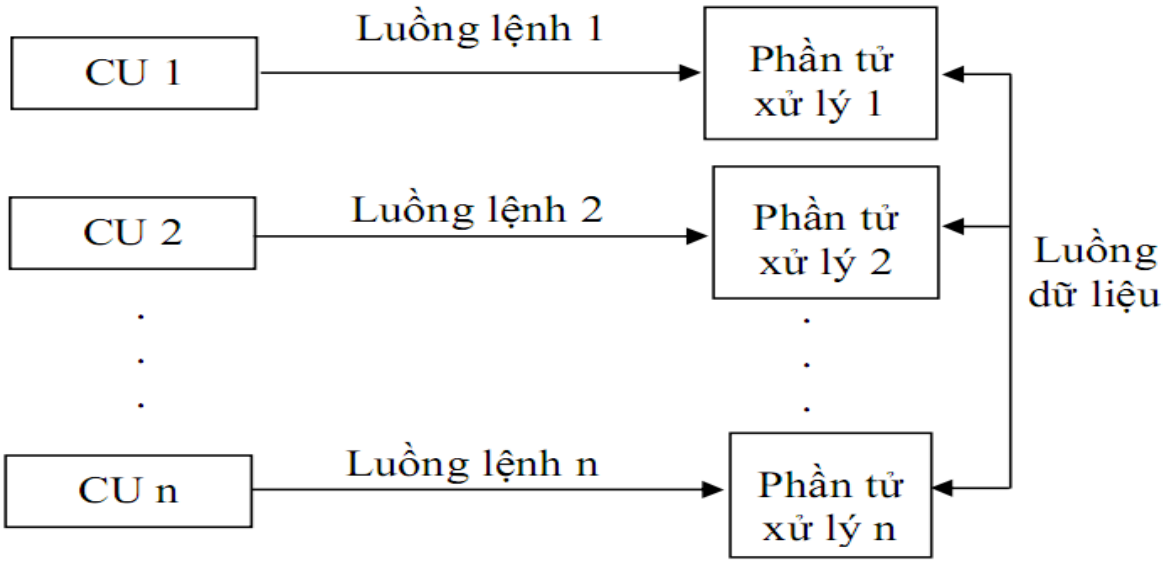
2.1.3. Mô hình MISD

(Multiple Instruction Stream, Single Data Stream – Đa luồng lệnh, đơn luồng dữ liệu)

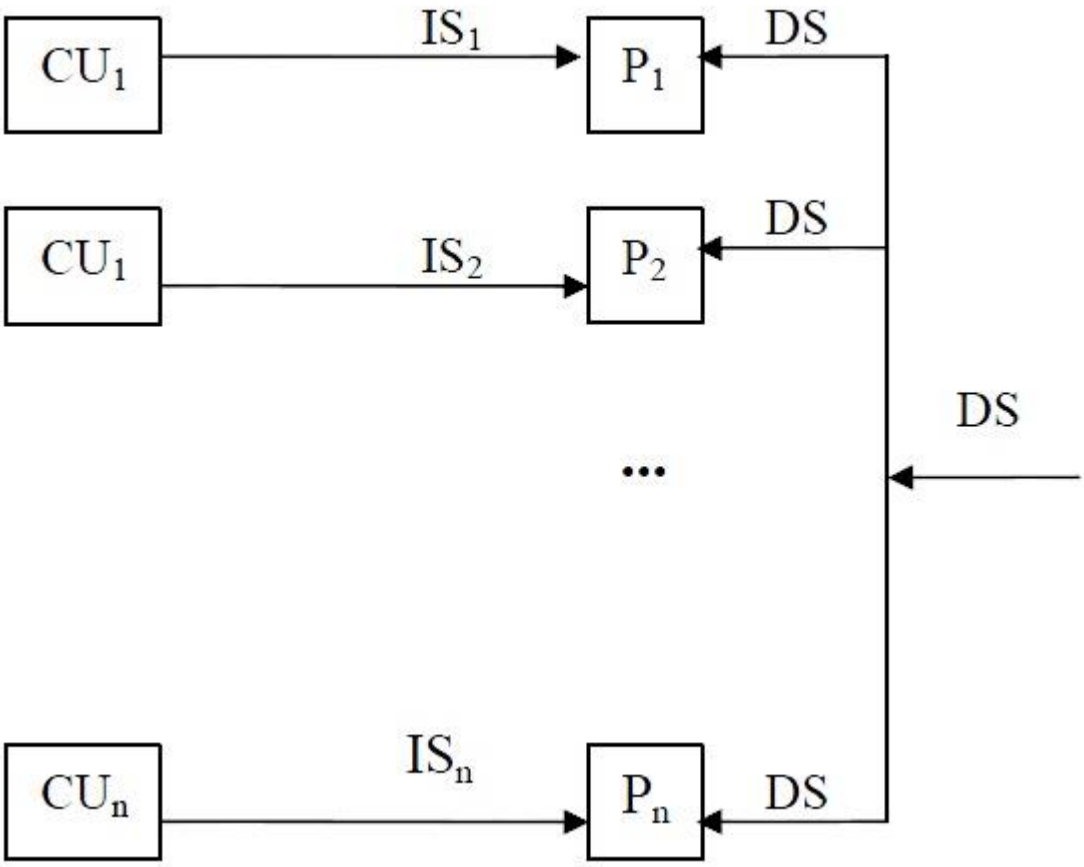
Máy tính loại MISD là ngược lại với SIMD. Máy tính MISD có thể thực hiện nhiều chương trình (nhiều lệnh) trên cùng một mục dữ liệu, nên còn được gọi là MPSD (đa chương trình, đơn dữ liệu). Kiến trúc kiểu này có thể chia thành hai nhóm:

- *Nhóm 1*: lớp các máy tính gồm nhiều đơn vị xử lý (CU) khác nhau có thể nhận được những chỉ lệnh khác nhau để thực hiện trên cùng một mục dữ liệu. Đây là kiến trúc khó thực hiện.

Mô hình của kiến trúc MISD được mô tả như hình 2-4.



Hình 2-4: Mô hình của kiến trúc MISD



- *Nhóm 2*: Lớp các máy tính có các luồng dữ liệu được gửi tuần tự theo dãy các CPU liên tiếp. Đây là loại kiến trúc hình ống, kiến trúc này rất hiệu quả và nhiều ứng dụng sẽ được đề cập tới ở chương sau.

2.1.4. Mô hình MIMD

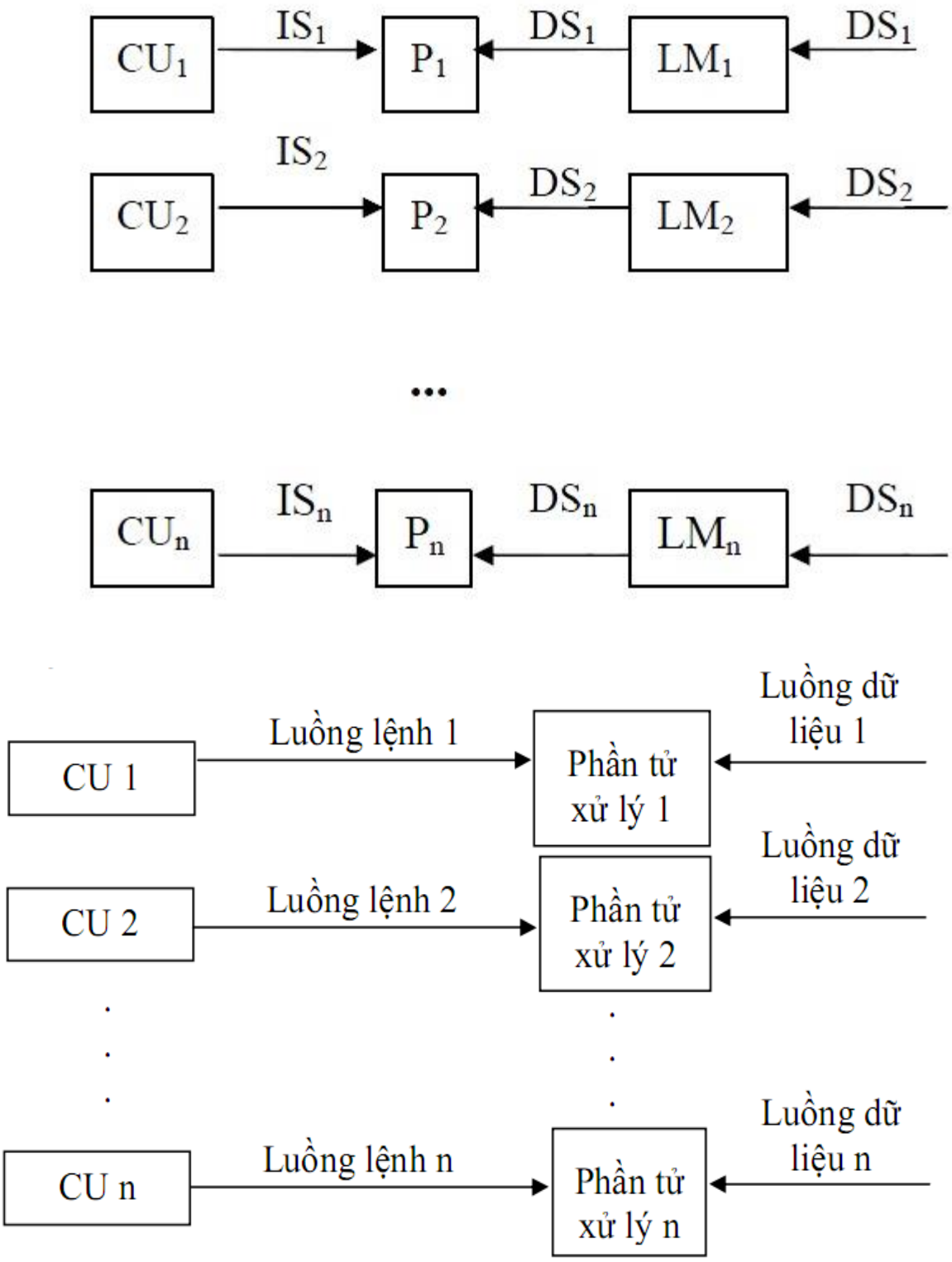
(Multiple Instruction Stream, Multiple Data Stream - Đa luồng lệnh, đa luồng dữ liệu)

Máy tính loại MIMD còn gọi là đa BXL, trong đó mỗi BXL có thể thực hiện những luồng lệnh (chương trình) khác nhau trên các luồng dữ liệu riêng.

Hầu hết các hệ thống MIMD đều có bộ nhớ riêng và cũng có thể truy cập vào được bộ nhớ chung (global) khi cần, do vậy giảm thiểu được thời gian trao đổi dữ liệu giữa các BXL trong hệ thống.

Đây là kiến trúc phức tạp nhất, nhưng nó là mô hình hỗ trợ xử lý song song cao nhất và đã có nhiều máy tính được sản xuất theo kiến trúc này, ví dụ: BBN Butterfly, Alliant FX, iSPC của Intel, v.v.

Mô hình của kiến trúc MIMD được mô tả như hình 2-5.



Hình 2-5: Mô hình của kiến trúc MIMD

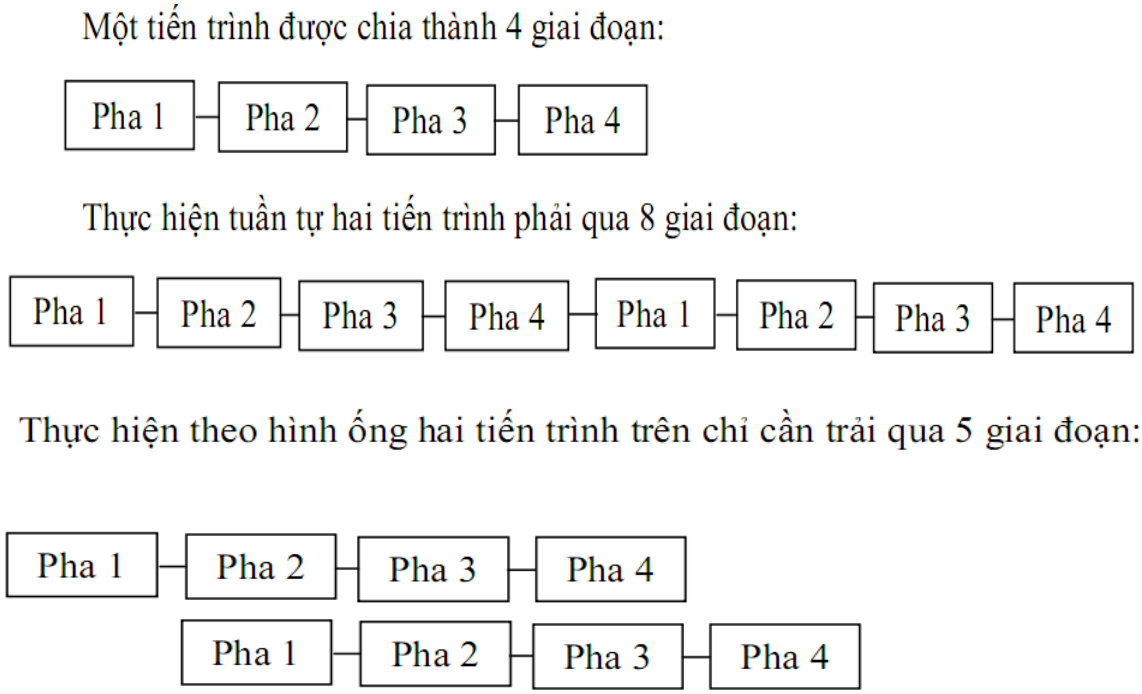
2.2. Kiến trúc máy tính song song

2.2.1. Xử lý theo nguyên lý hình ống trong CPU

Nguyên lý hình ống (pipelined) dựa vào phương pháp phân đoạn hoặc chia nhỏ một tiến trình tính toán thành một số đoạn nhỏ hơn để thực hiện trong các pha liên tiếp. Tất cả các giai đoạn của một tiến trình được thực hiện tuần tự, khi thực hiện xong thì bắt đầu thực hiện của tiến trình tiếp theo. Mỗi pha thực hiện xong sẽ gửi kết quả cho pha tiếp theo.

Như vậy, trong cách thực hiện theo nguyên lý hình ống, khi một giai đoạn công việc đang thực hiện thì một giai đoạn khác có thể nạp dữ liệu vào, và dữ liệu vào của giai đoạn này có thể là kết quả của giai đoạn trước nó.

Ví dụ, hình 2-6 mô tả một tiến trình được phân thành 4 giai đoạn thực hiện tuần tự, nhưng có thể thực hiện song song theo nguyên lý hình ống để tăng tốc độ tính toán khi phải thực hiện nhiều tiến trình như thế.



Hình 2-6: Thực hiện tuần tự và hình ống của hai tiến trình gồm 4 giai đoạn

Nếu ký hiệu Si là thời gian cần thiết để thực hiện giai đoạn thứ i thì:

+ Tổng thời gian tính toán tuần tự là: 2*(S1 + S2 + S3+ S4)

+ Tổng thời gian tính toán hình ống là: S1 + S2 + S3+ S4 + S4

2.2.2. Đa chương trình và chia sẻ thời gian

Các hệ điều hành của máy tính đơn bộ xử lý cho phép thực hiện song song dựa vào cách tiếp cận phần mềm.

Trong cùng một khoảng thời gian, có nhiều tiến trình cùng truy cập vào dữ liệu từ những thiết bị vào/ra chung (VD:Cổng giao tiếp, Đĩa cứng, CD, …). Chúng ta biết rằng phần lớn các chương trình đều có hai phần: phần vào/ra và các thành phần tính toán trong quá trình xử lý.

Các hệ điều hành đa chương trình luân phiên thực hiện các chương trình khác nhau. Để thực hiện việc này HĐH sử dụng Bộ lập lịch chia sẻ thời gian làm nhiệm vụ phân chia CPU cho mỗi tiến trình một khoảng thời gian cố định theo phương pháp quay vòng tròn.

Bằng cách đó, tất cả các tiến trình đều được sẵn sàng để thực hiện trên cơ sở được phép sử dụng CPU và những tài nguyên khác của hệ thống.

Do vậy, về nguyên tắc việc phát triển những chương trình song song trên máy đơn BXL thực hiện được nếu có hệ điều hành cho phép nhiều tiến trình thực hiện, nghĩa là có thể xem hệ thống như là đa bộ xử lý.

2.2.3. Mô hình trừu tượng của máy tính song song

Mục đích: muốn thể hiện được những khả năng tính toán của MTSS mà không quan tâm đến những ràng buộc cụ thể của những máy tính có trong thực tế.

Chú ý: khi xây dựng các thuật toán song song, chúng ta qui ước là phát triển thuật toán cho mô hình trừu tượng này, sau đó ánh xạ sang những máy tính cụ thể với một số các ràng buộc nào đó.

2.2.4. Máy tính truy cập ngẫu nhiên song song PRAM

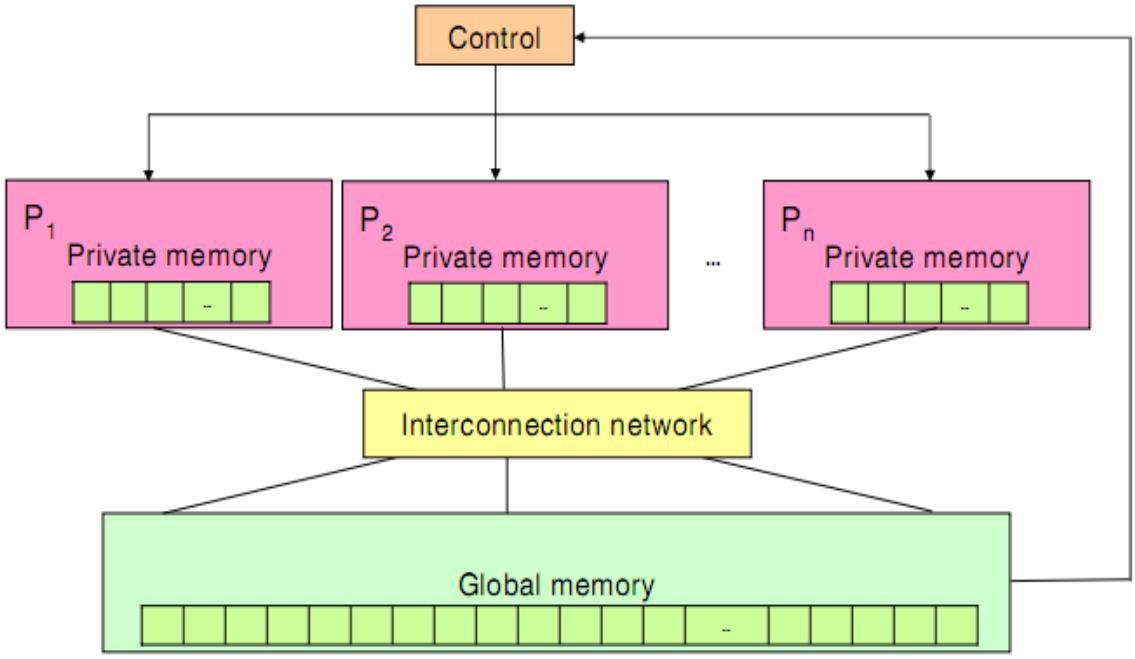
PRAM (Parallel random-access machine)

- Chứa một đơn vị điều khiển CU
- Một bộ nhớ chung
- Một tập không giới hạn các BXL

Mỗi BXL lại có bộ nhớ riêng và có một chỉ số duy nhất được sử dụng để xác định địa chỉ trong quá trình trao đổi các tín hiệu và quản lý các ngắt.

Tất cả các BXL đều chia sẻ bộ nhớ chung với yêu cầu không bị giới hạn. Các câu lệnh có thể bắt đầu thực hiện ở bất kỳ thời điểm nào, ở bất kỳ vị trí nào của bộ nhớ (riêng hoặc chung).

Hình 2-7 mô tả PRAM



Hình 2-7: PRAM

Đây cũng là mô hình tổng quát cho máy tính song song kiểu MIMD

2.2.5. Một số điều cần lưu ý khi phát triển những thuật toán cho các MTSS tổng quát

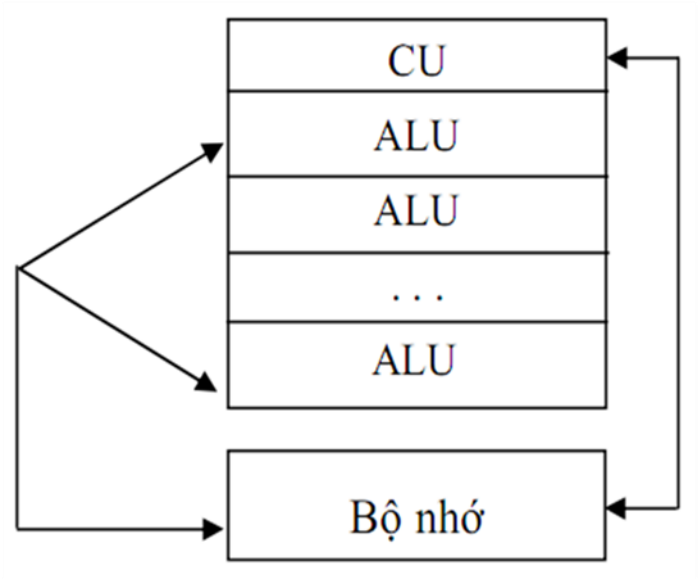
- Không bị giới hạn về số lượng BXL
- Mọi vị trí của bộ nhớ đều truy cập được bởi bất kỳ BXL nào
- Không giới hạn về dung lượng bộ nhớ chung chia sẻ trong hệ thống.
- Các BXL có thể đọc bất kỳ một vị trí nào của bộ nhớ, nghĩa là không cần phải chờ để những BXL khác kết thúc công việc truy cập vào bộ nhớ.

2.2.6. Áp dụng nguyên lý hình ống

*) Nguyên lý hình ống có thể áp dụng theo hai mức:

- *Hình ống theo đơn vị số học*:

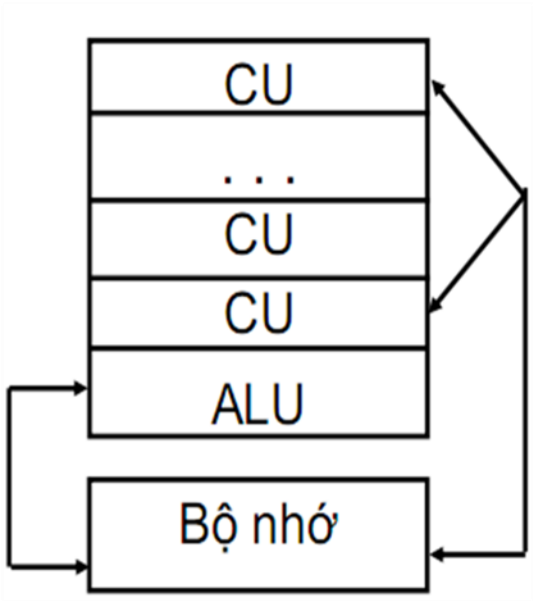
Các đơn vị số học và logic ALU được tổ chức thành mảng, các phép toán bên trong được thực hiện theo nguyên lý hình ống.



Hình 2-8: Xử lý hình ống theo ALU

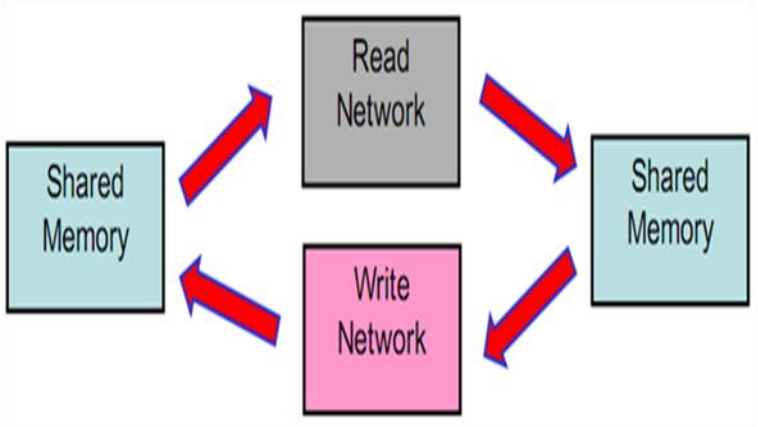
- *Hình ống theo đơn vị câu lệnh*:

Các đơn vị điều khiển CU được phân đoạn và tổ chức theo hình ống.



Hình 2-9: Xử lý hình ống theo CU

*) Xây dựng hình ống vòng tròn giữa các BXL, bộ nhớ và mạng liên kết



Hình 2-10: Ví dụ về một hình ống vòng tròn

Phép toán thực hiện bởi CU theo kiến trúc này có thể chia thành 5 giai đoạn:

GĐ1: Đọc dữ liệu: đọc dữ liệu từ bộ nhớ chia sẻ (Shared Memory).

GĐ2: Chuyển tải dữ liệu: chuyển dữ liệu từ bộ nhớ tới các phần tử xử lý PE thông qua mạng đọc (Read Network).

GĐ3. Thực hiện câu lệnh: sử dụng PE để thực hiện các câu lệnh.

GĐ4. Chuyển tải dữ liệu: chuyển các kết quả từ các PE tới bộ nhớ thông qua mạng ghi (Write Network).

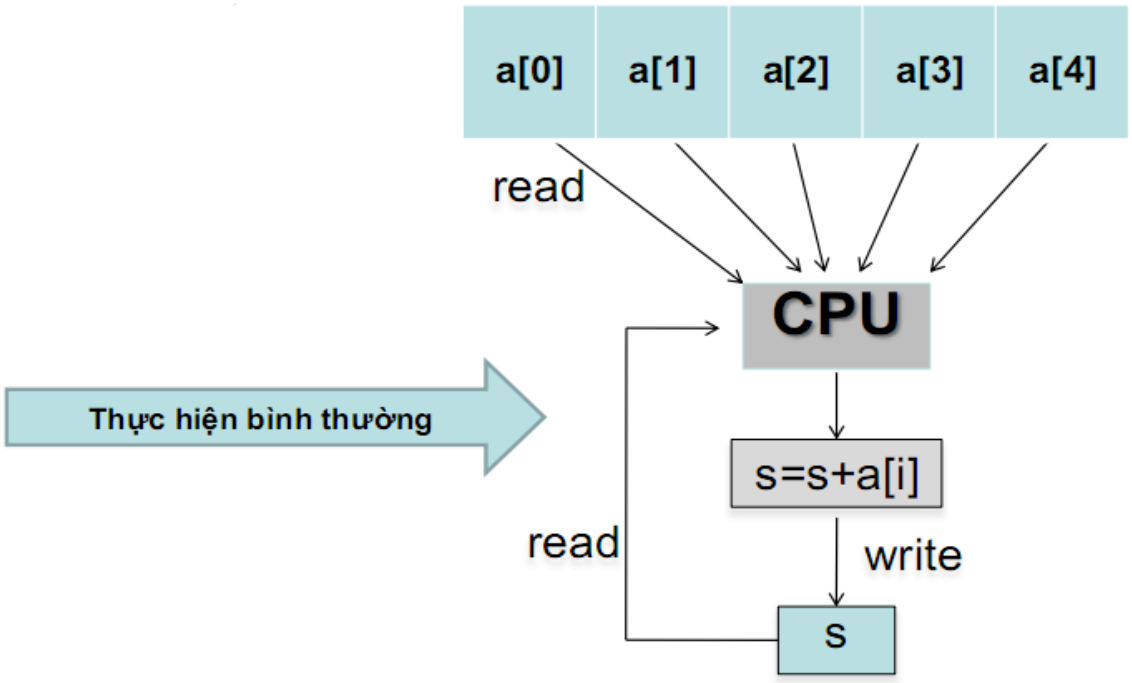
GĐ5. Lưu trữ dữ liệu : ghi lại các kết quả vào bộ nhớ chia sẻ.

*) Một ví dụ đơn giản

Tính tổng n phần tử của một mảng a:

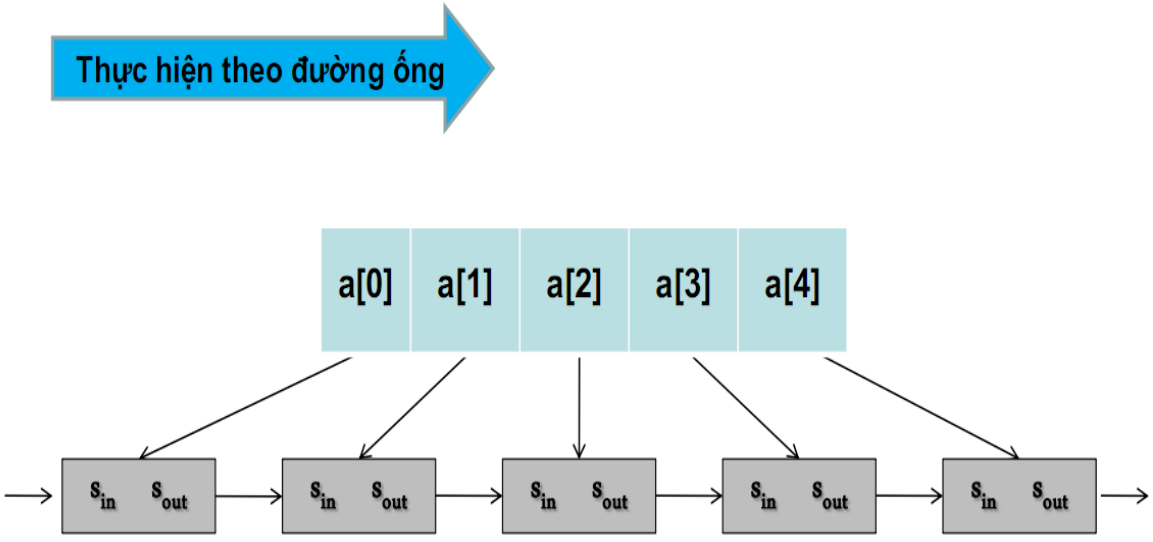
For (i=0; i<n; i++)

s=s+a[i]



Tính tổng n phần tử của một mảng a:

$S_{in} = s_{out} + a[i]$



*) Ví dụ 2: Sắp xếp n phần tử bằng kỹ thuật đường ống

Ý tưởng thuật toán (insertion sort):

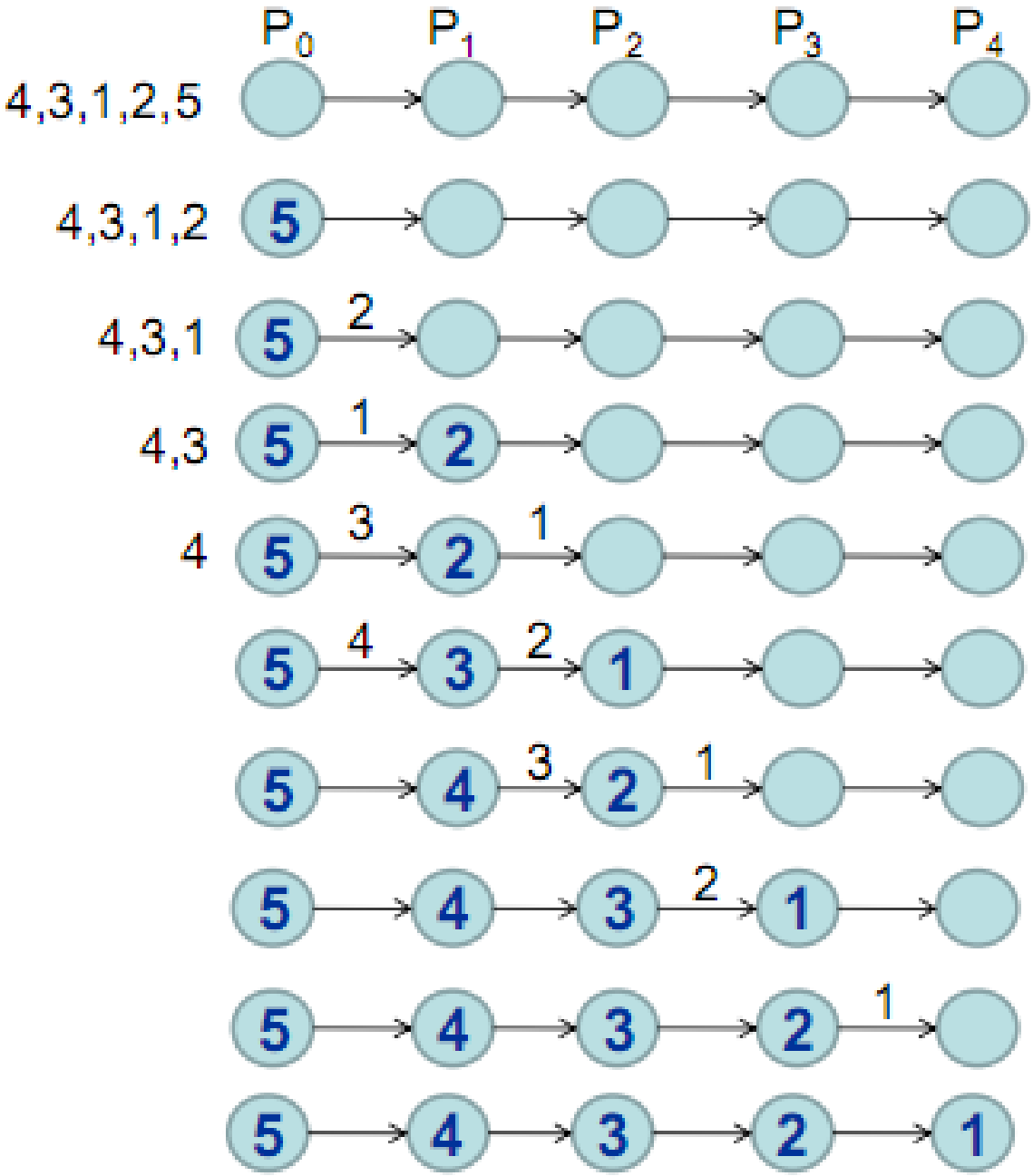
+ P_0 nhận một dãy các số

+ Lưu trữ số lớn nhất

+ Chuyển các số nhỏ hơn đi

+ Nếu số nhận được lớn hơn số lưu trữ hiện thời thì thay thế nó và chuyển số nhỏ hơn đi.

```
Recv(&number, P_{i-1});
If (number > x) {
        send(&x, P_{i-1});
        x= number;
} else send(number, P_{i+1});
```



2.2.7. Bộ xử lý mảng tâm thu SAP (Systolic Array Processor)

Năm 1978 Kung và Leiserson đề xuất một loại kiến trúc được gọi là mảng tâm thu (Systolic Array) cho những tính toán đặc biệt.

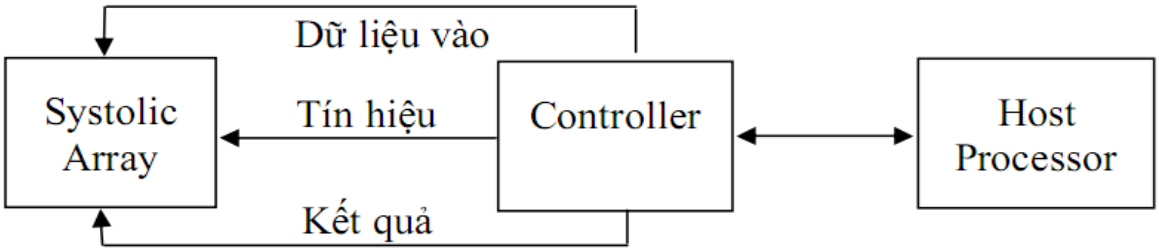
Mảng tâm thu viết tắt là SA, là một mảng các đơn vị xử lý được kết nối cục bộ với nhau.

Trong mảng tâm thu SA, mỗi PE (Processing Element) được xem như một tế bào (một ô trong mảng) bao gồm:

- Một số thanh ghi (register)
- Một bộ cộng (adder)
- Các mạch điều khiển
- Đơn vị logic - số học ALU.

Các ô lân cận có thể trao đổi dữ liệu với nhau.

Dựa vào SA người ta xây dựng kiến trúc SAP.

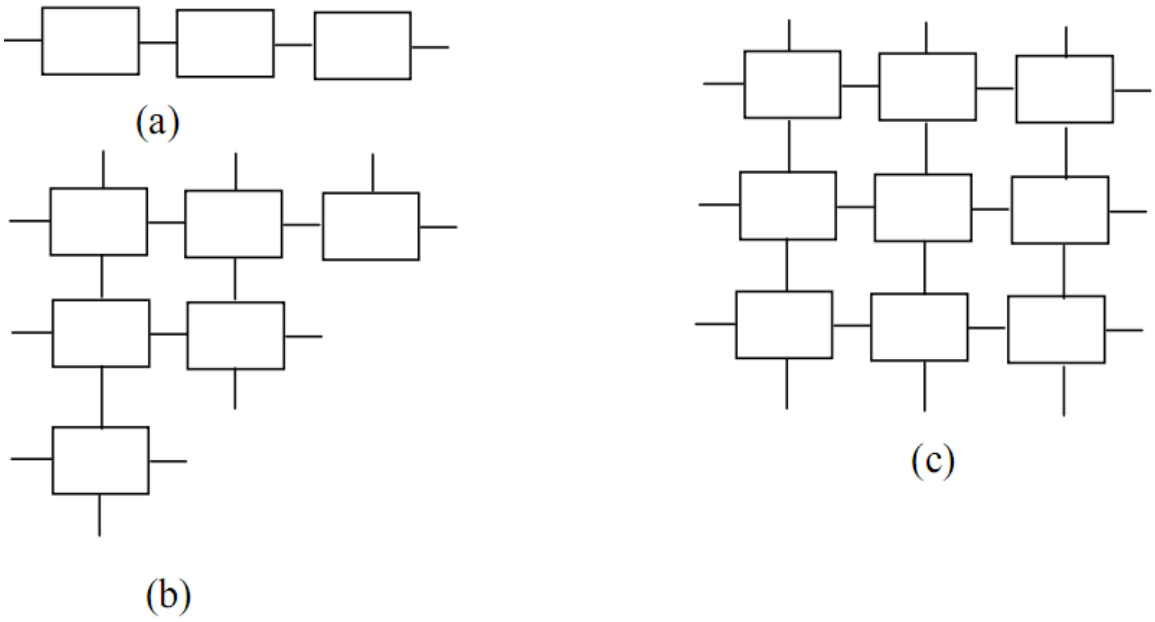


Hình 2-11: Kiến trúc bộ xử lý mảng tâm thu

Trong kiến trúc SAP nêu trên: bộ điều khiển (Controller) làm nhiệm vụ giao diện cho BXL chính (Host Processor) và gửi các tín hiệu điều khiển quá trình vào/ra dữ liệu cho SA.

Hoạt động của hệ thống theo từng nhịp và lặp lại một cách đều đặn để tận dụng được khả năng song song của tất cả các phần tử xử lý.

SA có thể tổ chức theo nhiều cấu hình tôpô khác nhau. Hình 2-12 mô tả một số cấu hình phổ biến của SA.



Hình 2-12: Một số cấu hình phổ biến của mảng tâm thu:

(a) mảng tuyến tính, (b) mảng hình tam giác, (c) mảng hai chiều hình vuông

Hiệu quả của SA phụ thuộc rất nhiều vào các đặc tính vào/ra của dữ liệu. Nó sẽ rất hiệu quả đối với những bài toán mà số liệu đọc/ghi thực hiện với nhịp độ cao, đều đều và liên tục như các bài toán xử lý ảnh, qui hoạch tuyến tính, v.v.

Ví dụ, xét bài toán nhân hai ma trận cỡ 2 × 2: A * B = C

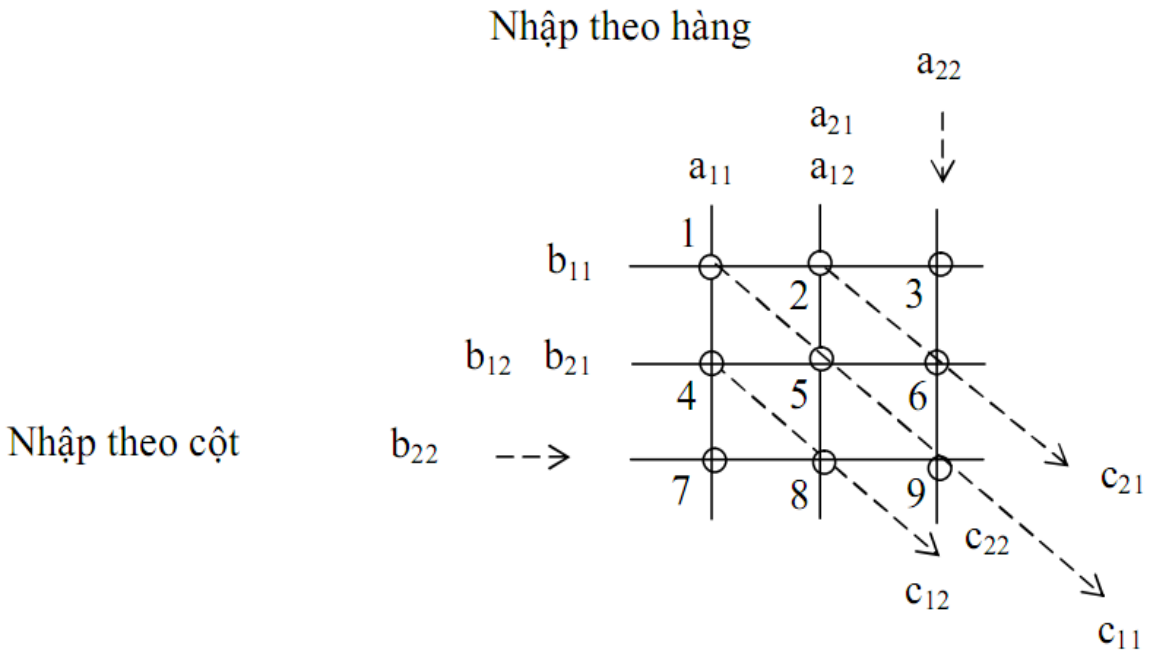
$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

Hiển nhiên $Cij = \sum a_{ik} * b_{kj}$ (k=1,2)

$$c_{11} = a_{11} * b_{11} + a_{12} * b_{21} \qquad c_{12} = a_{11} * b_{12} + a_{12} * b_{22}$$

$$c_{21} = a_{21} * b_{11} + a_{22} * b_{21} \qquad c_{22} = a_{21} * b_{12} + a_{22} * b_{22}$$

Chúng ta có thể thiết kế SA có 9 PE () để thực hiện nhân hai ma trận trên như sau:



Kiến trúc SA để thực hiện nhân hai ma trận.

Năm 1986 Intel kết hợp với Kung đã xây dựng một hệ máy tính kiểu SAP đặt tên là iWrap System, version sau được cải tiến vào năm 1990. Trong những năm 1990 còn có seri máy tính loại mini-super của Convex Computer Corporation được xây dựng từ những bộ CPU 64 bit được gắn với bộ nhớ chung.

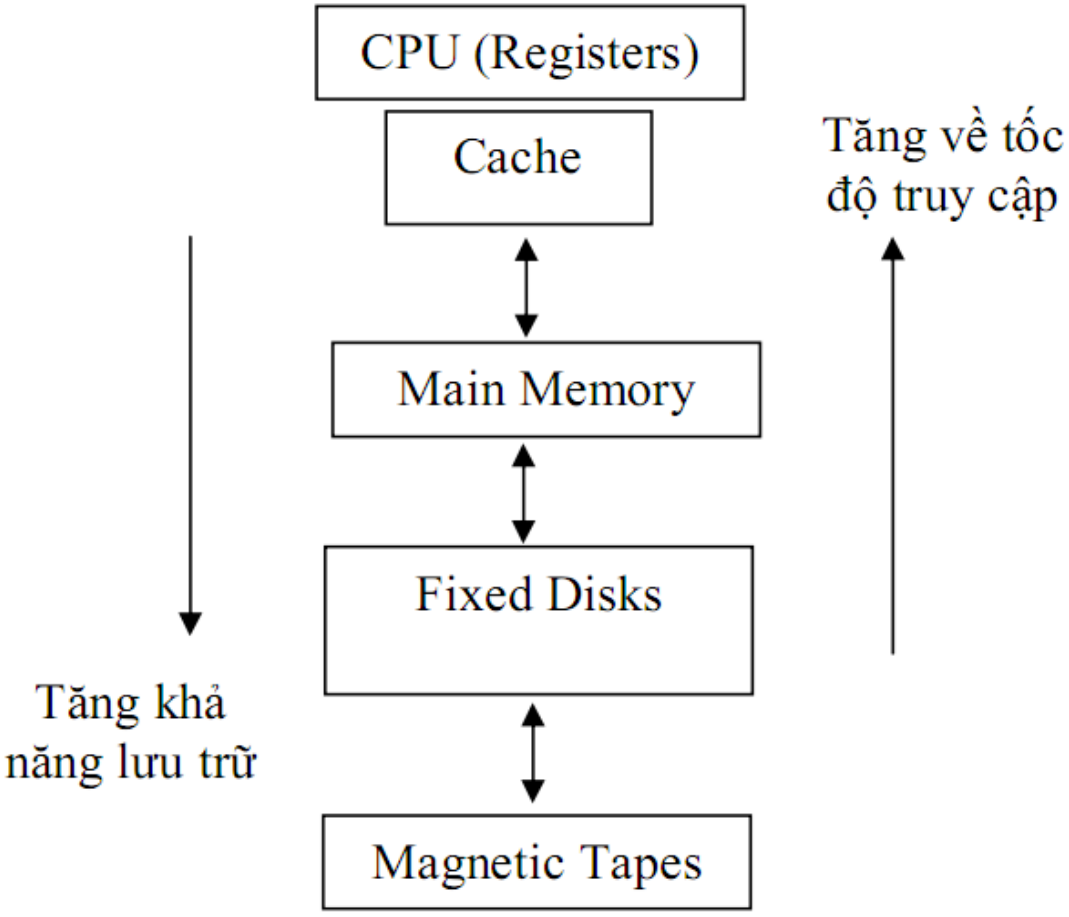
2.3. Bộ nhớ của máy tính song song

2.3.1. Phân cấp hệ thống bộ nhớ

Các hệ thống bộ nhớ phân cấp có tốc độ thực hiện các phép toán trong BXL nhanh hơn rất nhiều so với việc truy cập vào bộ nhớ.

Tốc độ truy cập vào bộ nhớ trong (RAM) nhanh hơn rất nhiều so với việc truy cập vào bộ nhớ ngoài.

Hệ thống bộ nhớ phân cấp như thế có thể mô tả như hình 2-13



Hình 2-13: Hệ thống bộ nhớ phân cấp

Các thanh ghi được sử dụng trực tiếp cho ALU. Bộ nhớ cache được xem như vùng đệm giữa BXL và bộ nhớ chính.

Sự song song hóa trong sự trao đổi dữ liệu theo cấu trúc phân cấp là cách khai thác chung để cải tiến hiệu quả xử lý của hệ thống.

Ví dụ, trong khi dữ liệu được lấy từ bộ nhớ ngoài vào bộ nhớ chính thì đồng thời có thể gửi dữ liệu từ cache vào cho CPU.

2.3.2. Bộ nhớ kết hợp (AM – Associative Memory)

AM bao gồm các ô nhớ (cell). Mỗi ô nhớ của AM có 4 đầu vào và 2 đầu ra. Các đầu vào (input) của mỗi ô nhớ bao gồm:

. Bit đối số *a*

. Bit *đọc/ghi R/W* xác định thao tác tương ứng cần thực hiện

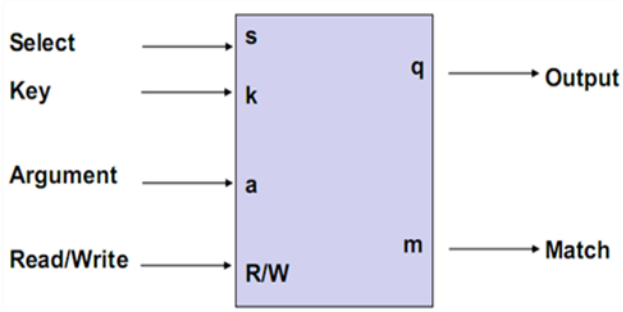
. Bit khoá *k*

. Bit lựa chọn s để xác định ô nhớ thích hợp cho việc thực hiện đọc/ghi.

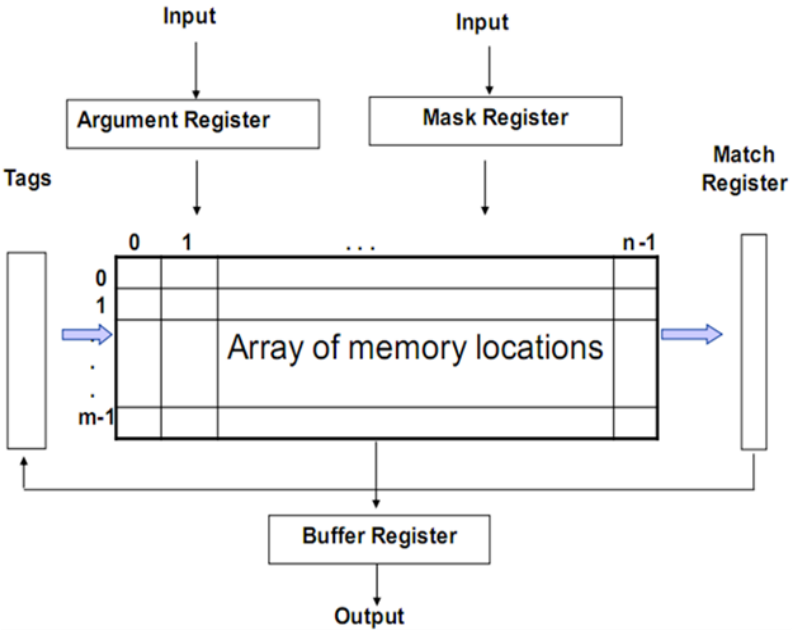
Hai kết quả ở đầu ra:

. Bit đối sánh *m* chỉ ra dữ liệu được lưu trong bộ nhớ có đối sánh được với bit đối số *a*.

. Bit kết quả ra *q*.



Hình 2-14: Mô tả một ô nhớ của AM



Hình 2-15: Mô tả cấu trúc của bộ nhớ kết hợp.

Ví dụ, tìm trong AM tất cả các từ có 8 bit cao nhất chứa giá trị $(1101\ 1100)_2$ và trả lại từ đầu tiên được tìm thấy.

Giá trị $(1101\ 1100)_2$ là đối số để tìm.

Thanh ghi đánh dấu (mask register) là 8 bit cao nhất. Quá trình tìm kiếm thực hiện như sau:

1. Từ cần tìm $(1101\ 1100)_2$ được nạp vào thanh ghi đối số Argument Register.

2. Đoạn mà chúng ta quan tâm là 8 bit cao nhất, những bit này được đưa vào thanh ghi đánh dấu Mask Register để đánh dấu.

3. Tất cả các từ trong bộ nhớ được so sánh song song với thanh ghi đối số.

2.3.3. Bộ nhớ truy cập ngẫu nhiên song song

*) Bộ nhớ PRAM gồm:

- *m* vùng bộ nhớ đủ lớn để chia sẻ cho *p* bộ xử lý.

- được sử dụng để lưu trữ dữ liệu và là vùng để trao đổi dữ liệu giữa các BXL.

- các BXL có thể truy cập vào bộ nhớ PRAM đồng thời và có thể hoạt động một cách dị bộ.

Ví dụ, bộ xử lý P_i ghi dữ liệu vào một vùng nhớ và dữ liệu này có thể được P_j truy cập, nghĩa là P_i và P_j có thể dùng bộ nhớ chung để trao đổi với nhau.

Mô hình loại này có các phương thức truy cập sau:

**)* Các phương thức truy cập bộ nhớ

- Concurrent Read (CR): nhiều BXL có thể đọc đồng thời cùng một ô nhớ.

- Exlusive Read (ER): *p* BXL đọc được *p* vùng nhớ khác nhau và mỗi BXL đọc được duy nhất một vùng nhớ và mỗi vùng nhớ được đọc bởi một BXL.

- Concurrent Write (CW): cùng một thời điểm cho phép nhiều BXL ghi vào cùng một vùng nhớ.

- Exlusive Write (EW): *p* BXL ghi được vào *p* vùng nhớ khác nhau. Mỗi BXL ghi được vào một vùng nhớ và mỗi vùng nhớ được ghi bởi một BXL.

*) Người ta phân CW thành các loại như sau:

- Ghi đồng thời có ưu tiên (Priority CW): các BXL được gắn mức ưu tiên và khi có nhiều BXL muốn ghi dữ liệu vào một vùng nhớ thì ưu tiên cho BXL có mức ưu tiên cao nhất. Các mức ưu tiên có thể gắn tĩnh hoặc động theo những qui tắc được xác định khi thực hiện.

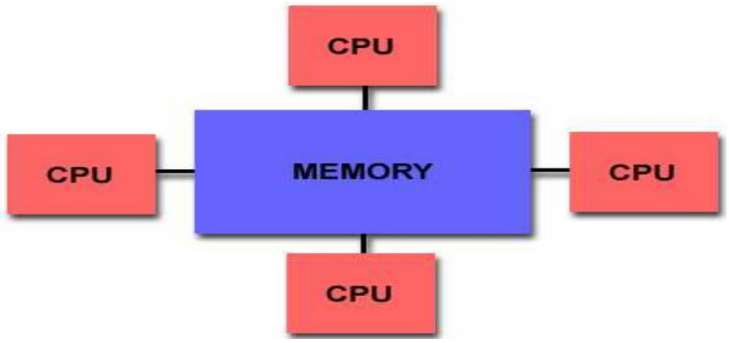
- Ghi đồng thời chung (Common CW): Khi các BXL ghi cùng một giá trị thì chúng được phép ghi vào cùng một vùng nhớ. Trong trường hợp này, một BXL sẽ được chọn để ghi dữ liệu đó.

- Ghi đồng thời tự do (Arbitrary CW): một số BXL muốn ghi dữ liệu đồng thời vào một vùng nhớ, nhưng có một BXL được phép thay đổi giá trị của vùng nhớ đó. Trong trường hợp này, chúng ta phải chỉ ra cách để lựa chọn BXL thực hiện việc ghi.

- Ghi đồng thời ngẫu nhiên (Random CW): BXL được lựa chọn để ghi dữ liệu là ngẫu nhiên.

- Ghi đồng thời tổ hợp (Combining CW): tất cả các dữ liệu mà các BXL định ghi đồng thời vào bộ nhớ được tổ hợp lại thành một giá trị. Giá trị này sẽ được ghi vào bộ nhớ đó.

2.3.4. Bộ nhớ chia sẻ (Share Memory)



Đặc trưng:

- Dung lượng lớn

- Các BXL đều có thể truy cập vào SM

- Các BXL có thể hoạt động độc lập nhưng luôn chia sẻ địa chỉ các ô nhớ (không đọc độc quyền)

- Những thay đổi nội dung ô nhớ được thực hiện bởi một BXL nào đó sẽ được nhìn thấy bởi các BXL khác.

- Các máy tính sử dụng bộ nhớ chia sẻ được phân làm 2 loại:

+ UMA (Uniform Memory Access)

+ NUMA (NonUniform Memory Access)

a. Mô hình UMA của bộ nhớ chia sẻ

Đặc điểm:

- Các BXL làm việc nhờ cơ chế chuyển mạch tập trung (central switching mechanism) để điều khiển việc truy cập tới bộ nhớ chia sẻ.

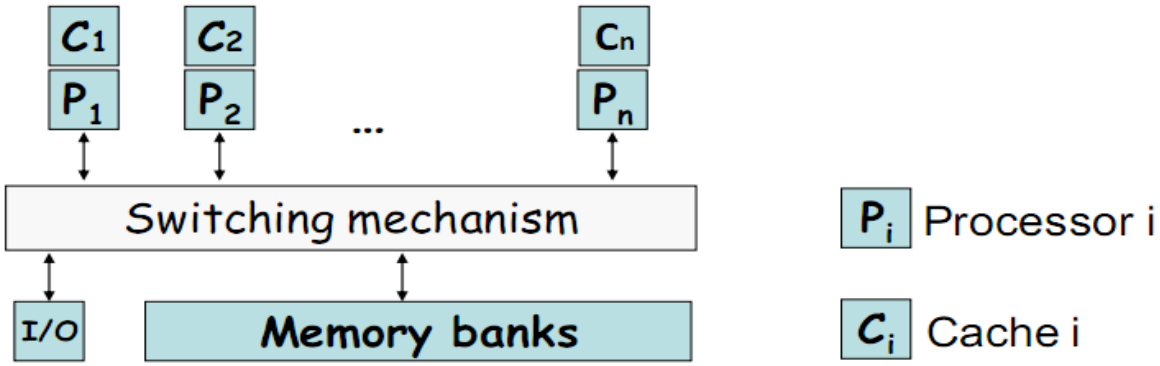
- Các BXL đều có thể truy cập đến bộ nhớ chia sẻ toàn cục (global shared memory)

- Thời gian truy cập vào bộ nhớ là như nhau đối với mọi BXL.

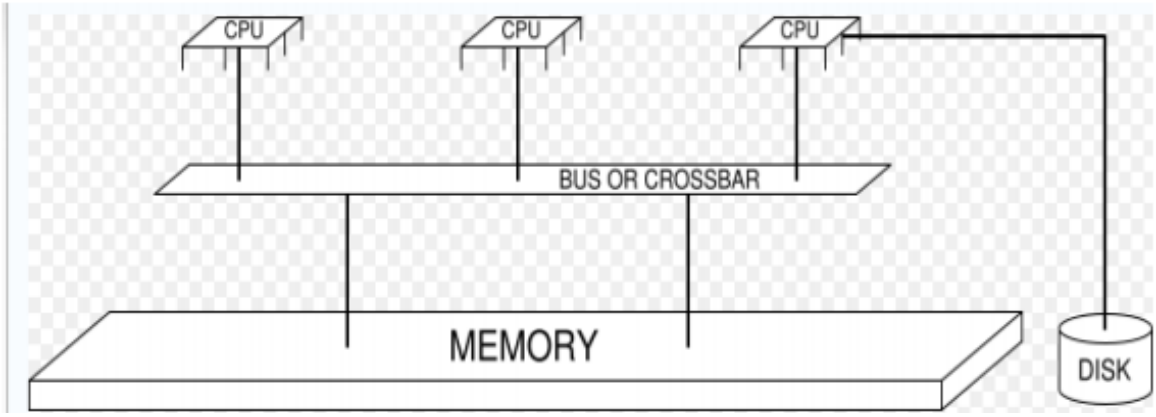
- Một BXL này có thể trao đổi thông tin với một BXL khác bằng cách ghi thông tin vào bộ nhớ toàn cục và BXL kia sẽ đọc dữ liệu tại cùng vị trí đó trong bộ nhớ.

- Nếu tất cả các BXL của máy tính đều có thời gian truy cập đến các thiết bị là như nhau thì gọi là máy tính đa bộ xử lý đối xứng - Symmetric MultiProcessor (SMP) machines

- Máy tính với kiến trúc UMA còn được gọi: CC-UMA – Cache Coherent UMA



Hình 2-16a: Mô hình UMA (1)



Hình 2-16b: Mô hình UMA (2)

b. Mô hình NUMA của bộ nhớ chia sẻ

Đặc điểm:

- Bộ nhớ chia sẻ được phân tán và chia thành các mođun nhớ độc lập.

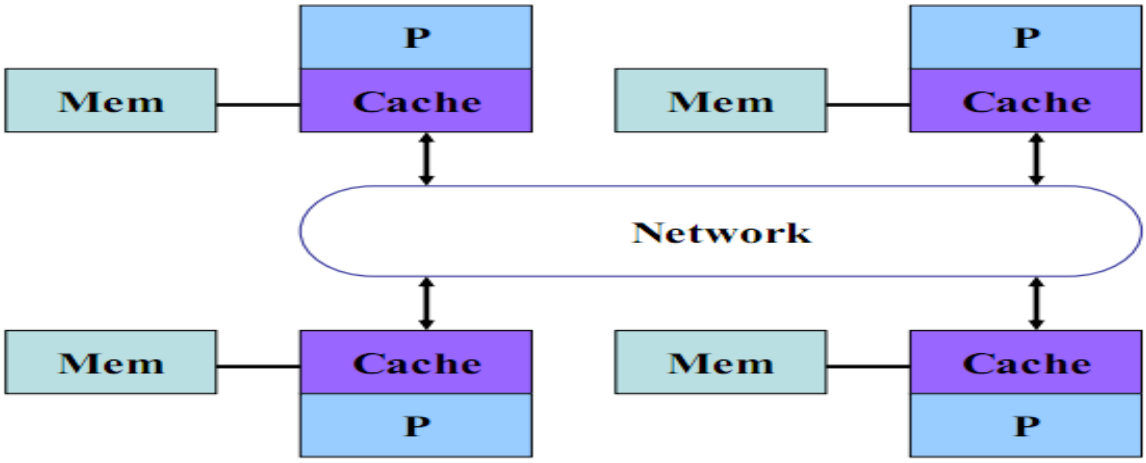
- Bộ nhớ chia sẻ được phân tán cho tất cả các BXL thành bộ nhớ cục bộ và tất cả các mođun nhớ sẽ là bộ nhớ chung cho các BXL.

- Mô hình NUMA thường được tạo thành từ hai hoặc nhiều SMP nối với lại với nhau bởi một đường truyền vật lý.

- Một SMP có thể truy cập trực tiếp đến một SMP khác.

- Các BXL được phép truy cập đồng thời tới một hay nhiều mô đun nhớ và có thể hoạt động độc lập với nhau.

- Không phải tất cả các BXL đều có thời gian truy cập đến các bộ nhớ là như nhau. Truy cập bộ nhớ qua link sẽ chậm hơn.



Hình 2-17: NUMA với Distributed Memory

2.3.5. Bộ nhớ phân tán

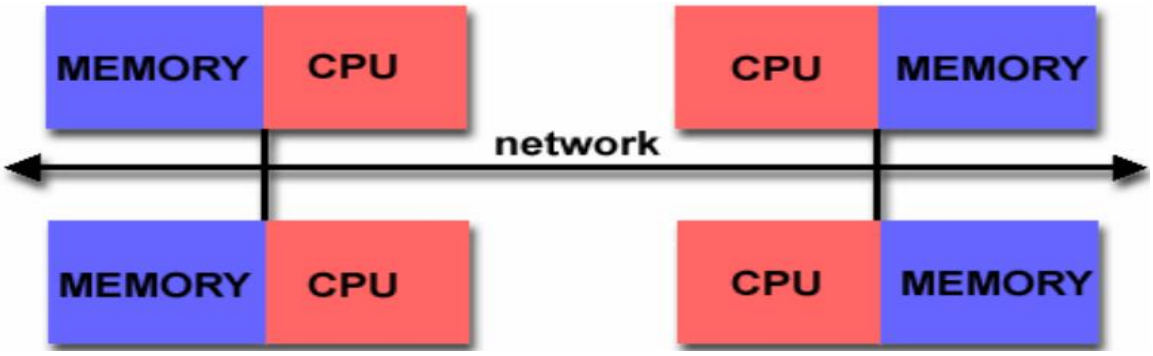
Đặc tính chung của kiến trúc bộ nhớ phân tán :

- Trong mô hình bộ nhớ phân tán, mỗi CPU có bộ nhớ cục bộ riêng. Địa chỉ bộ nhớ của một CPU không được ánh xạ sang CPU khác, do đó không có khái niệm bộ nhớ tổng thể trong mô hình bộ nhớ phân tán. Hệ thống sử dụng kết nối mạng để trao đổi dữ liệu giữa bộ nhớ của các CPU.

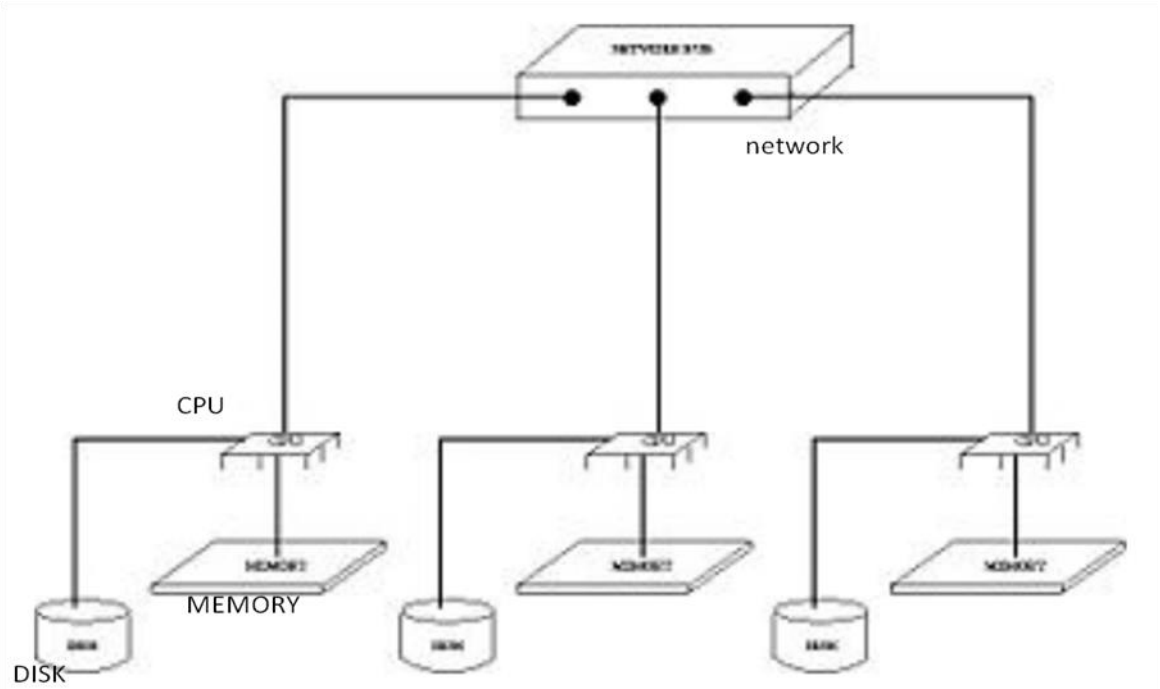
- Do mỗi CPU có bộ nhớ cục bộ riêng được thao tác một cách độc lập, sự thay đổi nội dung bộ nhớ của mỗi CPU không ảnh hưởng đến bộ nhớ của các CPU khác. Do đó không có khái niệm cache-coherent trong mô hình này.

- Khi một CPU cần truy cập dữ liệu của một CPU khác, người lập trình cần chỉ rõ một cách tường minh phương thức truy cập dữ liệu. Người lập trình phải chịu trách nhiệm về việc đồng bộ hóa giữa các tác vụ song song.

- Kết nối mạng cho các hệ thống máy tính bộ nhớ phân tán rất đa dạng như Myrinet, cáp quang, Ethernet …



Hình 2-18a: Kiến trúc bộ nhớ phân tán. Ký hiệu CPU: bộ xử lý, MEMORY: bộ nhớ, network: kết nối mạng.



Hình 2-18b: Kiến trúc bộ nhớ phân tán

*) Lợi ích của mô hình bộ nhớ phân tán :

- Bộ nhớ có thể mở rộng theo số lượng CPU. Số lượng CPU tăng lên hoàn toàn có thể tỷ lệ với tăng dung lượng bộ nhớ.

- Mỗi CPU có khả năng truy cập nhanh tới bộ nhớ cục bộ mà không bị ảnh hưởng bởi các CPU khác, không chịu phí tổn cho các cache-coherent.

- Máy tính mô hình bộ nhớ phân tán có giá thấp và có thể sử dụng các loại phần cứng thông dụng như máy tính cá nhân PC, mạng Ethernet cáp đồng … để lắp ráp.

*) Các bất lợi của mô hình bộ nhớ phân tán :

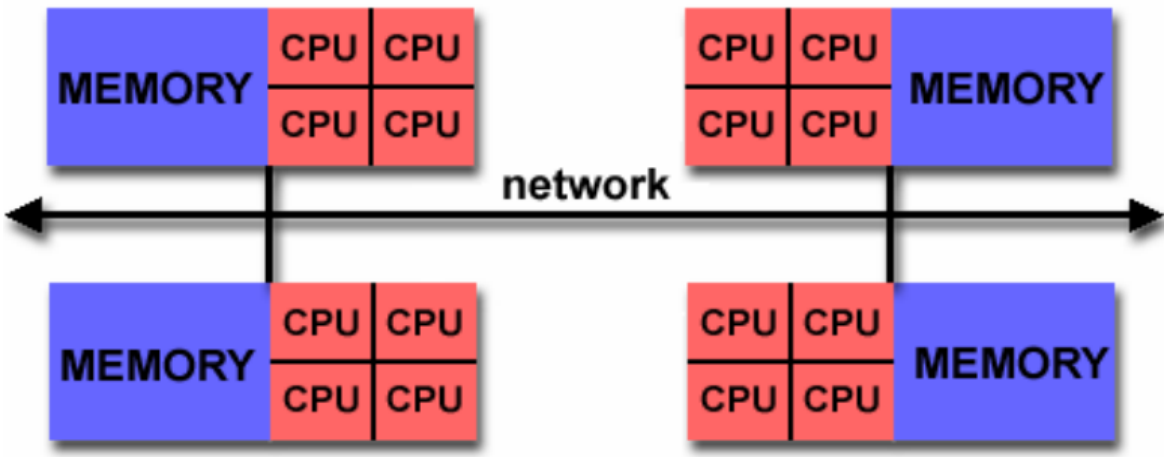
- Người lập trình phải chịu trách nhiệm hoàn toàn về truyền dữ liệu giữa các CPU.

- Khó ánh xạ các cấu trúc dữ liệu dựa trên bộ nhớ tổng thể vào bộ nhớ phân tán.

- Truy cập đến bộ nhớ không đồng nhất.

2.4. Máy tính với bộ nhớ lai

Kiến trúc bộ nhớ lai là sự kết hợp của kiến trúc chia sẻ bộ nhớ chung và phân tán.



Hình 2-19: Kiến trúc bộ nhớ lai. Ký hiệu CPU: bộ xử lý, MEMORY: bộ nhớ, network: kết nối mạng.

Các siêu máy tính nhanh nhất thế giới hiện nay thường được thiết kế theo kiến trúc lai.

Hình 2-19 minh họa kiến trúc này, trong đó thành phần máy tính có bộ nhớ chung thường được thiết kế theo kiểu SMP sử dụng cấu trúc cache-coherent. Thành phần phân tán bao gồm nhiều thành phần bộ nhớ chung được kết nối qua đường mạng.

Xu hướng hiện nay cho thấy kiến trúc máy tính theo kiểu lai giữa bộ nhớ phân tán và bộ nhớ chung đang chiếm ưu thế trong việc thiết kế xây dựng các siêu máy tính.

Chương 3 - MẠNG LIÊN KẾT GIỮA CAC BỘ XỬ LÝ VÀ BỘ NHỚ

3.1. Các loại cấu hình tôpô cho mạng liên kết

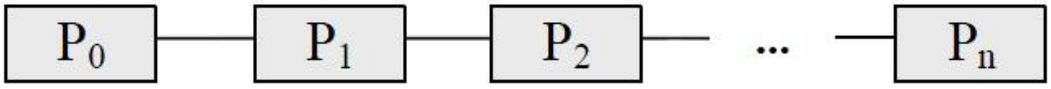
- ***Mạng liên kết tĩnh*** là mạng các thành phần của hệ thống máy tính, trong đó các BXL, bộ nhớ được liên kết với nhau một cách cố định, không thay đổi được.
- ***Mạng liên kết động*** là mạng các thành phần của hệ thống máy tính, trong đó sự liên kết giữa các BXL, bộ nhớ có thể thay đổi được.

3.2. Mạng liên kết tuyến tính

Mô hình đơn giản nhất với các đặc điểm: mỗi bộ xử lý P_i ($0<i<n$) có hai láng giềng là P_{i-1} và P_{i+1},

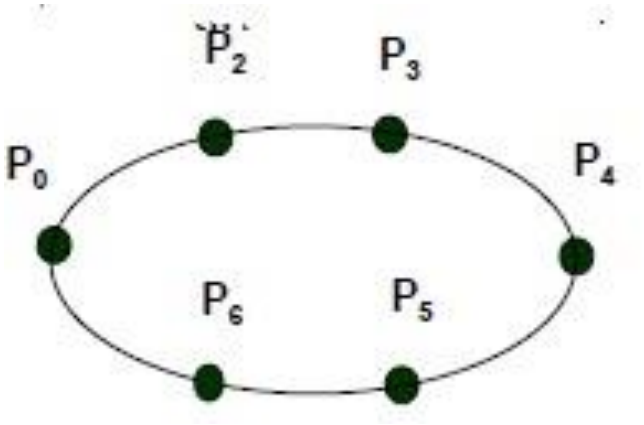
- Các bộ xử lý đầu và cuối chỉ có một láng giềng
- Một đường truyền lần lượt chạy qua các bộ xử lý
- Thông tin có thể chạy trên đường truyền theo một chiều (mạng đơn) hoặc theo cả hai chiều (mạng kép)

Dữ liệu gửi từ bộ xử lý này tới bộ xử lý kia phải lần lượt đi qua các bộ xử lý trung gian nên thời gian bị chậm, tối đa có thể phải qua (n-1) trung gian. Mạng liên kết tuyến tính còn một dạng khác là mạng liên kết vòng, trong đó bộ xử lý cuối và bộ xử lý đầu tiên kết nối trực tiếp với nhau để giảm số bộ xử lý trung gian xuống dưới n/2



3.3. Mạng liên kết vòng

- Được tổ chức tương tự như liên kết tuyến tính nhưng BXL đầu và cuối được nối vòng với nhau

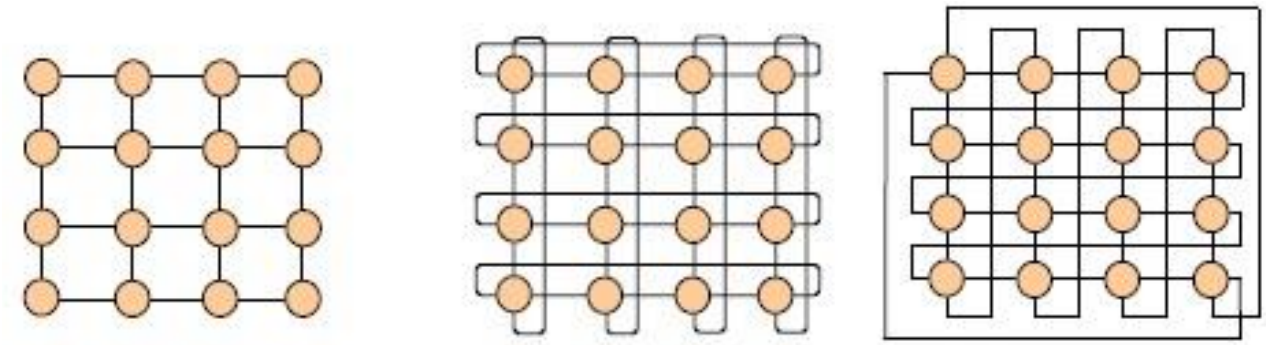


✦ Đặc điểm:

- Sự trao đổi giữa các BXL có thể thực hiện theo một chiều, gọi là mạng đơn, hoặc theo cả hai chiều gọi là mạng kép.
- Sự truyền thông trong mạng liên kết vòng, nhất là giữa những BXL ở xa nhau thì cũng vẫn bị trễ.

3.4. Mạng liên kết lưới hai chiều

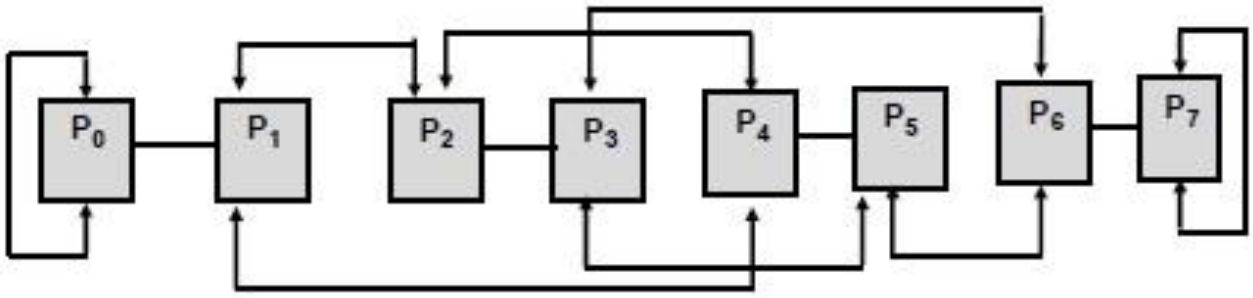
Mạng liên kết lưới hai chiều được sử dụng trong các máy ILLIAC IV, MPP, DAP. Trong mạng liên kết này, mỗi bộ xử lý được kết nối trực tiếp với 4 láng giềng: trên, dưới, phải, trái. Đặc trưng quan trọng của mạng không quay vòng là các đường nối có độ dài không đổi.



(a) Lưới không quay vòng (b) Lưới quay vòng

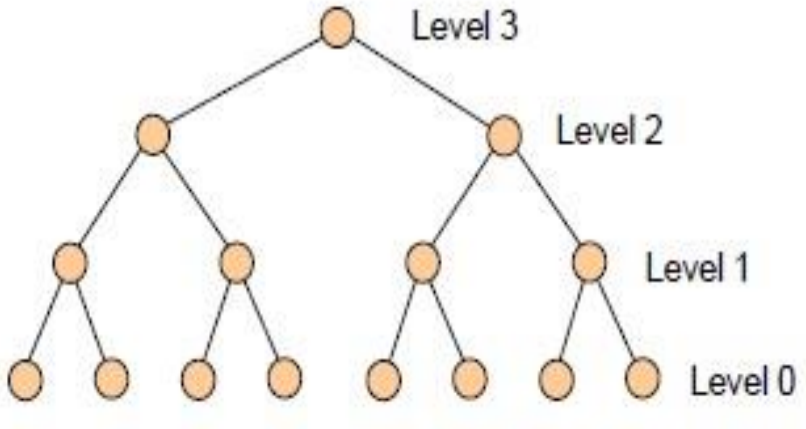
3.5. Mạng liên kết xáo trộn hoàn hảo hai chiều

Giả sử có n bộ xử lý đánh số từ P_0, P_1, ... P_{n-1} với n là lũy thừa của 2. Mạng liên kết xáo trộn gồm những đường liên kết một chiều từ P_i tới P_j.

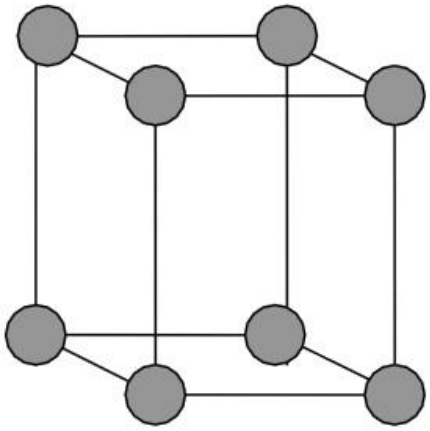

P0
P1
P2
P3
P4
P5
P6
P7

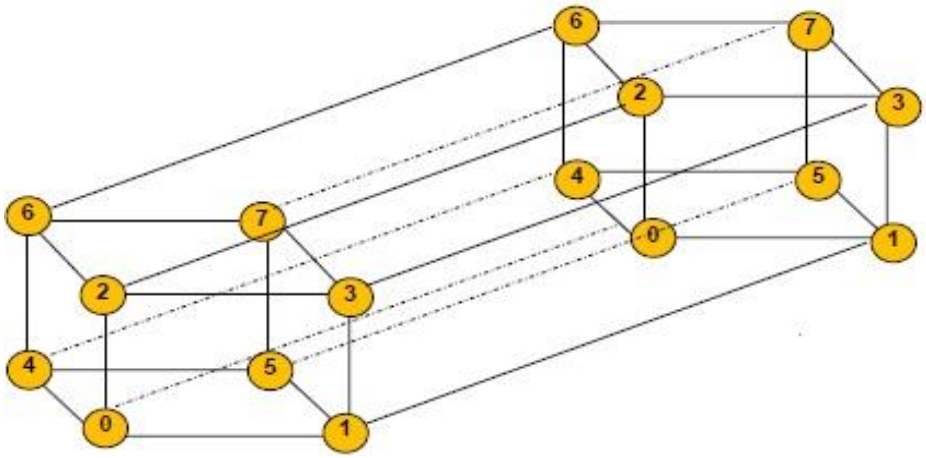
3.6. Mạng liên kết nhị phân (binary tree)

Các bộ xử lý liên kết với nhau theo một mạng hình cây nhị phân, tỏa ra từ nút gốc. Mỗi bộ xử lý nối trực tiếp với các bộ xử lý ở nút mẹ và hai nút con.



3.7. Mạng liên kết khối hộp (cube connection):





Mỗi bộ xử lý có d láng giềng với d là số chiều của liên kết

Chương 4 - GIỚI THIỆU LẬP TRÌNH SONG SONG

4.1. Giới thiệu chung

Để cài đặt các thuật toán song song trên các máy tính song song chúng ta phải sử dụng những ngôn ngữ lập trình song song. Nhiều ngôn ngữ lập trình song song đang được sử dụng như: Fortran 90, Pthread/C++, Fortran/C++, MPI với C/C++, PVM với C/C++, OpenMP với C/C++, v.v….

Việc cố gắng chuyển mã từ tuần tự sang song song dựa trên tác vụ là quá trình này cần rất nhiều thời gian, có khi cả vài năm trời. Do tốc độ gia tăng số nhân/lõi hiện tại quá nhanh, nên nhiều khi thuật toán song song mà chúng ta đã bỏ ra vài năm kể từ năm nay để thiết kế lại không mở rộng tốt với số nhân/lõi của vài năm nữa, ở thời điểm chúng ta hoàn tất việc cài đặt nó một cách song song. Ý tưởng gán một công việc nhỏ cho từng bộ xử lý/nhân/lõi trong máy tính nhằm tăng hiệu năng phần mềm chạy trên đó là hoàn toàn tự nhiên và được đúc kết từ kinh nghiệm trong đời sống hàng ngày.

Tuy nhiên để tận dụng được hết sức mạnh của hàng trăm, rồi hàng ngàn nhân/lõi trong tương lai, chúng ta có lẻ cần phải "đi trước đón đầu" bằng cách tạo ra hàng chục ngàn, hàng trăm ngàn công việc nhỏ li ti cung cấp cho đội quân nhân/lõi đông đảo và ngày một gia tăng đó.

4.2. Giới thiệu các ngôn ngữ lập trình cho xử lý song song

4.2.1. Pthread Posix

POSIX (Portable Operating System Interface) threads. POSIX (chính xác hơn là chuẩn IEEE 1003.1c của tổ chức IEEE đưa ra) bao gồm những định nghĩa "giao diện" chung cho các hệ điều hành. Điều đó có nghĩa là những hệ điều hành nào support POSIX (GNU/Linux, BSD, Sun Solaris, Unix, …) thì đều có những system call có prototype, mặc dù đối với mỗi hệ điều hành có cách implement khác nhau.

POSIX threads (Pthreads) là một cách rất tốt để làm tăng độ tin cậy và performance cho chương trình.

Threads cũng tương tự như processes, đều được phân chia thời gian bởi kernel. Với hệ thống chỉ có một bộ vi xử lý thì kernel sử dụng cách phân chia thời gian để "làm cho" các threads như là chạy đồng thời theo cùng cách thức kernel thực hiện với processes. Và với các hệ thống đa nhân thì các threads thực sự có thể chay đồng thời giống như là nhiều processes.

Pthreads không cần phải những lời gọi expensive và phực tạp bởi vì threads "sống cùng một ngôi nhà". Bạn không phải đẩy dữ liệu thông qua một file hoặc một vùng nhớ nào đó. Chính vì lý do này mà nên cân nhắc mô hình một process/nhiều threads hơn là nhiều process/một thread.

Tạo một thread nhanh hơn rất nhiều (từ 10 cho đến 100 lần) so với tạo một process. Kernel không cần phải tạo một bản copy độc lập của không gian bộ nhớ hiện thời, bảng các mô tả file (file descriptors). Điều đó tiết kiệm rất nhiều thời gian của CPU.

4.2.2. MPI

MPI là một bộ thưviện hỗ trợ cho việc lập trình kiểu message passing.

- Thư viện MPI bao gồm các thủ tục truyền tin kiểu point-to-point , và các toán hạng chuyển dữ liệu, tính toán và đồngbộ hóa.
- MPI(1) chỉ làm việc trên các chu trình tĩnh, tất cả các chu trình cần phải đượcđịnhnghĩa trướckhi thực hiện và chúng sẽ đượcthực hiện đồngthời.
- MPI-2 là phiên bản nâng cấp của MPI, có thêm các chức năng có thể đápứng cho các chu trình động, kiểu server-client…
- Trong một chương trình ứng dụng, lập trình viên đưa thêm một số lệnh điều khiển link đến các hàm/thủ tục của bộ thưviện MPI. Mỗi một tác vụ trong chương

trình đượcphân hạng (rank) hay đánhsố bằng các số nguyên từ 0đến*n*-1. *n*là tổng số các tác vụ.

Các tác vụ MPI dựa trên các rank đóđểphân loại message gửi và nhận, sau đó áp dụng các toán hạng phù hợp đểthực hiện các tác vụ.

Các tác vụ MPI có thể chạy đồngthời trên cùng một processor hoặc trên các processor khác nhau.

4.2.3. OpenMP

OpenMP là một giao diện lập trình ứng dụng (API) được sử dụng để điều khiển các luồng trên cấu trúc chia sẻ bộ nhớ chung. Thành phần của OpenMP bao gồm :

1. Các chỉ thị biên dịch (Compiler Directives)

2. Các thư viện runtime (Runtime Library Routines)

3. Các biến môi trường (Emviroment Variables) .

Các chỉ thị biên dịch, các thư viện runtime và các biến môi trường này được sử dụng để lập trình song song với hai ngôn ngữ Fortran và C/C++. OpenMP là một chuẩn bộ nhớ chia sẻ hỗ trợ bởi nhiều nền phần cứng và phần mềm như là DEC, Intel, IBM, SGI, Numerical Algorithms Group. Hơn thế nữa OpenMP còn rất khả chuyển và có thể thực thi trên cả môi trường UNIX và Windows NT

OpenMP ra đời với mục tiêu cung cấp một chuẩn chung cho rất nhiều kiến trúc và nền tảng phần cứng. Nó thiết lập một tập các chỉ thị biên dịch hỗ trợ việc lập trình song song trên máy tính chia sẻ bộ nhớ chung. Một mức song song chính thường được thực thi với ba đến bốn chỉ thị. OpenMP ra đời giúp cho việc lập trình song song một cách dễ dàng nó cung cấp khả năng song song hóa chương trình tuần tự mà không dùng đến thư viện thông điệp v.v...

Có thể sử dụng OpenMP để giải quết các vấn đề giới hạn về thời gian như bàitoán dự báo thời tiết, và để mô phỏng các vấn đề thực tế như bài toán mô phỏng tai nạn xe hơi, giải quyết các bài toán khoa học yêu cầu khối lượng tính toán lớn như bài toán mô phỏng N-Body, dự báo thời tiết …

4.2.4. 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/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.*

- PVM Parallel Virtual Machine (máy ảo song song) được dùng để chỉ một máy tính logic có bộ nhớ phân tán
- PVM cung cấp các thủ tục để khởi tạo các task trên máy ảo (virtual machine)và cho phép các task này trao đổi với nhau
- Task trên hệ thống PVM được coi là một đơn vị tính toán, có ý nghĩa như một UNIX process

4.3. Các mô hình lập trình song song

4.3.1. Mô hình bộ nhớ chia sẻ

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.* 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.

4.3.2. Mô hình truyền thông điệp

- 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.

4.3.3. Mô hình kết hợp

Mô hình kết hợp là mô hình kết hợp của lập trình song song chia sẻ bộ nhớ và lập trình song song truyền thông điệp để có thể giải quyết công việc hiệu quả hơn.

4.4. Lập trình song song Pthread Posix

4.4.1. Cách cài đặt PTHREAD POSIX với VC++ 6

- Cách biên dịch

+ Sao chép thư viện của hàm pthread có trong 2 thư mục: **Pre-built.2; pthread.2**

+ Vào menu option của VC++6 chọn **Tools->Option->Directories** thiết lập thư mục như sau: pthread\Pre-built.2**include**; pthread\Pre-built.2**lib**

+ Sao chép tệp tin pthreadVC2.dll vào thư mục theo đường dẫn sau: C:\WINDOWS\system32

- Chạy chương trình:

Cấu hình project trong VC++ theo các bước sau:

Bước 1: Chọn Project ->Settings -> Link ->

Bước 2: Thêm pthread.lib vào ô có tiêu đề **Object/library modules**

4.4.2. Giới thiệu các hàm chính của pthread

- Tạo thread

int pthread_create (pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg) ;

Trong đó:

thread: Một định danh ID duy nhất cho thread mới

attr: Lưu trữ các tham số của thread mới, mặc định là Null

start_routine: Địa chỉ của chương trình con mà chúng ra cần chạy trong thread mới này

arg: Các tham số của chương trình con, mặc định là NULL

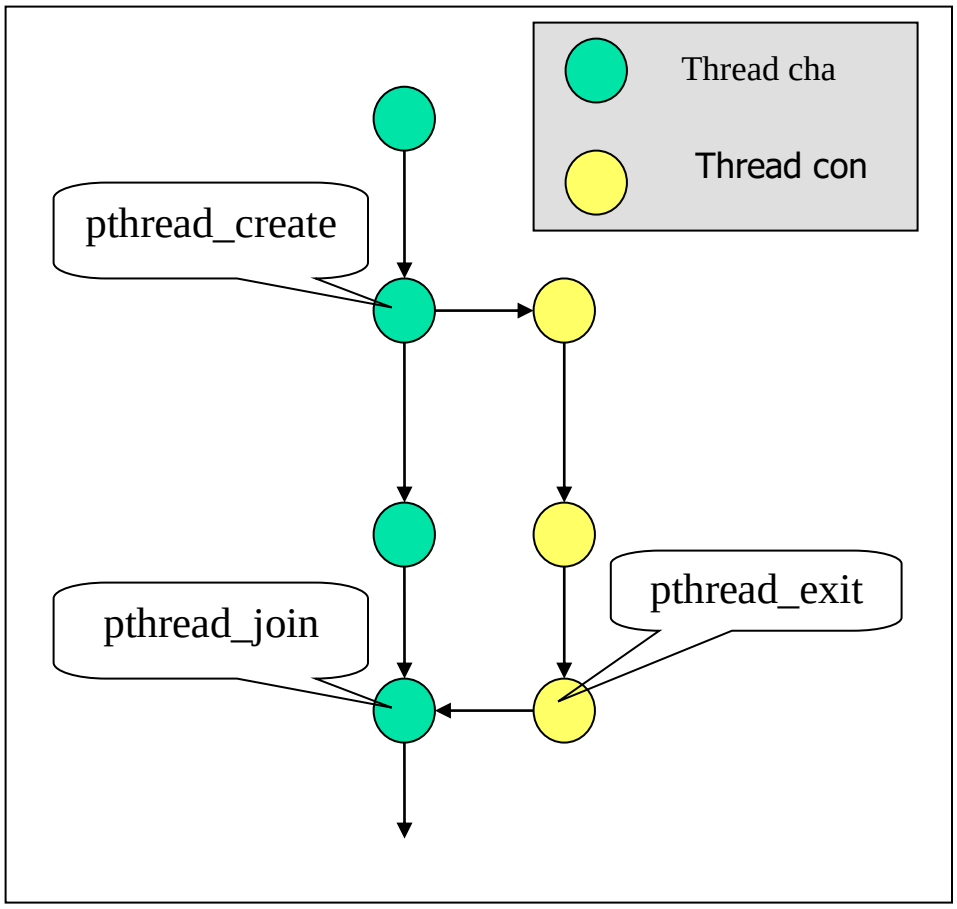
- Kết thúc thread

void pthread_exit (void *value_ptr);

- Thu hồi và kết thúc thread

int pthread_join(pthread_t inHandle, void **outReturnValue);

Thực hiện việc thu hồi bộ nhớ và các thiết của thread có định danh inHandle. Thread cha bị block, và phải chờ cho đến khi thread inHandle kết thúc.



4.4.3. Ví dụ về lập trình trong Pthread Posix

Chương trình chính (thread chính) tạo ra một thread con để làm một việc gì đó cho mình. Trong ví dụ này chương trình sẽ in ra chữ "Hello, World!", trong đó thread con in "Hello, ", còn thread chính in "World!".

```
//1. Khai bao thu vien
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
// 2. Viet ham con tro; Se goi cac ham nay khi chung ta tao thread
void *print_hello(void *args)
{
        printf("Hello, ");
```

```
        fflush(stdout);
        return(0);
}
// 3. Chuong trinh chinh
void main()
{// 3.1 Khai bao cac bien kieu pthread_t
  pthread_t tid;
 //3.2 Tao thread thuc hien cac nhiem ⇔ cac ham con tro pthread_create()
  if (pthread_create(&tid, NULL, print_hello, NULL))
 {
       perror("pthread_create() failed");
       exit(1);
  }
//3.3 Dong bo hoa thread chinh voi cac thread con pthread_join()
  if (pthread_join(tid, NULL))
       {perror("pthread_join() failed"); exit(1); }

  printf(" World!\n");
  fflush(stdout);
  exit(0);
}
```

Chương 5 - THUẬT TOÁN SONG SONG

5.1. Khái niệm thuật toán song song

Những thuật toán, trong đó có một số thao tác có thể thực hiện đồng thời được gọi là *thuật toán song song.*

5.2. Các thuật ngữ

5.2.2. Speedup

Speedup (hệ số tăng tốc): speedup của thuật toán song song là tỷ số giữa thời gian thực hiện trong tình huống xấu nhất của thuật toán tuần tự tốt nhất và thời gian thực hiện cũng công việc đó của thuật toán song song. Trong trường hợp thông thường, có thể hình dung một cách tương đối như sau:

$$Speedup = \frac{\text{thời gian thực hiện trên máy tính tuần tự}}{\text{thời gian thực hiện trên máy tính song song}}$$

5.2.2. Efficiency

Hiệu quả (**Efficiency**) của thuật toán song song được tính bằng

$$\textbf{Efficiency} = \frac{\textbf{Speedup}}{\textbf{Số bộ xử lý tham gia tính toán}}$$

5.2.3. Flop

Flop: một đơn vị đo tốc độ của máy tính song song. Flop là viết tắt của *floating point operations per second :* số phép tính toán hạng số thực dấu phảy động thực hiện được trong một giây

5.2.4. Scalability

Tính quy mô (**Scalability**): một thuật toán được gọi là có tính quy mô nếu như mức độ song song hóa tăng lên theo tỷ lệ ít nhất là tuyến tính với sự tăng lên của kích thước bài toán.

5.2.5. Cost

Giá (cost) của một quá trình tính toán trên PRAM

- Giá = Độ phức tạp tính toán × Số lượng bộ xử lý tham gia tính
- Trong đó *độ phức tạp tính toán* hay *thời gian tính* chính là số bước thực hiện thao tác cơ bản (với giả thiết rằng thực hiện một thao tác cơ bản tốn một đơn vị thời gian)

5.2.6. Thuật toán song song *Tối ưu về chi phí*

Thuật toán song song *Tối ưu về chi phí* (Cost optimal parallel algorithm) là thuật toán có chi phí (độ phức tạp, thời gian tính) tương đương với thuật toán tuần tự tốt nhất.

5.3.Cơ chế điều khiển song song, dữ liệu song song và pepiline

5.3.1. Cơ chế *dữ liệu song song* (Data Parallel):

Cơ chế *dữ liệu song song* (Data Parallel): nhiều bộ xử lý đồng thời áp dụng cùng một thao tác lên nhiều mục dữ liệu khác nhau

5.3.2. Cơ chế *dây chuyền* (Pipeline*):*

Cơ chế *dây chuyền* (Pipeline*)*: nhiều bộ xử lý đồng thời áp dụng nhiều thao tác khác nhau (công đoạn chế biến) lên các phần khác nhau của một mục dữ liệu

5.3.3. *Cơ chế điều khiển song song (Control Parallel)*

Cơ chế *điều khiển song song* (Control Parallel): nhiều bộ xử lý đồng thời áp dụng nhiều thao tác khác nhau lên nhiều mục dữ liệu khác nhau

Cơ chế *dữ liệu song song*	nhiều bộ xử lý đồng thời thực thi	cùng một	thao tác lên	nhiều mục dữ liệu khác nhau
Cơ chế pipeline		nhiều		các phần khác nhau của một mục dữ liệu
Cơ chế *điều khiển song song*		nhiều		nhiều mục dữ liệu khác nhau

5.4. Câu hỏi đặt ra trước khi thiết kế thuật toán song song?

1.Việc phân chia dữ liệu cho các tác vụ như thế nào?

2.Dữ liệu được truy cập như thế nào,

3.Những dữ liệu nào cần phải chia sẻ?

4.Phân các tác vụ cho các tiến trình (bộ xử lý) như thế nào?

5.Các tiến trình được đồng bộ hóa ra sao?

5.5. Cách thiết kế thuật toán song song

1. Thực hiện song song hoá những thuật toán tuần tự: Biến đổi những cấu trúc tuần tự để tận dụng được những khả năng song song tự nhiên của tất cả các thành phần trong hệ thống xử lý.

2.Thiết kế những thuật toán song song mới phù hợp với kiến trúc song song.

3.Xây dựng những thuật toán song song từ những thuật toán song song đã được xây dựng cho phù hợp với cấu hình tôpô và môi trường song song thực tế.

5.6 Một số giải thuật song song

5.6.1 Nhân ma trận với vec tơ.

Trong phần này, chúng ta đề cập đến việc nhân một ma trận với một vec tơ. Để đơn giản hóa bài toán, chúng ta giả sử có một ma trận vuông A cấp n(ma trận có số hàng bằng số cột và bằng n) và một vec tơ cột*x* có n dòng.

Tích của ma trận A(n x n) và vec tơ *x*(n x 1) là một vec tơ *y*(n x1).

Bây giờ chúng ta đi tìm hiểu giải thuật tuần tự để tính tích của một ma trận và một vec tơ.

a. Giải thuật tuần tự.

Ta có: $y[i] = \sum_{j=0}^{n-1} A(i,j) * x(j)$

Vì vậy, ta có giải thuật tuần tự như sau:

```
1.   procedure MAT_VECT ( A, x, y)
2.   begin
3.      for i := 0 to n - 1 do
4.      begin
5.         y[i]:=0;
6.         for j := 0 to n - 1 do
7.             y[i] := y[i] + A[i, j] x x[j];
8.      endfor;
9.   end MAT_VECT
```

Giải thuật tuần tự cần thực hiện n^2 phép nhân và phép cộng trong quá trình tính toán. Giả sử mỗi phép toán thực hiện tốn 1 đơn vị thời gian, khi đó độ phức tạp của giải thuật này về mặt thời gian sẽ là O(n^2). Vậy khi chuyển sang giải thuật song song thì độ phức tạp về thời gian của giải thuật song song sẽ như thế nào?

b. Giải thuật song song.

Cho ma trận A (n x n)

Cho vec tơ *x*(n x 1)

Số bộ xử lý hệ thống sử dụng: p=n

Kết quả của việc nhân ma trận A và vec tơ *x* được lưu vào vec tơ *y*.

Để thực hiện việc nhân ma trận A và vec tơ *x*, giải thuật thực hiện qua các bước sau:

<u>Bước 1. Phân chia công việc</u>

- ✓ Phân chia ma trận

Giải thuật song song ở đây sử dụng phương pháp phân chia ma trận A thành các khối theo hàng 1 chiều (Việc phân thành các khối theo cột 1 chiều hoàn toàn tương tự.)

Ma trận A có n hàng và chúng ta có n bộ xử lý, khi đó ta phân chia mỗi hàng của ma trận A là một khối 1 chiều, và mỗi hàng đó sẽ được giao cho 1 bộ xử lý, hay nói cách khác, bộ xử lý Pi(i=0..n-1) sẽ xử lý khối thứ i trên ma trận A.

- ✓ Phân chia vec tơ *x*

Vec tơ *x* có n phần tử nằm trên n hàng, do đó, mỗi phần tử trên một hàng của vec tơ sẽ được phân cho bộ xử lý tương ứng với nó. Tức là bộ xử lý Pi(i=1..n) sẽ giữ phần tử thứ i trên vec tơ *x*.

Bước 2. Truyền thông điệp

Mỗi bộ xử lý thứ i chỉ chứa 1 phần tử thứ i của vec tơ *x*, mà trong quá trình thực hiện nhân hàng thứ i của ma trận A với vec tơ *x* thì lại cần toàn bộ các phần tử của *x* vì vậy cần có một bước để truyền tất cả phần tử trên *x* ở các bộ xử lý đến tất cả các bộ xử lý.

Bước 3. Thực hiện tính toán.

Tại mỗi bộ xử lý Pi thực hiện công việc tính toán:

$$y[i] = \sum_{j=0}^{n-1} A(i,j) * x(j)$$

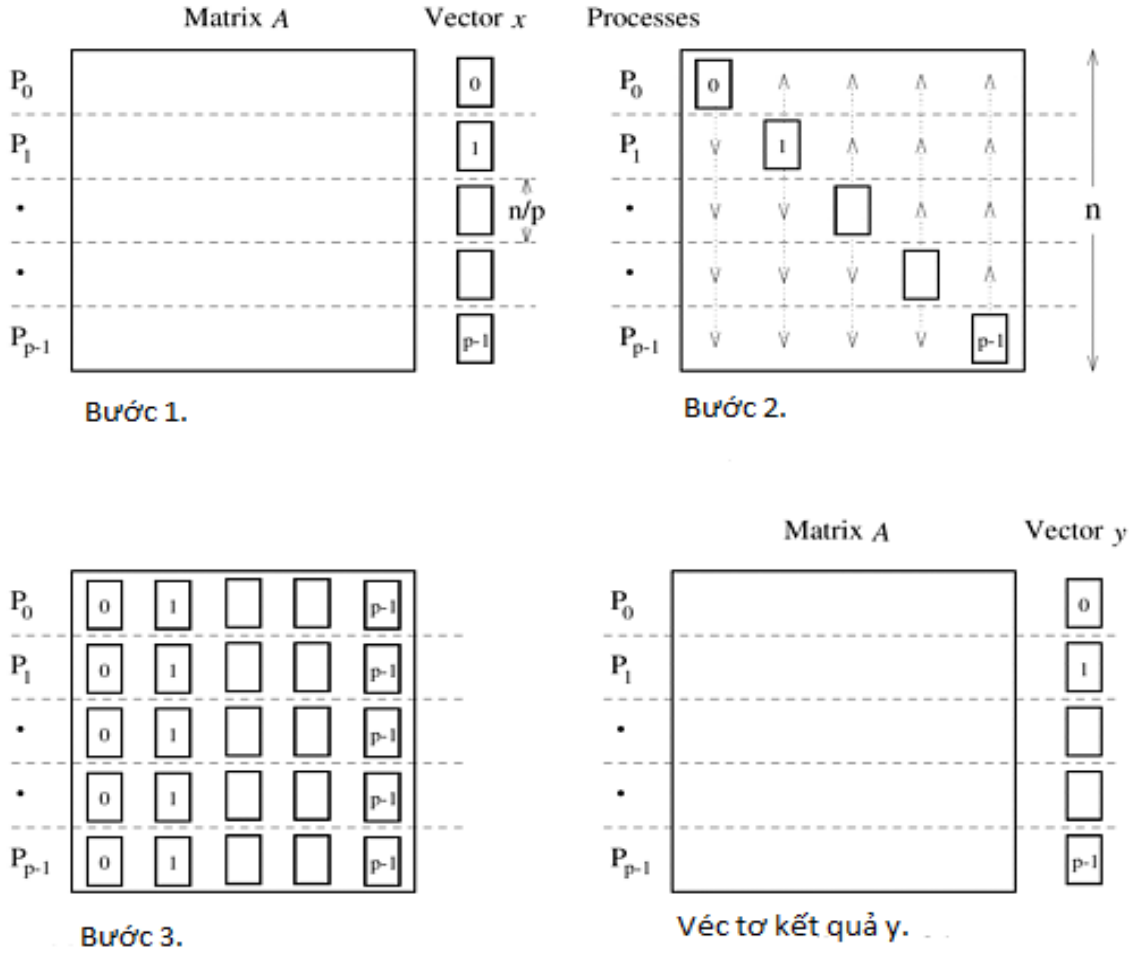
c. Đánh giá độ phức tạp của giải thuật song song.

Thời gian truyền các giá trị của vecto *x* đến tất cả các bộ xử lý: O(n)

Thời gian để thực hiện phép nhân mỗi hàng của ma trận A với vecto *x* trên mỗi bộ xử lý: O(n).

Tổng thời gian để thực hiện là: O(n)+O(n)=O(n).

Vậy thời gian thực hiện theo giải thuật song song là tối ưu hơn.



d. Mở rộng vấn đề.

Trong giải thuật trên, chúng ta sử dụng P bộ xử lý cho ma trận vuông cấp n. Bài toán đặt ra nếu số bộ xử lý P<n và ma trận của chúng ta không phải là ma trận vuông, và chúng ta không phân chia ma trận thành các khối theo hàng nữa thì chúng ta xử lý như thế nào? Và bài toán nhân hai ma trận với nhau có khác gì với nhân ma trận với véc tơ hay không?

Phần này dành cho các bạn tìm hiểu thêm trong tài liệu "**Introduction to Parallel Computing, Ananth Grama, Anshul Gupta, Geogre Karipys - Addison Wesley – 2003"** *Chương số 8- Mục 8.2*

5.6.2. Cộng hai ma trận

Xét một ma trận vuông A cấp n (ma trận có số hàng bằng số cột và bằng n) và ma trận vuông B cấp n (ma trận có số hàng bằng số cột và bằng n)

Tổng của ma trận A(n x n) và ma trận B(n x n) là một ma trận *C*(n xn).

Giải thuật tuần tự để tính tổng của 2 ma trận.

a. Giải thuật tuần tự.

Ta có C(i,j) = $\sum_{i,j=0}^{n} A(i,j) + B(i,j)$

Vì vậy, ta có giải thuật tuần tự như sau:

```
void cong_matran(A,B,C)
    {
    for (i=0; i<n; i++)
    for (j=0; j<n; j++)
    C(i,j) = A(i,j) + B(i,j)
```

Giải thuật tuần tự cần thực hiện n^2 phép cộng trong quá trình tính toán. Giả sử mỗi phép toán thực hiện tốn 1 đơn vị thời gian, khi đó độ phức tạp của giải thuật này về mặt thời gian sẽ là O(n^2).

b. Giải thuật song song

Được thực hiện tương tự như nhân vectơ với ma trận, bạn đọc tự thiết kế.

5.6.3. Bài toán SàngEratosthenes.

Sàng Eratosthenses là một giải thuật cổ xưa để lập bảng tất cả các số nguyên tố nhỏ hơn một số N cho trước.

Giải thuật dựa trên tính chất: mọi hợp số n đều có ước nguyên tố không vượt quá căn của chính nó (sqrt(N)).

Giải thuật được xây dựng như sau:

- ✓ Bắt đầu từ số nguyên tố thứ nhất t(số 2)
- ✓ Sàng bỏ các bội của số này kể từ t^2 cho đến N
- ✓ Lấy số tiếp theo còn trên sàn t_2 (số 3)

- ✓ Quay trở lại bước 2 cho đến khi t_2> N
- ✓ Các số còn lại trên sàng là số nguyên tố <N.

Ví dụ: Cho số N=1000$\sqrt{N} = 100$.

Các số nguyên tố nhỏ hơn 100 là 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31.

a. Giải thuật tuần tự.

Giải thuật tuần tự để sàn một số nguyên tố t nào đó như sau:

```
i=t*t
While i<=N
{
     San so thu i trong day;/Eratosthenes
     i=i+t;
}
```

```
Eratosthene(n)
 Var List Prime[1..n]
     Int i,j,k
 for i:=1 to n Prime[i]:=True
 Prime[1]:=false
 k=0
 while k < sqrt(n) {
    i=k+1
    while Prime[i]=False i:=i+1
    k=i
    j=2
    while k*j<=n {
         Prime[k*j]:= False
         j:=j+1
         }
   }
```

Chi phí thời gian cho giải thuật được tính theo công thức :

$$\frac{N+1-t^2}{t}$$

Như vậy ta có chi phí để sàng từng số nguyên tố như sau:

Số nguyên tố	Thời gian
2	499
3	331
5	196
7	137
11	80
13	64
17	42
19	34
23	21
29	6
31	1

Vậy chi phí của giải thuật tuần tự là 1411 đơn vị thời gian.

b. Giải thuật song song.

Chi phí cho giải thuật song song được thực hiện dựa vào số bộ xử lý tham gia.

- ✓ Có 2 bộ xử lý tham gia:
  - P_o : Sàn các bội của 2, 7, 17, 23, 29, 31
  - P_1 : Sàn các bội của 3, 5, 11, 13, 19
  - Khi đó chi phí thời gian là 706
  - Speedup = 1411/706=1.9985
- ✓ Có 3 bộ xử lý tham gia:
  - P_0: Sàn các bội của 2
  - P1: Sàn các bội của 3,11,19,29,31

- P2: Sàn các bội của 5,7,13,17,23
- Khi đó chi phí thời gian là 499
- Speedup = 1411/499=2.8276

✓ Có nhiều hơn 3 bộ xử lý tham gia:

- P_0: Sàn các bội của 2
- P1: Sàn các bội của 3
- P2: Sàn các bội của 5
- P3: Sàn các bội của 7
- ……..
- Khi đó chi phí thời gian vẫn là 499
- Speedup=1411/499=2.8276

c. Kết luận

Vậy thông qua giải thuật trên, ta có bảng kết luận sau:

Trường hợp	Số bộ xử lý	Speed up	Efficiency
1	2	1.9985	0.9993
2	3	2.8276	0.9425
3	4	2.8276	0.7069
4	5	2.8276	0.5655

Qua đó các bạn rút ra được điều gì?

5.6.4. Thuật toán tìm kiếm

Cho dãy n phần tử, giả sử chúng được lưu trữ dưới dạng mảng:
a[1], a[2], …., a[n], và các phần tử là số tự nhiên. Hãy tìm vị trí của phần tử có giá trị là x trong mảng

a. Thuật giải tuần tự

- ***Tư tưởng giải thuật***

Tiến hành so sánh x lần lượt với phần tử thứ nhất, thứ 2, …., của mảng cho đến khi gặp được phần tử có khoá cần tìm, hoặc tìm hết mảng mà không thấy x.

- Các bước tiến hành như sau:

<u>Bước 1</u>: i=1; *// bắt đầu từ phần tử đầu tiên của dãy*

<u>Bước 2</u>: So sánh a[i] với x, có 2 khả năng

- a[i]=x: Tìm thấy. Dừng
- a[i] <>x: Sang Bước 3

<u>Bước 3</u>: i=i+1; *// xét tiếp phần tử kế trong mảng*

- Nếu **i>n:** Hết mảng không tìm thấy. Dừng
- Ngược lại: Lặp lại Bước 2.

- Thuật toán

```
int  tuyentinh(int a[], int N,int x)
        {
        int i=0;
        while ((i<N)&&(a[i]!=x))    i ++;
        if (i==N) //Khong thay
                        return -1;
        else    //Thay vi tri i
                        return i;
        }
```

b. Thuật giải song song

Chuyển bài toán tuần tự thành bài toán song song, yêu cầu dùng các hàm truyền thông cộng tác để tìm ra số x trong mảng các số nguyên.

Các bước thực hiện như sau:

•Gọi np là số tiến trình chạy và có một mảng 1 chiều: int *dataSet;

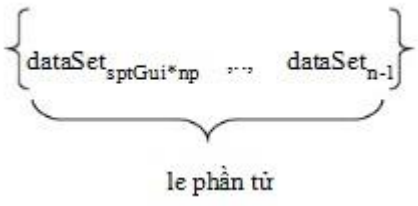
•Gọi n là size của mảng số nguyên đó.

•Gọi sptGui là số phần tử gửi cho các tiến trình ranki(i=0..np-1):

sptGui = n / np;

•Gọi le là số phần tử còn lẻ lại sau khi chia cho các tiến trình:

- le = n % np;
- rank0 sẽ giải quyết mảng có độ dài lẻ này, có chỉ số từ:



•Ta gửi các mảng con đến các rank yêu cầu tìm phần tử x (đây là giai đoạn song song của bài toán

5.6.5 Thuật toán sắp xếp.

Một trong những vấn đề nền tảng của khoa học máy tính là sắp xếp một tập các phần tử cho trước theo thứ tự nào đó. Có rất nhiều các giải pháp cho vấn đề này, được biết đến như là các thuật toán sắp xếp (sorting algorithms). Bên cạnh các thuật toán sắp xếp đơn giản và rất rỏ ràng, như là sắp xếp nổi bọt (bubble sort). Một số khác, như phương pháp sắp xếp nhanh (Quick Sort) thì lại rất phức tạp nhưng đổi lại có kết quả thực thi nhanh hơn một cách đáng kể.

Hai nhóm thuật toán sắp xếp được phân như sau: nhóm thứ nhất có độ phức tạp là O(n^2) bao gồm pilot, bubble, insertion, selection, và shell sorts; Nhóm thứ hai có độ phức tạp là O(n log n) gồm heap, merge, và quick sorts.

Trong giới hạn của môn học, chúng ta chỉ xem xét một vài thuật toán sắp xếp điển hình để chuyển từ giải thuật sắp xếp tuần tự sang giải thuật sắp xếp song song. Các thuật toán khác dành cho các bạn tìm hiểu thêm.

5.6.5.1 Thuật toán sắp xếp theo phần tử hoa tiêu

Một trong những thuật toán sắp xếp đơn giản nhất là thuật toán sắp xếp theo phần tử hoa tiêu (Pilot hay còn gọi là sắp xếp theo cách đánh số). Trong cách sắp

xếp này, số những số nhỏ hơn một số đã chọn sẽ được đếm. Số đếm được sẽ xác định vị trí của phần tử đã được chọn trong danh sách cần sắp xếp. (Tức là muốn biết số a nằm ở vị trí nào trong mảng thì ta đếm xem trong mảng đó có bao nhiêu số lớn hơn hoặc nhỏ hơn số a, dựa vào đó để biết được vị trí của a trong mảng.) Giả sử có n số được lưu giữ trong mảng A[n] và kết quả sau khi sắp xếp là mảng B[n].

a. Thuật toán tuần tự

```
For (i=0; i<n; i++)
     x=0;
     for (j=0; j<n; j++)
          if (A[i] > A[j])  x++;
     B[x]= A[i];
```

Qua giải thuật trên ta thấy độ phức tạp của giải thuật tuần tự là O(n^2)

b. Thuật toán song song

Mảng A[n], có P bộ xử lý P=n; Các bộ xử lý được đánh số P_o, P_1,….. P_{n-1} đều được cấp dãy số A[n] cho trước và bộ xử lý P_i phải tìm vị trí của nó dựa vào giá trị chứa trong A[i] tương ứng trên dãy số cần được sắp xếp. Thuật toán song song được viết như sau:

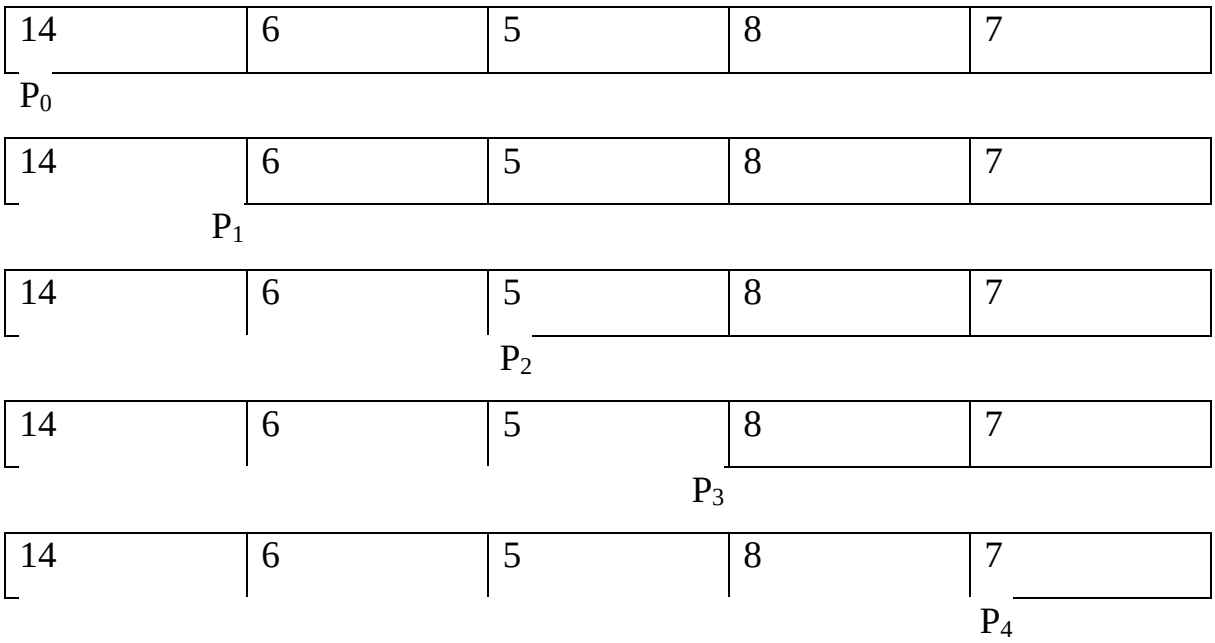
```
Forall (i=0; i<n; i++)          // n bộ xử lý song song
     x=0;
     for (j=0;    j<n; j++)     // Đếm số phần tử nhỏ hơn
          if (A[i] > A[j])x++;
     B[x] =a[i]                 // Sao sang mảng mới
```

Ta thấy tất cả n bộ xử lý thự hiện song song nên thuật toán để sắp xếp mảng A[n] với n bộ xử lý có độ phức tạp tuyến tính O(n).

c. Ví dụ.

Cho mảng A[5]={14, 6, 5, 8, 7}

Với số bộ xử lý là P=5.



Trong bộ xử lý P_0 số nó cần sắp xếp lại là 14, có 4 số nhỏ hơn 14. Vậy nó xếp thứ 4.

Trong bộ xử lý P_1 số nó cần sắp xếp lại là 6, có 1 số nhỏ hơn 6. Vậy P_1 xếp số 1.

Trong bộ xử lý P_2 số nó cần sắp xếp lại là 5, có 0 số nhỏ hơn 5. Vậy P_0 xếp số 0.

Trong bộ xử lý P_3 số nó cần sắp xếp lại là 8, có 3 số nhỏ hơn 8. Vậy P_3 xếp số 3.

Trong bộ xử lý P_4 số nó cần sắp xếp lại là 7, có 2 số nhỏ hơn 7. Vậy P_4 xếp số 2.

Một cách hình thức có thể thu được kết quả như sau:

5	6	7	8	14
P_2	P_1	P_3	P_4	P_0

5.6.5.2 Thuật toán sắp xếp so sánh và đổi chỗ.

Thuật toán này so sánh hai phần tử liền kề nhau và nếu chúng chưa theo thứ tự cần sắp xếp thì đổi chỗ chúng cho nhau. Quá trình này lặp lại cho đến khi không còn cặp nào không thỏa mãn thì dừng lại.

a. Thuật toán tuần tự.

```
For (i=n-1; i<n; i--)
    For (j=0; j<n; j++)
        {
            k=j+1;
            if (A[j] > A[k])
                {
                    temp=A[j];
                    A[j]=A[k];
                    A[k]=temp;
                }
        }
```

Thông qua giải thuật, ta thấy độ phức tạp tính toán của giải thuật tuần tự là O(n^2)

b. Thuật toán song song

Dùng n tác vụ kết hợp theo nguyên lý hình ống để sắp xếp mảng A[n]. Hệ thống được chia thành 2 pha: Pha chẵn và pha lẻ.

- ✓ Pha chẵn: Các tác vụ được đánh số chẵn so sánh với những tác vụ tiếp theo nó(tác vụ có số lẻ), nếu nó giữ phần tử lớn hơn thì đổi chỗ dữ liệu với tác vụ đó.
- ✓ Pha lẻ: Các tác vụ đánh số lẻ so sánh với những tác vụ tiếp theo nó (tác vụ có số chẵn), nếu nó giữ phần tử lớn hơn thì đổi chỗ dữ liệu với tác vụ đó.

Thuật toán song song có độ phức tạp O(n). (Vì sao lại có độ phức tạp O(n), coi như một bài tập nhỏ về nhà cho các bạn cùng tìm hiểu.)

c. Ví dụ

Cho A[8]={4, 2, 7, 8, 5, 1, 3, 6}. Sắp xếp lại mảng A theo giải thuật song song sử dụng 8 bộ xử lý đánh số Pi (i=0..8).

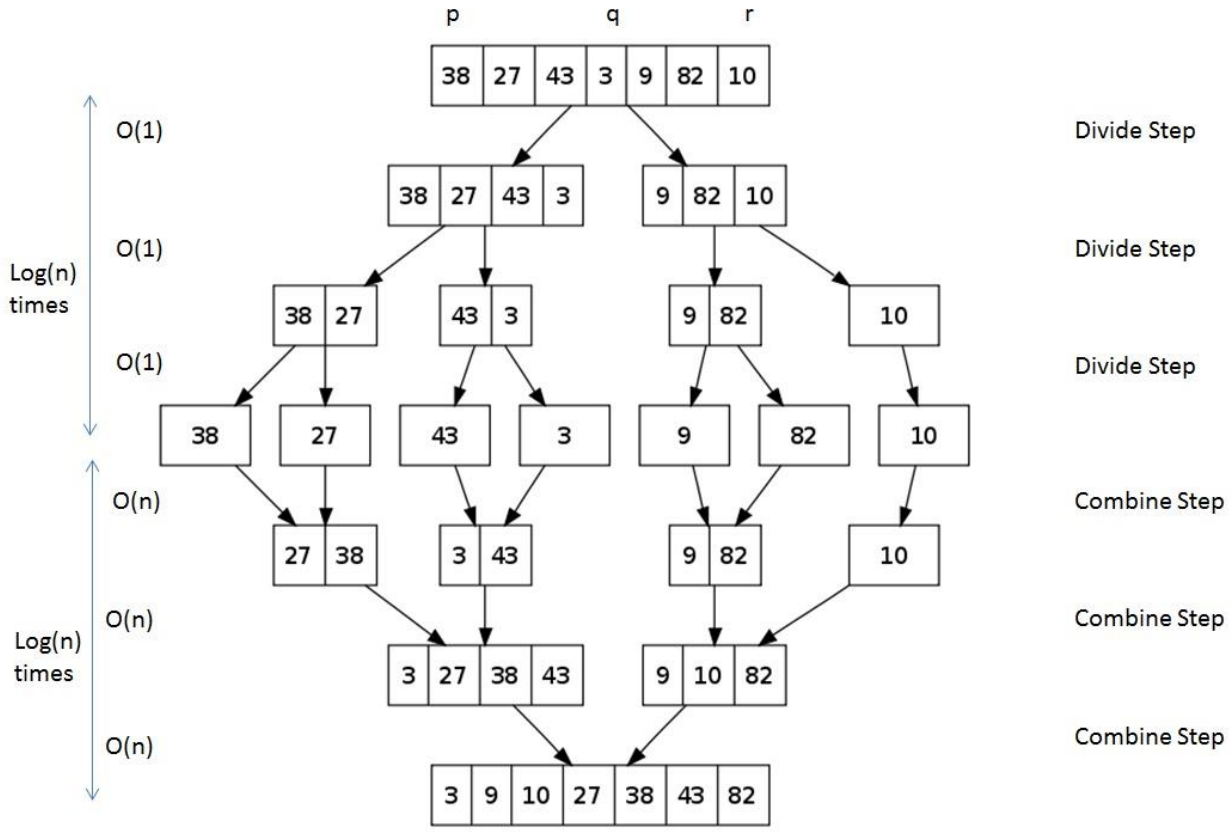
Tác vụ / Pha	P_0		P_1		P_2		P_3		P_4		P_5		P_6		P_7
0	4	↔	2		7	–	8		5	↔	1		3	–	6
1	2		4	–	7		8	↔	1		5	↔	3		6
2	2	–	4		7	↔	1		8	↔	3		5	–	6
3	2		4	↔	1		7	↔	3		8	↔	5		6
4	2	↔	1		4	↔	3		7	↔	5		8	↔	6
5	1		2	–	3		4	–	5		7	↔	6		8
6	1	–	2		3	–	4		5	–	6		7	–	8
7	1		2	–	3		4	–	5		6	–	7		8

5.6.5.3 Thuật toán Merge Sort

MergeSort là thuật toán "Chia để trị" (Nguyên lý chia để trị được thể hiện bằng cách chia một bài toán thành những bài toán con sao cho chúng có dạng giốn như bài toán lớn. Vì vậy, phương pháp chia để trị thường có dạng đệ quy.)

Một danh sách chưa được sắp xếp được chia thành 2 phần. Mỗi phần lại được chia đôi tiếp theo cho đến khi mỗi phần nhỏ chỉ còn một phần tử. Sau đó, từng cặp được ghép lại theo thứ tự cần xắp xếp. Quá trình ghép làm ngược lại quá trình phân rã và tiếp tục được ghép để cuối cùng thu được dãy được sắp xếp.

Cách chia đôi các phần tử như trong thuật toán MergeSort sẽ tạo ra cấu trúc cây nhị nguyên. Nếu n là lũy thừa của 2 thì chúng ta được cây cân bằng.



Total Runtime = Total time required in Divide + Total time required in Combine
= 1 * Log(n) + n * Log(n) = n Log(n).

Độ phức tạp của thuật toán tuần tự là O(nlogn)

Độ phức tạp thời gian tính toán của giải thuật song song sử dụng n bộ xử lý là O(n).

Do giới hạn của chương trình chúng tôi không thể trình bày cụ thể giải thuật. Coi như một bài tập lớn dành cho các bạn!

5.6.5.4 Mở rộng

Trong mảng bài toán sắp xếp, chúng ta còn rất nhiều thuật toán khác như Quicksort, RadixSort, InsertSort, BioticSort, BubbleSort…. Các bạn có thể tìm hiểu thêm các thuật toán đó trong tài liệu "**Introduction to Parallel Computing, Ananth Grama, Anshul Gupta, Geogre Karipys - Addison Wesley – 2003"** *Chapter 9.Sorting.*

5.6.6. Tìm phần tử lớn nhất trên ma trận

Viết chương trình tìm phần tử có giá trị lớn nhất trong một ma trận cho trước. Đầu vào là một tệp tin. Dòng dầu tiên của tệp tin gồm 2 phần tử cách nhau bởi khoảng trắng, thể hiện số dòng và số cột của ma trận. Các dòng tiếp theo là các dòng trong ma trận. Các phần tử trong một dòng cách nhau bởi khoảng trắng trong dãy. Dòng thứ hai của tệp tin là danh sách phần tử trong dãy, các phần tử cách nhau bởi một khoảng trắng. Sử dụng hàm truyền thông cộng tác

5.6.6.1. Giải thuật song song

Giải quyết vấn đề chuyển bài toán tuần tự thành bài toán song song, yêu cầu dùng các hàm truyền thông cộng tác để tìm ra số Max trong mảng các số nguyên.

Giải quyết vấn đề trên như sau:

•Gọi np là số tiến trình chạy, và có một mảng 1 chiều: int *dataSet;

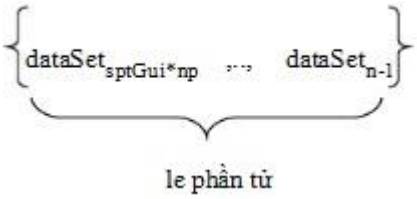
•Gọi n là size của mảng số nguyên đó.

•Gọi sptGui là số phần tử gửi cho các tiến trình ranki(i=0..np-1):

$$sptGui = n / np;$$

•Gọi le là số phần tử còn lẻ lại sau khi chia cho các tiến trình:

- le = n % np;
- rank0 sẽ giải quyết mảng có độ dài lẻ này, có chỉ số từ:



•Ta gửi các mảng con đến các rank yêu cầu tính Max (đây là giai đoạn song song của bài toán

Mô tả giải thuật song song cho bài toán

→Bắt đầu:

•Đọc file “dulieu.in” với ma trận int (n x m), đưa các dữ liệu vào một mảng một chiều dataSet có độ dài(total): total = n * m.

•Cài đặt hàm tìm max:int maxFind(int *array,int size); dùng chung cho các tiến trình để tìm max.

•Gởi Broadcast số phần tử của mảng đến các rank:

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

•Gởi mảng buff tấc cả các rank:

MPI_Scatter(dataSet,sptGui,MPI_INT,buff,sptGui,MPI_INT,0,MPI_COMM_WORLD);

•Các tiến trình bắt đầu làm việc:

- R a n k 0 :

+ Tính max phần mảng mình đảm nhiệm(max0):

max0=maxFind(buff,sptGui);

+ Nếu le >0 tính max của mảng lẻ sau khi chia đủ các rank:

maxCon=maxFind(dataCon,le);

if(maxCon>max0)max0=maxCon;

-Rank(i):

+ Nhận các dữ liệu: sptGui (kích thước mảng), buff

+ Tính các max(i) ở các rank:

max=maxFind(buff,sptGui);

+ Gửi về cho rank0 số maxI đó bằng hàm:

MPI_Reduce(&max,NULL,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);

•Rank0 bắt đầu tổng kết để nhận kết quả max trả về từ các Rank, sosánh với max0 của mình, rồi in kết quả max tìm được ra màn hình:

MPI_Reduce(&max0,&max,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);

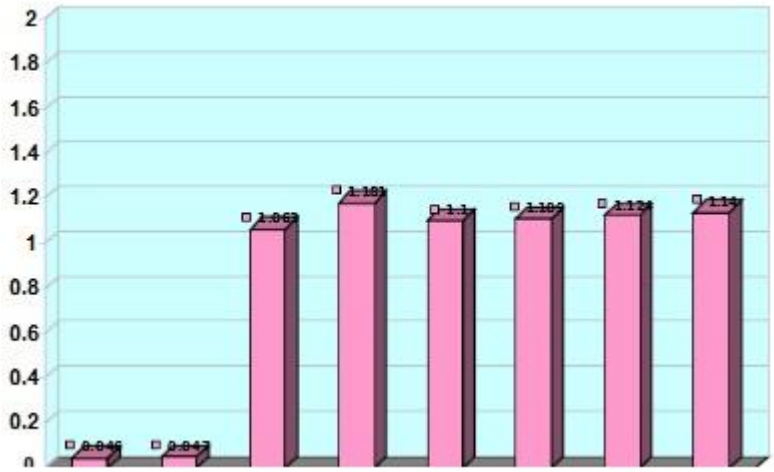
→Kết thúc.

5.6.6.2. Đánh giá thời gian chạy của chương trình khi số lượng CPU thay đổi

➢ Biểu đồ thời gian chạy chương trình:

- Ma trận vào 5x5:

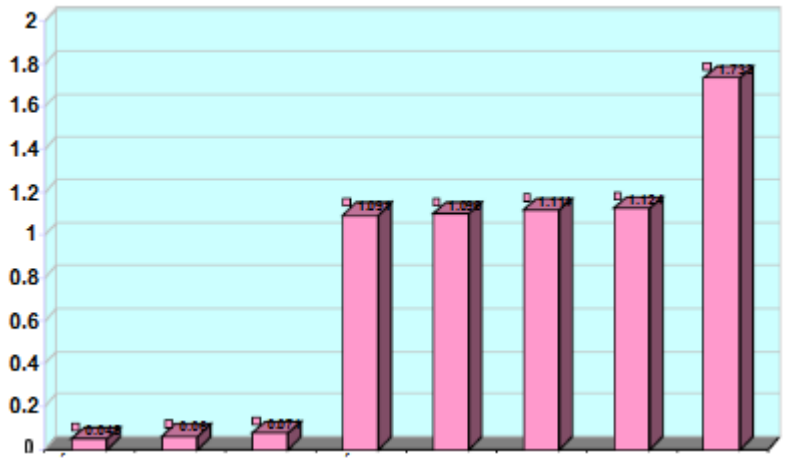
Số CPU	Thời lượng(s)
1	0.043
2	0.047
3	1.063
4	1.181
5	1.1
6	1.109
7	1.124
8	1.14



Đánh giá: Các bạn sinh viên dựa vào bảng số liệu trên đối với đầu vào ma trận 5x5 thì số lượng CPU là bao nhiêu là phù hợp với chương trình?

➢ Ma trận vào 40x40:

Số CPU	Thời lượng(s)
1	0.045
2	0.06
3	1.071
4	1.091
5	1.096
6	1.114
7	1.124
8	1.733



Đánh giá: Các bạn sinh viên dựa vào bảng số liệu và biểu đồ trên đối với đầu vào ma trận 40x40 thì số lượng CPU là bao nhiêu là phù hợp với chương trình?

5.6.6.3. Kết luận và nhận xét

-Mỗi chương trình có một số lượng CPU nhất định để khi chạy đạt hiệu quả cao nhất.

-Đối với những chương trình nhỏ như thế này thì chỉ cần 1 hoặc 2 CPU là thích hợp.

-Còn với những chương trình lớn thì cần số lượng lớn CPU hơn nhưng cũng chỉ một giới hạn nào đó. Không phải càng nhiều thì càng tốt.

5.6.6.4. Chương trình tham khảo

```
//truyen thong cong tac
#include<mpi.h>
#include<stdio.h>
#include<stdlib.h>
int Rank0();
int Ranki();
int main(int argc, char **argv)
    {
    int rank;
```

```
        MPI_Init(&argc,&argv);
        MPI_Comm_rank(MPI_COMM_WORLD,&rank);
        if(rank==0)Rank0();
        else Ranki();
        MPI_Finalize();
        return 0;
        }


//ham tim so max cua mot mang
int maxFind(int *array,int size)
    {
    int i,max=-32768;
    for(i=0;i<size;i++)
        if(array[i]>max) max=array[i];
    return max;
    }
```

```
//ham cua Rank 0
int Rank0()
      {
            int total;  //tong so phan tu mang can tim max
            int *dataSet,*buff;     //du lieu mang
            int *dataCon;     //mang data con du khi chia cho cac
tien trinh
            int *dataClient;//mang kq tra ve cac max ma tien
trinh tim dc
            int np;           // so proccess
            int sptGui; //so phan tu trong mang con cua mang
dataSet, de gui cho cac proccess tim max
            int max,maxCon,maxClients,max0;
            int le,i,j; //bien le : le=total%np;

            MPI_Comm_size(MPI_COMM_WORLD,&np);//lay ve so tien
trinh
            dataSet=loadData(&total);
            if(np==0){printf("error process=0");return 0;}

            sptGui=total/np;
```

```
                  buff=(int*)malloc(sizeof(int)*sptGui);
                  le=total%np;
                  //gui cac mang con di den cac tien trinh
                  if(np>=2)
                        {

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


     MPI_Scatter(dataSet,sptGui,MPI_INT,buff,sptGui,MPI_INT,0,MPI_COMM_WORLD
     );
```

```
      // rank0 se lam cong viec cua minh : tim max tu 0 den(sptGui-1)
      va mang con du khi chia deu cho cac tien trinh
                  if(np>=2)max0=maxFind(buff,sptGui);
                        else max=maxFind(dataSet,total);
                  if(le>0)
                        {
                              dataCon=(int*)malloc(sizeof(int)*le);
                              for(j=0;j<le;j++)
                                    dataCon[j]=dataSet[np*sptGui+j];
                              maxCon=maxFind(dataCon,le);
                              if(maxCon>max0)max0=maxCon;
                        }
                        //nhan du lieu cua cac tien trinh con gui ve va
      tinh Max


   if(np>=2)MPI_Reduce(&max0,&max,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);

                  printf("Ket qua Max la : %d\n",max);
                  free(dataSet);
```

//ham cac tien trinh con

```
int Ranki()
      {
            int n;//so pt mang
            int *array;
            int max;
            //MPI_Status status;
            //nhan du lieu
                  MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);//nhan
mot so tu tien trinh rank0, chinhla kichthuoc mang
            //khai bao cap phat dong
            array=(int*)malloc(sizeof(int)*n);

MPI_Scatter(NULL,n,MPI_INT,array,n,MPI_INT,0,MPI_COMM_WORLD);
            //tim max
            max=maxFind(array,n);
            // gui lai cho rank 0 so max tim dc

MPI_Reduce(&max,NULL,1,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);
            return 0;
      }
```

5.6.7. Bài toán xếp balo

Trong môn Cấu trúc dữ liệu và giải thuật, chúng ta đã được làm quen với bài toán xếp balo. Đây là một bài toán rất hay và có ý nghĩa. Bài toán được phát biểu như sau:

Cho một dãy các đồ vật, mỗi đồ vật có 2 thông số là giá trị và thể tích của đồ vật đó. Bài toán đặt ra, chúng ta có một chiếc balo có thể tích là C. Yêu cầu chúng ta phải lựa chọn các đồ vật sao cho tổng giá trị của đồ vật lấy được là lớn nhất nhưng tổng thể tích của các đồ vật lấy được phải nhỏ hơn hoặc bằng thể tích của balo.

Để giải quyết bài toán này chúng ta đã có rất nhiều giải thuật tuần tự để làm việc với nó, điển hình là giải thuật sử dụng chiến lược tham lam.(Sắp xếp các đồ vật theo thứ tự, lấy giá trị của đồ vật thứ i chia cho thể tích của đồ vật thứ i. Sau đó dựa vào tỉ số đó, lần lượt chọn các đồ vật có tỉ số vừa tính theo chiều từ cao xuống thấp đến bao giờ đầy balo thì thôi.) Giải thuật này rất ngắn gọn, dễ hiểu và dễ cài đặt. Tuy nhiên, không phải lúc nào cũng đưa ra được kết quả tối ưu trong khi bài toán có thể đưa ra được kết quả tối ưu.

Ví dụ: cho 6 đồ vật với các giá trị tương ứng về thể tích và giá trị như sau:

$$n = 6 \text{ and } C = 16,$$

$$(w_1, ..., w_n) = (5, 3, 2, 1, 5, 9),$$

$$(p_1, ..., p_n) = (20, 8, 5, 4, 14, 27).$$

Với giải thuật trình bày ở trên, tỉ số lần lượt của các đồ vật là :

(20/5, 8/3, 5/2, 4/1, 14/5, 27/9)=(4, 2.7, 2.5, 4, 2.8, 3)

Và kết quả ta được các đồ vật sẽ được chọn là đồ vật 1, 4, 6 tổng thể tích là 15 và tổng giá trị là 51.

Nhưng trên thực tế thì tổng thể tích là 16 và giá trị 52 mới là kết quả tối ưu.

Vì vậy dưới đây tôi giới thiệu một thuật toán mới cho bài toán xếp balo để chúng ta có thể tìm được một kết quả tối ưu hơn giải thuật trên và chúng ta cùng xem xét giải thuật ở dạng tuần tự và song song.

5.6.7.1 Thuật toán tuần tự sử dụng phần tử ưu thế cho bài toán xếp balo 0/1.

a. Phát biểu bài toán:

Tìm Max$\left\{\sum_{j=1}^{n} p_j x_j \mid \sum_{j=1}^{n} w_j x_j \leq C\ ;\ x_j \in \{0,1\}, j = 1,2,\dots,n\right\}$

Trong đó:

p_j: Giá trị của đồ vật thứ j

w_j: Thể tích của đồ vật thứ j

x_j=0: Đồ vật thứ j không được chọn

x_j=1: Đồ vật thứ j được lựa chọn

C: tổng thể tích của balo.

Để tránh trường hợp nhỏ lẻ sảy ra ta giả sử luôn có: : $\sum_{j=1}^{n} w_j > C$ và w_j<C với j $\in$ {1,.., n}.(Các bạn thử giải thích tại sao?)

b. Giải thuật tuần tự

Trong giải thuật này, chúng ta sử dụng định nghĩa cặp phần tử ưu thế.

Cho 2 cặp (w,p) và (w',p') cặp (w',p') được gọi là có ưu thế hơn so với cặp (w,p) nếu sảy ra: w ≥ w' nhưng p<p'.

Dựa vào định nghĩa đó, chúng ta xây dựng giải thuật tuần tự để giải quyết bài toán này như sau:

Để giải quyết bài toán này, ta phải trải qua n bước lặp. Tại mỗi bước lặp thứ k, ta lần lượt đi xây dựng các tập: L_k, N_k, D_k .

Xây dựng tập L_k : Là tập chứa các phần tử chiếm ưu thế tại bước lặp thứ k(k=0..n). Trong danh sách L_k chỉ chứa các cặp đơn điệu tăng tức là (w,p),(w',p') thuộc L_k thì bắt buộc phải sảy ra w<w' và p<p'. Khởi tạo tập L_0={(0,0)}.

Xây dựng tập N_k là tập các phần tử sinh tại bước lặp thứ k.

$$N_k = \{(w + w_k, p + p_k) \mid (w,p) \in L_{k-1},\ w + w_k \leq C\}.$$

Xây dựng tập D_k: Tập chứa các phần tử không tối ưu tại bước lặp thứ k.

$$D_k = \{(w,p) \mid (w,p) \in L_{k-1} \cup N_k, \exists (w',p') \in L_{k-1} \cup N_k \text{ với } w' \leq w, p \leq p', (w',p') \neq (w,p)\}.$$

Từ đó ta suy ra tập các phần tử tối ưu tại bước lặp thứ k là:

$$L_k = L_{k-1} \cup N_k - D_k.$$

Do trong danh sách L_k các phần tử được sắp sếp theo dãy đơn điệu tăng, vì vậy tại bước lặp cuối cùng tức tại bước lặp thứ n thì phần tử nằm cuối cùng danh sách sẽ là phần tử tối ưu nhất của bài toán xếp balo.

Ví dụ:

$$n = 6 \text{ and } C = 16,$$

$$(w_1, ..., w_n) = (5, 3, 2, 1, 5, 9),$$

$$(p_1, ..., p_n) = (20, 8, 5, 4, 14, 27).$$

Vòng lặp k	Danh sách	Phần tử danh sách
0	L_0	(0,0)
1	N_1	(5, 20)
	D_1	

	L_1	(0, 0); (5, 20)
2	N_2	(3, 8); (8, 28)
	D_2	
	L_2	(0, 0); (3, 8); (5, 20); (8, 28)
3	N_3	(2, 5); (5,13); (7; 25); (10, 33)
	D_3	(5, 13)
	L_3	(0, 0); (2, 5); (3, 8); (5, 20); (7, 25); (8, 28); (10, 33)
4	N_4	(1, 4); (3, 9); (4, 12); (6, 24); (8, 29); (9, 32); (11, 37)
	D_4	(3, 8); (8,28)
	L_4	(0, 0); (1, 4); (2,5); (3, 9); (4, 12); (5, 20); (6, 24); (7, 25); (8, 29); (9, 32); (10, 33); (11, 37)
5	N_5	(5, 14); (6, 18); (7, 19); (8, 23); (9, 26); (10, 34); (11, 38); (12, 39); (13, 43); (14, 46); (15, 47); (16, 51)
	D_5	(5, 14); (6, 18); (7, 19); (8, 23); (9, 26); (10, 33); (11, 37)
	L_5	(0, 0); (1, 4); (2, 5); (3, 9);(4, 12);(5, 20); (6, 24); (7, 25); (8, 29); (9, 32); (10, 34); (11, 38); (12, 39); (13, 43); (14, 46); (15, 47); (16; 51)
6	N_6	(9, 27); (10, 31); (11, 32); (12, 36); (13, 39); (14, 47); (15, 51); (16, 52)
	D_6	(9, 27); (10, 31); (11, 32); (12, 36); (13, 39); (14, 46); (15, 47); (16, 51)
	L_6	(0, 0); (1, 4); (2, 5); (3, 9); (4, 12); (5, 20); (6, 24); (7, 25); (8, 29); (9, 32); (10, 34); (11, 38); (12, 39); (13, 43); (14, 47); (15, 51); (16, 52)

Và ta được (16, 52) chính là kết quả tối ưu cần tìm.

5.6.7.2 Thuật toán song song sử dụng phần tử ưu thế cho bài toán xếp balo 0/1.

Tìm $\text{Max}\left\{\sum_{j=1}^{n} p_j x_j \mid \sum_{j=1}^{n} w_j x_j \leq C\ ;\ x_j \in \{0,1\}, j = 1,2,\dots,n\right\}$

Trong đó:

p_j: Giá trị của đồ vật thứ j

w_j: Thể tích của đồ vật thứ j

x_j=0: Đồ vật thứ j không được chọn

x_j=1: Đồ vật thứ j được lựa chọn

C: tổng thể tích của balo.

Hệ thống sử dụng q bộ xử lý (0..q-1)

Để giải quyết bài toán này, ta định nghĩa một số tập sau

L_k^i: tập các phần tử chiếm ưu thế tại vòng lặp thứ k ở bộ xử lý thứ i

N_k^i: tập các phần tử sinh tại vòng lặp thứ k ở bộ xử lí thứ i

C_k^i: tập các phần tử trao đổi ở bộ xử lý thứ i tại vòng lặp k

D_k^i: tập các phần tử không chiếm ưu thế tại vòng lặp k ở bộ xử lý thứ i

1. Phân chia công việc cho các bộ xử lý.
   a. Chọn ra vòng lặp thứ k theo thứ tự tuần tự để phân chia công việc cho các bộ xử lý.
   b. Vòng lặp thứ k được chọn là vòng lặp đảm bảo được điều kiện:

      $|L_k^i|$ = l với i=1..p-1, và tại bộ xử lí đầu tiên: $|L_k^0| \geq$ l tức là vòng lặp thứ k trong bước làm tuần tự được chọn ra phải đảm bảo được rằng khi chia các cặp trong danh sách L_k cho các bộ xử lí thì mỗi bộ xử lý từ 1..p-1 phải có số phần tử bằng nhau và bằng l, còn dư bao nhiêu cặp thì chúng ta cho và bộ xử lí đầu tiên(bộ xử lý 0)

2. Xử lý song song

   Tại mỗi bộ xử lý luôn tồn tại một cặp $(w_k^{i,0}, p_k^{i,0})$ là cặp có giá trị nhỏ nhất cả về thể tích và giá trị tại bộ xử lý thứ i ở vòng lặp k.

   Bây giờ ta lần lượt đi xây dựng các tập được định nghĩa ở trên:

   a. N_k^i: tập các phần tử sinh tại vòng lặp thứ k ở bộ xử lí thứ i

      ✓ Với bộ xử lí từ 0..q-2

$$N_k^i = \{(w_{k-1}^{i,.} + w_k, p_{k-1}^{i,.} + p_k) \mid (w_{k-1}^{i,.}, p_{k-1}^{i,.}) \in L_{k-1}^i, w_{k-1}^{i,.} + w_k < w_{k-1}^{i+1,0}\}$$

      ✓ Với bộ xử lí q-1

$$N_k^{q-1} = \{(w_{k-1}^{q-1,\cdot} + w_k, p_{k-1}^{q-1,\cdot} + p_k) \mid (w_{k-1}^{q-1,\cdot}, p_{k-1}^{q-1,\cdot}) \in L_{k-1}^{q-1}, w_{k-1}^{q-1,\cdot} + w_k \leq C\}$$

b. C_k^i: tập các phần tử trao đổi ở bộ xử lý thứ i tại vòng lặp k

- ✓ Với bộ xử lí i=1..q-2

$$C_k^i = \{(w_{k-1}^{j,\cdot} + w_k, p_{k-1}^{j,\cdot} + p_k) \mid (w_{k-1}^{j,\cdot}, p_{k-1}^{j,\cdot}) \in L_{k-1}^j, j < i,$$
$$w_{k-1}^{i,0} \leq w_{k-1}^{j,\cdot} + w_k < w_{k-1}^{i+1,0} \text{ or } w_{k-1}^{j,\cdot} + w_k < w_{k-1}^{i,0}, p_{k-1}^{j,\cdot} + p_k \geq p_{k-1}^{i,0}\}.$$

- ✓ Với bộ xử lí 0

$$C_k^0 = \emptyset \; \forall \; k$$

- ✓ Với bộ xử lí q-1

$$C_k^{q-1} = \{(w_{k-1}^{j,\cdot} + w_k, p_{k-1}^{j,\cdot} + p_k) \mid (w_{k-1}^{j,\cdot}, p_{k-1}^{j,\cdot}) \in L_{k-1}^j, j < q-1,$$
$$w_{k-1}^{q-1,0} \leq w_{k-1}^{j,\cdot} + w_k < C \text{ or } w_{k-1}^{j,\cdot} + w_k < w_{k-1}^{q-1,0}, p_{k-1}^{j,\cdot} + p_k \geq p_{k-1}^{q-1,0}\}.$$

c. D_k^i: tập các phần tử không chiếm ưu thế tại vòng lặp k ở bộ xử lý thứ i

- ✓ Tại các bộ xử lí 0..p-2

$$D_k^i = \hat{D}_k^i \cup \{(w,p) \mid w < w_{k-1}^{i+1,0} \text{ and } p \geq p_{k-1}^{i+1,0}\},$$

$$\hat{D}_k^i = \{(w,p) \mid (w,p) \in L_{k-1}^i \cup N_k^i \cup C_k^i \text{ and } \exists (w',p') \in L_{k-1}^i \cup N_k^i \cup C_k^i$$
$$(w',p') \neq (w,p) \text{ and } w' \leq w, p \leq p'\},$$

- ✓ Tại bộ xử lý p-1

$$D_k^{q-1} = \{(w,p) \mid (w,p) \in L_{k-1}^{q-1} \cup N_k^{q-1} \cup C_k^{q-1}, \exists (w',p') \in L_{k-1}^{q-1} \cup N_k^{q-1} \cup C_k^{q-1}$$
$$(w',p') \neq (w,p), w' \leq w, p \leq p'\},$$

d. L_k^i: tập các phần tử chiếm ưu thế tại vòng lặp thứ k ở bộ xử lý thứ i

$$L_k^i = L_{k-1}^i \cup N_k^i \cup C_k^i - D_k^i.$$

Phần tử cuối cùng trong danh sách L_n^{q-1}chính là phần tử tối ưu mà bài toán xếp balo cần tìm.

Ví dụ: Với bài toán trên ta được vòng lặp được chọn để phân công công việc cho các bộ xử lý là k=2

i		0	1	2
Bước k	Danh sách			
2	L_2^i	(0, 0); (3, 8)	(5, 20)	(8, 28)
3	N^i_3	(2, 5)	(7, 25)	(10, 33)
	C_3^i		(5, 13)	
	D_3^i		(5, 13)	
	L_3^i	(0, 0); (2,5); (3, 8)	(5, 20); (7, 25);	(8, 28); (10, 33)
4	N^i_4	(1, 4); (3, 9); (4, 12)	(6, 24)	(9, 32); (11, 37)
	C_3^i			(8, 29)
	D_4^i	(3, 8)		(8, 28)
	L_4^i	(0, 0); (1, 4); (2, 5); (3, 9); (4, 12)	(5, 20); (6, 24); (7, 25)	(8, 29); (9, 32); (10, 33); (11, 37)
5	N^i_5			(13, 43); (14, 46); (15, 47); (16; 51)
	C_5^i		(5, 14); (6, 18); (7, 19)	(8, 23); (9, 26); (10, 34); (11, 38); (12, 39)
	D_5^i		(5, 14); (6, 18); (7, 19)	(8, 23); (9, 26); (10, 33); (11, 37)
	L_5^i	(0, 0); (1,4); (2, 5); (3, 9); (4, 12)	(5, 20); (6, 24); (7, 25)	(8, 29); (9, 32); (10, 34); (11, 38); (12, 39); (13, 43); (14, 46); (15, 47);

				(16, 51)
6	N^i_6			
	C_6^i			(9, 27); (10, 31); (11, 32); (12, 36); (13, 39); (14, 47); (15, 51); (16, 52)
	D_6^i			(9, 27); (10, 31); (11, 32); (12, 36); (13, 39); (14, 46); (15, 47); (16, 51)
	L_6^i	(0, 0); (1,4); (2, 5); (3, 9); (4, 12)	(5, 20); (6, 24); (7, 25)	

Tài liệu của môn học:

- Đoàn văn Ban, Nguyễn Mậu Hân, *Xử lý song song và phân tán*, NXB KH&KT, 2009.
- Introduction to Parallel Computing, Ananth Grama, Anshul Gupta, Geogre Karipys - Addison Wesley - 2008
- M. Sasikumar, Dinesh Shikhare, P. Ravi Prakash, *Introduction to Parallel Processing*, Prentice -Hall, 2000
- "*Parallel Computing – theory and practice*", Michael J. Quinn, McGRAW-HILL, 1994.

Introduction to Parallel computing

http://www.llnl.gov/computing/tutorials/parallel_comp/index.html

IBM Parallel Enviroment Manuals

http://www_1.ibm.com/servers/eserver/pseries/library/sp_books

MPI Tutorial http://www.llnl.gov/computing/mpi

Programming with POSIX pthreads

http://www.awl.com/cseng/titles/0-201-63392-2

POSIX pthreads programming

http://www.llnl.gov/computing/tutorials/pthreads